



Πανεπιστήμιο Δυτικής Αττικής  
Σχολή Τεχνολογικών Εφαρμογών  
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

## ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

# Καινοτόμες προσεγγίσεις για την διαχείριση κύκλου ζωής των συστημάτων λογισμικού [CODE EVERYTHING]

**ΚΑΤΣΙΜΠΡΑΣ Κ. ΔΡΟΣΟΣ**  
cs13110

Επιβλέπων:

**Καρκαζής Παναγιώτης**  
Αναπληρωτής Καθηγητής

Αιγάλεω - Αθήνα, Ιούλιος, 2024



Εξετάστηκε την 29/07/2024

**Παναγιώτης Καρκαζής**  
Αναπληρωτής Καθηγητής

**Μυριδάκης Νικόλαος**  
Αναπληρωτής Καθηγητής

**Μαυρομάτης Κωνσταντίνος**  
Λέκτορας

Αιγάλεω - Αθήνα, Ιούλιος, 2024



## ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Κατσίμπρας Δρόσος  
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών  
Πανεπιστήμιο Δυτικής Αττικής

Copyright © Κατσίμπρας Δρόσος, 2024  
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

«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,  
Κατσίμπρας Δρόσος





*Αφιερώνω την διπλωματική μου εργασία, στην οικογένεια μου.*



# Ευχαριστίες

Η ολοκλήρωση αυτής της διπλωματικής εργασίας δεν θα ήταν δυνατή χωρίς τη στήριξη και τη βοήθεια κάποιων ανθρώπων, τους οποίους θέλω να ευχαριστήσω από καρδιάς. Πρώτα απ' όλα, θέλω να εκφράσω την ευγνωμοσύνη μου προς τον καθηγητή μου, κ. Παναγιώτη Καρκαζή, για την ακαδημαϊκή του βοήθεια και την εκτίμησή του προς την ερευνητική μου προσπάθεια. Οι συμβουλές, και οι καθοδηγήσεις του ήταν πολύτιμες κατά την πορεία αυτής της εργασίας.

Επίσης, δεν μπορώ παρά να ευχαριστήσω την οικογένειά μου για τη στήριξη και την αμέριστη συμπαράστασή τους. Τα ευχαριστώ μου δεν επαρκούν για να εκφράσω το βάθος της ευγνωμοσύνης μου για την αφοσίωση και την ανυποχώρητη υποστήριξή τους κατά τη διάρκεια αυτής της σημαντικής πορείας της ζωής μου.

01/03/2024  
Κατσίμπρας Δρόσος



# Περίληψη

Η παρούσα διπλωματική εργασία εστιάζει σε καινοτόμες μεθοδολογίες για την διαχείριση του κύκλου ζωής των συστημάτων λογισμικού. Πιο συγκεκριμένα, εμβαθύνει στην έννοια του "code everything", διερευνά νέες προσεγγίσεις στην ανάπτυξη καθώς και την συντήρηση λογισμικού. Με την υιοθέτηση τεχνικών και εργαλειών αιχμής, όπως η υποδομή ως κώδικας (IaC) και τα εργαλεία τα οποία έχει (π.χ Terraform, Pulumi, Ansible), οι αγωγοί συνεχούς ολοκλήρωσης/συνεχούς ανάπτυξης (CI/CD) και τα προηγμένα συστήματα ελέγχου εκδόσεων (Git), η μελέτη έχει ως στόχο να φέρει την επανάσταση στις παροδοσιακές πρακτικές διαχείρισης του κύκλου ζωής λογισμικού. Συνολικά ο πρακτικός πειραματισμός και η θεωρητική ανάλυση, επιδιώκει να αποκαλύψει νέες στρατηγικές για την βελτιστοποίηση των διαδικασιών κατά την ανάπτυξη του λογισμικού, την ενίσχυση της αξιοπιστίας του συστήματος και την επιτάχυνση του χρόνου διάθεσης στην αγορά. Συνεπώς, η έρευνα στοχεύει να συμβάλει στην εξέλιξη των πρακτικών του software engineering και να ανοίξει το δρόμο για μελλοντικές καινοτομίες στον τομέα.

**Λέξεις Κλειδιά:** Υποδομή ως Κώδικας (IaC), Εργαλεία IaC: Terraform, Pulumi, Ansible, Καινοτόμες Προσεγγίσεις, Διαχείριση Κύκλου Ζωής Λογισμικού, Συνεχής Ενοποίηση (CI), Συνεχής Ανάπτυξη (CD), Συστήματα Ελέγχου Εκδόσεων, Ανάπτυξη Λογισμικού, Αυτοματισμός Ανάπτυξης, Αξιοπιστία Συστήματος.



# Abstract

This diploma thesis focuses on innovative methodologies for software systems lifecycle management. More specifically, it delves into the concept of "code everything", explores new approaches to software development as well as maintenance. By adopting cutting-edge techniques and tools, such as infrastructure as code (IaC), continuous integration/-continuous development (CI/CD) pipelines, and advanced version control systems (Git), the study aims to revolutionize traditional software lifecycle management practices. Overall practical experimentation and theoretical analysis, it seeks to uncover new strategies to optimize processes during software development, enhance system reliability and accelerate time to market. Therefore, the research aims to contribute to the evolution of software engineering practices and pave the way for future innovations in the field.

**Keywords:** Infrastructure as Code (IaC), IaC tools: Terraform, Pulumi, Ansible, Innovative Approaches, Software Lifecycle Management, Code Everything, Continuous Integration (CI), Continuous Deployment (CD), Version Control Systems, Software Development, Deployment Automation, System Reliability.



# Πίνακας περιεχομένων

|   |             |
|---|-------------|
| <b>Ευχαριστίες</b>  | <b>iii</b>  |
| <b>Περίληψη</b>   | <b>v</b>    |
| <b>Abstract</b>   | <b>vii</b>  |
| <b>Πίνακας περιεχομένων</b>                               | <b>xi</b>   |
| <b>Πίνακας σχημάτων</b>                                   | <b>xiv</b>  |
| <b>Πίνακας πινάκων</b>                                    | <b>xv</b>   |
| <b>Λίστα Συντομογραφιών</b>                               | <b>xvii</b> |
| <b>1 Εισαγωγή</b>   | <b>1</b>    |
| 1.1 Πλαίσιο, σκοπός και στόχοι . . . . .                  | 1           |
| 1.2 Μεθοδολογία . . . . .                                 | 2           |
| 1.3 Περιορισμοί . . . . .                                 | 2           |
| 1.3.1 Περιορισμοί πόρων . . . . .                         | 3           |
| 1.3.2 Περιορισμοί χρόνου . . . . .                        | 3           |
| 1.3.3 Τεχνογνωσία . . . . .                               | 3           |
| 1.3.4 Επεκτασιμότητα . . . . .                            | 3           |
| 1.4 Οργάνωση της εργασίας . . . . .                       | 4           |
| 1.4.1 Επισκόπηση Κεφαλαίων . . . . .                      | 4           |
| 1.4.2 Ροή Κεφαλαίων και Συνδέσεις . . . . .               | 4           |
| <b>2 Θεμελιώδες αρχές του Υπολογιστικού Νέφους</b>        | <b>7</b>    |
| 2.1 Εισαγωγή στο Cloud Computing . . . . .                | 7           |
| 2.1.1 Ιστορική Αναδρομή και Εξέλιξη . . . . .             | 7           |
| 2.1.2 Ορισμός του Cloud Computing . . . . .               | 8           |
| 2.1.3 Βασικά χαρακτηριστικά του cloud computing . . . . . | 8           |
| 2.1.4 Οφέλη του Cloud Computing . . . . .                 | 9           |
| 2.1.5 Προκλήσεις του Cloud Computing . . . . .            | 9           |
| 2.2 Τύποι Cloud Services . . . . .                        | 10          |
| 2.2.1 SaaS (Software as a Service) . . . . .              | 10          |
| 2.2.2 IaaS (Infrastructure as a Service) . . . . .        | 10          |
| 2.2.3 PaaS (Platform as a Service) . . . . .              | 11          |
| 2.3 Τύποι Ανάπτυξης Cloud . . . . .                       | 12          |
| 2.3.1 Δημόσιο Νέφος . . . . .                             | 12          |
| 2.3.2 Ιδιωτικό Νέφος . . . . .                            | 12          |
| 2.3.3 Υβριδικό Νέφος . . . . .                            | 12          |

|  |           |
|--|-----------|
| 2.3.4 Σύννεφο Κοινότητας . . . . .   | 13        |
| <b>3 Εξέλιξη της Διαχείριση Κύκλου ζωής Λογισμικού</b>                     | <b>15</b> |
| 3.1 Εισαγωγή . . . . .   | 15        |
| 3.2 Ιστορική εξέλιξη του SDLC . . . . .                                    | 16        |
| 3.2.1 Φάσεις του SDLC . . . . .  | 17        |
| 3.3 Εξέλιξη των μεθοδολογιών του SDLC . . . . .                            | 20        |
| 3.3.1 Παραδοσιακά μοντέλα του SDLC . . . . .                               | 20        |
| 3.3.2 Μοντέρνα μοντέλα του SDLC . . . . .                                  | 24        |
| 3.3.3 Σύγκριση παραδοσιακών - μοντέρνων Μοντέλων SDLC . . . . .            | 27        |
| 3.4 Μεταβατικές Προσεγγίσεις του SDLC . . . . .                            | 29        |
| 3.4.1 Παραδοσιακή χρήση πόρων στο SDLC . . . . .                           | 29        |
| 3.4.2 Σύγχρονη χρήση πόρων στο SDLC . . . . .                              | 31        |
| <b>4 Προσεγγίσεις Infrastructure as Code</b>                               | <b>33</b> |
| 4.1 Εισαγωγή στο Infrastructure as Code . . . . .                          | 33        |
| 4.1.1 Ορισμός του Infrastructure as Code . . . . .                         | 33        |
| 4.1.2 Βασικές προσεγγίσεις του IaC . . . . .                               | 34        |
| 4.1.3 Σύγκριση Mutable IaC και Immutable IaC σε σχέση με το SDLC . . . . . | 35        |
| 4.1.4 Βασικές προσεγγίσεις προγραμματισμού του IaC . . . . .               | 36        |
| 4.2 Βασικά Χαρακτηριστικά του IaC . . . . .                                | 37        |
| 4.3 Πλεονεκτήματα και Προκλήσεις του IaC . . . . .                         | 38        |
| 4.3.1 Πλεονεκτήματα του IaC . . . . .                                      | 38        |
| 4.3.2 Προκλήσεις στο πλαίσιο του IaC . . . . .                             | 42        |
| 4.3.3 Συμπεράσματα για το IaC . . . . .                                    | 43        |
| 4.4 Καλύτερες πρακτικές IaC . . . . .                                      | 44        |
| 4.5 Εργαλεία του IaC . . . . .   | 46        |
| 4.5.1 Εισαγωγή στα εργαλεία του IaC . . . . .                              | 46        |
| 4.5.2 Terraform . . . . .  | 47        |
| 4.5.3 Ansible . . . . .  | 57        |
| 4.5.4 Pulumi . . . . .   | 63        |
| 4.5.5 Chef . . . . .   | 67        |
| 4.5.6 Puppet . . . . .   | 70        |
| 4.6 Infrastructure as Code εργαλεία από Cloud Providers . . . . .          | 72        |
| 4.6.1 AWS CloudFormation . . . . .   | 72        |
| 4.6.2 Azure Resource Manager . . . . .                                     | 74        |
| 4.6.3 Google Cloud Deployment Manager . . . . .                            | 77        |
| 4.7 Σύγκριση εργαλείων IaC . . . . .                                       | 79        |
| 4.8 IaC: DevOps και CI/CD pipelines . . . . .                              | 81        |
| 4.8.1 DevOps . . . . .   | 81        |
| 4.8.2 CI/CD pipelines . . . . .  | 85        |
| <b>5 Μεθοδολογία – Υλοποίηση – Εφαρμογή</b>                                | <b>89</b> |
| 5.1 Μεθοδολογία . . . . .  | 89        |
| 5.1.1 Ερευνητική προσέγγιση . . . . .                                      | 89        |
| 5.1.2 Μέθοδοι . . . . .  | 90        |
| 5.1.3 Τεχνικές Ανάλυσης . . . . .  | 91        |
| 5.2 Περιγραφή Υλοποίησης . . . . .   | 91        |
| 5.2.1 Λεπτομερής Περιγραφή των Διαδικασιών . . . . .                       | 92        |
| 5.2.2 Δυσκολίες που αντιμετωπίστηκαν . . . . .                             | 93        |

---

|                                |   |            |
|--------------------------------|---|------------|
| 5.3                            | Αρχιτεκτονική Συστήματος . . . . .                          | 94         |
| 5.3.1                          | Infrastructure : Terraform Configuration files . . . . .    | 94         |
| 5.3.2                          | CI/CD pipeline με GitLab . . . . .                          | 95         |
| 5.4                            | Use case σενάριο . . . . .                                  | 103        |
| 5.4.1                          | Use case σενάριο: Automated deploy of web app . . . . .     | 103        |
| 5.4.2                          | CI/CD pipeline - workflow . . . . .                         | 104        |
| 5.5                            | Χρονοδιάγραμμα . . . . .                                    | 114        |
| 5.6                            | Προϋπολογισμός . . . . .                                    | 115        |
| <b>6</b>                       | <b>Αποτελέσματα – Επιτεύγματα</b>                           | <b>117</b> |
| 6.1                            | Αποτελέσματα κατά την εκτέλεση του CI/CD pipeline . . . . . | 117        |
| 6.2                            | Ανάπτυξη και Προσβασιμότητα Υποδομής . . . . .              | 118        |
| 6.3                            | Ανάπτυξη και Προσβασιμότητα εφαρμογής . . . . .             | 120        |
| 6.4                            | Infrastructure Monitoring . . . . .                         | 121        |
| 6.5                            | Terraform Σχεδιάγραμμα . . . . .                            | 122        |
| 6.6                            | GitLab CI/CD pipeline . . . . .                             | 123        |
| <b>7</b>                       | <b>Συμπεράσματα – Μελλοντικές επεκτάσεις</b>                | <b>125</b> |
| 7.1                            | Αποτελέσματα . . . . .                                      | 125        |
| 7.1.1                          | Αποδοτικότητα αγωγού CI/CD . . . . .                        | 125        |
| 7.1.2                          | Σταθερότητα Υποδομής . . . . .                              | 125        |
| 7.2                            | Μελλοντικές επεκτάσεις . . . . .                            | 126        |
| 7.2.1                          | Βελτιωμένη παρακολούθηση και ειδοποίηση . . . . .           | 126        |
| 7.2.2                          | Βελτιώσεις ασφάλειας . . . . .                              | 126        |
| 7.3                            | Βελτίωση επεκτασιμότητας . . . . .                          | 127        |
| 7.4                            | Συμπέρασμα . . . . .  | 127        |
| <b>A</b>                       | <b>Terraform</b>  | <b>129</b> |
| A.1                            | Terraform Configuration files . . . . .                     | 129        |
| <b>B</b>                       | <b>GitLab CI/CD pipeline</b>                                | <b>133</b> |
| B.1                            | Αρχείο .gitlab-ci.yml . . . . .                             | 133        |
| <b>Βιβλιογραφικές Αναφορές</b> |   | <b>137</b> |



# Πίνακας σχημάτων

|            |   |    |
|------------|---|----|
| Εικόνα 1.  | Ροή εργασίας διπλωματικής . . . . .                               | 2  |
| Εικόνα 2.  | Τύποι των Υπολογιστικών Υπηρεσιών . . . . .                       | 11 |
| Εικόνα 3.  | Τύποι Ανάπτυξης Cloud . . . . .                                   | 13 |
| Εικόνα 4.  | Φάσεις του Software Development LifeCycle . . . . .               | 19 |
| Εικόνα 5.  | Waterfall μοντέλο . . . . .                                       | 20 |
| Εικόνα 6.  | V-shaped μοντέλο . . . . .  | 21 |
| Εικόνα 7.  | Spiral μοντέλο . . . . .  | 22 |
| Εικόνα 8.  | Big Bang μοντέλο . . . . .  | 23 |
| Εικόνα 9.  | Prototype μοντέλο . . . . .                                       | 23 |
| Εικόνα 10. | Agile μοντέλο . . . . .   | 24 |
| Εικόνα 11. | RAD μοντέλο . . . . .   | 25 |
| Εικόνα 12. | Incremental μοντέλο . . . . .                                     | 26 |
| Εικόνα 13. | Iterative μοντέλο . . . . .                                       | 26 |
| Εικόνα 14. | Μοντέλα του Software Development LifeCycle . . . . .              | 28 |
| Εικόνα 15. | Παραδοσιακή χρήση resources . . . . .                             | 30 |
| Εικόνα 16. | Δομή του Infrastructure as Code . . . . .                         | 34 |
| Εικόνα 17. | Mutable-Immutable IaC . . . . .                                   | 36 |
| Εικόνα 18. | Προσεγγίσεις προγραμματισμού του Infrastructure as Code . . . . . | 37 |
| Εικόνα 19. | Πλεονεκτήματα του Infrastructure as Code . . . . .                | 41 |
| Εικόνα 20. | Εφαρμογή του Infrastructure as Code . . . . .                     | 45 |
| Εικόνα 21. | Εργαλεία IaC . . . . .  | 46 |
| Εικόνα 22. | Αρχιτεκτονική του Terraform . . . . .                             | 51 |
| Εικόνα 23. | Ροή εργασίας του Terraform . . . . .                              | 53 |
| Εικόνα 24. | Terraform CDK . . . . .   | 56 |
| Εικόνα 25. | Αρχιτεκτονική Ansible . . . . .                                   | 61 |
| Εικόνα 26. | Αρχιτεκτονική Pulumi . . . . .                                    | 66 |
| Εικόνα 27. | Αρχιτεκτονική Chef . . . . .                                      | 69 |
| Εικόνα 28. | Αρχιτεκτονική Puppet . . . . .                                    | 71 |
| Εικόνα 29. | Αρχιτεκτονική AWS CloudFormation . . . . .                        | 73 |
| Εικόνα 30. | Αρχιτεκτονική Azure Resource Manager . . . . .                    | 76 |
| Εικόνα 31. | Αρχιτεκτονική Google Cloud Deployment Manager . . . . .           | 78 |
| Εικόνα 32. | Κύκλος ζωής του DevOps . . . . .                                  | 83 |
| Εικόνα 33. | Τα 7 Cs του DevOps . . . . .                                      | 85 |
| Εικόνα 34. | Διάγραμμα .gitlab-ci.yml : Template, variables, stages . . . . .  | 96 |
| Εικόνα 35. | Διάγραμμα .gitlab-ci.yml : Validate stage . . . . .               | 97 |
| Εικόνα 36. | Διάγραμμα .gitlab-ci.yml : test stage . . . . .                   | 98 |

|            |  |     |
|------------|--|-----|
| Εικόνα 37. | Διάγραμμα .gitlab-ci.yml : build stage . . . . .         | 100 |
| Εικόνα 38. | Διάγραμμα .gitlab-ci.yml : deploy-linode stage . . . . . | 101 |
| Εικόνα 39. | Διάγραμμα .gitlab-ci.yml : deploy-image stage . . . . .  | 102 |
| Εικόνα 40. | Validate stage - report . . . . .                        | 104 |
| Εικόνα 41. | Test stage - report [1] . . . . .                        | 105 |
| Εικόνα 42. | Test stage - report [2] . . . . .                        | 105 |
| Εικόνα 43. | Test stage - report [3] . . . . .                        | 106 |
| Εικόνα 44. | Test stage - report [4] . . . . .                        | 107 |
| Εικόνα 45. | Build stage: Linode VM - report [1] . . . . .            | 107 |
| Εικόνα 46. | Build stage: Linode VM - report [2] . . . . .            | 108 |
| Εικόνα 47. | Build stage: Linode VM - report [3] . . . . .            | 108 |
| Εικόνα 48. | Build stage: Docker image - report [1] . . . . .         | 109 |
| Εικόνα 49. | Build stage: Docker image - report [2] . . . . .         | 109 |
| Εικόνα 50. | Build stage: Docker image - report [3] . . . . .         | 110 |
| Εικόνα 51. | Build stage: Docker image - report [4] . . . . .         | 110 |
| Εικόνα 52. | Deploy stage: Linode VM - report [1] . . . . .           | 111 |
| Εικόνα 53. | Deploy stage: Linode VM - report [2] . . . . .           | 112 |
| Εικόνα 54. | Deploy stage: Linode VM - report [3] . . . . .           | 112 |
| Εικόνα 55. | Deploy stage: Docker image - report [1] . . . . .        | 113 |
| Εικόνα 56. | Deploy stage: Docker image - report [2] . . . . .        | 113 |
| Εικόνα 57. | CI/CD pipeline - Group by Dependencies . . . . .         | 117 |
| Εικόνα 58. | CI/CD pipeline - Group by Stage . . . . .                | 118 |
| Εικόνα 59. | Linode VM . . . . .                                      | 118 |
| Εικόνα 60. | Linode VM . . . . .                                      | 119 |
| Εικόνα 61. | Docker version . . . . .                                 | 119 |
| Εικόνα 62. | Deployed Application . . . . .                           | 120 |
| Εικόνα 63. | Docker ps . . . . .                                      | 120 |
| Εικόνα 64. | Docker ps . . . . .                                      | 121 |
| Εικόνα 65. | Infrastructure Monitoring . . . . .                      | 121 |
| Εικόνα 66. | Terraform diagram . . . . .                              | 122 |
| Εικόνα 67. | Terraform : Linode VM . . . . .                          | 122 |
| Εικόνα 68. | GitLab CI/CD pipeline . . . . .                          | 123 |
| Εικόνα 69. | Cloud Provider : Linode . . . . .                        | 129 |
| Εικόνα 70. | Linode VM . . . . .                                      | 130 |
| Εικόνα 71. | Linode VM : instance ip . . . . .                        | 130 |
| Εικόνα 72. | Αρχικοποίηση Terraform variables . . . . .               | 131 |
| Εικόνα 73. | Τιμές των Terraform variables . . . . .                  | 132 |
| Εικόνα 74. | .gitlab-ci.yml : Template, variables, stages . . . . .   | 133 |
| Εικόνα 75. | .gitlab-ci.yml : Validate, test-app . . . . .            | 134 |
| Εικόνα 76. | .gitlab-ci.yml : build-linode, build-image . . . . .     | 134 |
| Εικόνα 77. | .gitlab-ci.yml : deploy-linode, deploy-image . . . . .   | 135 |

# Πίνακας πινάκων

|            |  |     |
|------------|--|-----|
| Πίνακας 1. | Παράδειγμα terraform.tf αρχείου          | 54  |
| Πίνακας 2. | Παράδειγμα Playbook σε YAML αρχείο       | 62  |
| Πίνακας 3. | Σύγκριση Infrastructure as Code εργαλεία | 79  |
| Πίνακας 4. | Χρονοδιάγραμμα διπλωματικής εργασίας     | 114 |
| Πίνακας 5. | Προϋπολογισμός διπλωματικής εργασίας     | 115 |



# Λίστα συντομογραφιών

**AWS** Amazon Web

**GCP** Google Cloud Platform

**IaaS** Infrastructure as Service Inc.

**PaaS** Platform as Service

**SaaS** Software as Service

**IaC** Infrastructure as Code

**CI/CD** Continuous Integration/Continuous Delivery/Deployment

**SDLC** Software Development LifeCycle

**ARM** Azure Resource Manager

**GCDM** Google Cloud Deployment Manager

**IAM** Identity and Access Management

**HCL** HashiCorp Language

**JSON** JavaScript Object Notation

**YAML** Yet another markup language

**SDK** Software Development Kit

**UI** User Interface

**MacOS** Macintosh Operating System

**SSH** Secure Shell

**HTTPS** Hypertext transfer protocol secure

**REST** REpresentational State Transfer

**RAM** Random Access Memory

**CPU** Central Processing Unit

**CRM** Customer Relationship Management



# Κεφάλαιο 1

## Εισαγωγή

Στο πλαίσιο αυτής της εργασίας, γίνεται μια αναφορά στις θεμελιώδης αρχές του cloud computing, όπου θα είναι και η βάση ώστε να υπάρξει περαιτέρω ανάλυση στα επόμενα κεφάλαια για τον Κύκλο ζωής των συστημάτων λογισμικού, αλλά και των καινοτόμων τεχνολογικών προσεγγίσεων. Στόχος της διπλωματικής είναι να αναλυθεί με σαφήνεια το Software Development Lifecycle [SDLC] και έπειτα η έννοια της Υποδομής ως Κώδικα [IaC] με τα εργαλεία που υπάρχουν και εξελίσσονται συνεχώς, - όπου θα γίνει ανάλυση αυτών - καθώς παράλληλα και πως φτάσαμε στην υιοθέτησης των εννοιών του DevOps και των CI/CD pipelines.

### 1.1 Πλαίσιο, σκοπός και στόχοι

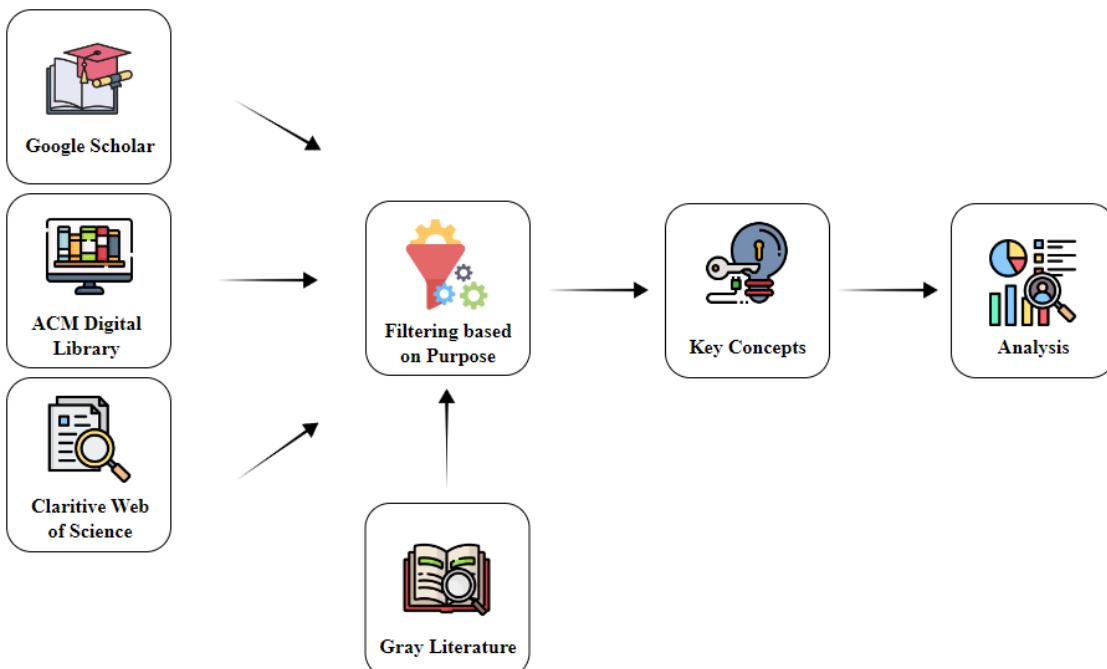
Η διπλωματική εργασία αναπτύσσει τη διαδικασία που χρειάζεται ώστε να δημιουργηθεί κατάλληλο περιβάλλον σε έναν cloud provider [AWS, Google Cloud, Microsoft Azure, Linode, DigitalOcean] με χρήση καινοτόμων εργαλείων του Infrastructure as Code [IaC], όπως για παράδειγμα το Terraform, το Ansible ή το Pulumi.

Συνεπώς σκοπός είναι η δημιουργία αρχικά μιας αποδοτικής και αξιόπιστης υποδομής στο cloud για εφαρμογές και υπηρεσίες και η υιοθέτηση των εννοιών του DevOps και CI/CD pipelines, για το πώς μπορεί πλέον ένας οργανισμός να παραδίδει ένα λογισμικό το συντομότερο δυνατόν με όλα τα πλεονεκτήματα που έχει η Υποδομή ως Κώδικας.

Παράλληλα, είναι σημαντικό να δοθεί έμφαση πως μέσα από version control συστήματα [GitLab] και όλων των παραπάνω, μπορούν οι προγραμματιστές να συνεργάζονται με αμεσότητα και αποτελεσματικότητα ώστε να αναπτύσσεται ένα συγκεκριμένο έργο που είναι προς υλοποίηση. Η φιλοσοφία της διπλωματικής, η οποία θα αναλυθεί διεξοδικά μέσα από την βιβλιογραφική έρευνα, καθώς και το πρακτικό σκέλος είναι η έννοια του **CODE EVERYTHING**.

## 1.2 Μεθοδολογία

Για την επίτευξη των στόχων της διπλωματικής εργασίας, πραγματοποιήθηκε εκτεταμένη βιβλιογραφική έρευνα σχετικά με τις καινοτόμες προσεγγίσεις του SDLC. Η ακριβής κατανόηση όλων των εννοιών είναι απαραίτητο για την σωστή αντίληψη που πρέπει να έχουμε της αποτελεσματικότητας των εργαλείων και του κύκλου ζωής του λογισμικού. Η αντληση των πληροφοριών και η μέθοδος που ακολουθήσαμε στην μελέτη κατά την επιστημονική έρευνα φαίνεται στο παρακάτω σχήμα:



Εικόνα 1. Ροή εργασίας διπλωματικής

## 1.3 Περιορισμοί

Σε αυτή την ενότητα είναι σημαντικό να αναφερθούν και οι περιορισμοί που αντιμετωπίστηκαν κατά τη διάρκεια της εργασίας, κυρίως σε σχέση με τους περιορισμούς των υπηρεσιών και των πόρων που προσφέρονται από τον πάροχο υπολογιστικών υπηρεσιών, Καθώς και περιορισμούς που δημιοργήθηκαν σε σχέση με τον χρόνο και την ισοροπία της εκπόνησης μιας άρτιας διπλωματικής.

Πιο συγκεκριμένα για να μπορέσεις να έχεις resources από cloud providers πρέπει να χρεωθείς κάποιο χρηματικό ποσό. Υπάρχει φυσικά και η δυνατότητα να μπορέσεις με λιγοστούς πόρους και υπολογιστική ισχύ, να κάνεις κάποια χρήσιμα τεστ. Όμως αυτοί οι περιορισμοί επηρέασαν ως ένα βαθμό το εύρος και τα αποτέλεσμα της έρευνας μας.

Συνεπώς, είναι απαραίτητο να μπορέσουμε να αναγνωρίσουμε αυτούς του περιορισμούς, καθώς παρέχουν χρήσιμες γνώσεις για τις προκλήσεις που αντιμετωπίσαμε κατά την διάρκεια της διπλωματικής εργασίας.

### 1.3.1 Περιορισμοί πόρων

Στην εισαγωγή της ενότητας αναφέραμε πως υπήρχαν περιορισμοί ως προς τους πόρους που ήταν διαθέσιμοι, αλλά και το κόστος που είχαν ανάλογα την επιλογή στις πλατφόρμες των cloud providers. Στο πρακτικό μέρος, συνατήσαμε μια πρόκληση στο να δημιουργήσουμε μεγάλης κλίμακας υποδομής. Αυτός ο περιορισμός, ήταν μια πρόκληση καθώς προσπαθήσαμε να βελτιστοποιήσουμε τη χρήση των πόρων μας και να εφαρμόσουμε αποτελεσματικά κάποια σενάρια με την υποδομή και τους πόρους που είχαμε με το λιγότερο δυνατόν κόστος.

### 1.3.2 Περιορισμοί χρόνου

Σημαντικό να αναφέρουμε πως υπήρξε κατά την εκπόνηση της διπλωματικής εργασίας κάποιοι περιορισμοί ως προς τον χρόνο. Σημαντικό ήταν να δημιουργήσουμε την κατάλληλη ισποροπία μεταξύ της ανάγκης για ανάλυση των πληροφοριών που συλλέγαμε από επιστημονικά άρθρα, τον πειραματισμό με τις καινοτόμες τεχνολογικές προσεγγίσεις του SDLC καθώς και την βαθιά κατανόηση των εννοιών αυτών, συνδιάζοντας το με την ανάγκη τήρησης των προθεσμιών που είχαμε στην διάθεση μας. Συνεπώς, αυτός ο περιορισμός ίσως δημιούργησε κατά την ανάλυση και την υλοποίηση του σεναρίου, την επηρέαση της πληρότητας των ευρημάτων μας, αλλά και την αποτύπωση τους.

### 1.3.3 Τεχνογνωσία

Κατά την διάρκεια της διπλωματικής εργασίας, παρουσιάστηκαν διάφορα ζητήματα ως προς την ανάγκη και την σημασία της γνώσης. Χρειάστηκε να δοθεί εστίαση στην βάση των γνώσεων που ήδη υπήρχαν και μέσα από αυτή να μπορέσουμε να αναλύσουμε, να καταβάλουμε προσπάθεια στην κατανόηση των εννοιών, ώστε να δημιουργηθεί μια διπλωματική άρτια και σημαντική, αλλά και χρήσιμη για το μέλλον. Συμπερασματικά λοιπόν, μπορούμε να πούμε πως είναι επιτακτική ανάγκη της συνεχούς εξέλιξης στο τομέα της τεχνογνωσίας και γενικότερα της γνώσης, ώστε μελλοντικά να υπάρχει η δυνατότητα να είμαστε ενημερωμένοι με τις τελευταίες εξελίξεις σε αυτούς τους τομείς.

### 1.3.4 Επεκτασιμότητα

Σε αυτή την υποενότητα είναι σημαντικό να αναφερθεί η επεκτασιμότητα που είχε από την αρχή έως το τέλος η ανάλυση και το σενάριο που ήταν προς υλοποίηση. Κατά την διάρκεια της ανάπτυξης των μεθοδολογιών και των σύγχρονων προσεγγίσεων για το SDLC, καθώς αυξανόταν η πολυπλοκότητα, αντιμετωπίσαμε κάποια ζητήματα ως προς την ενσωμάτωσης όλων των στοιχείων που δημιουργούσαμε στο πρακτικό κομμάτι της διπλωματικής. Γι' αυτό το λόγο φτάσαμε στο συμπέρασμα πως υπάρχει επιτακτική ανάγκη για υιοθέτηση καινοτόμων προσεγγίσεων που μπορούν να προσαρμοστούν στις συνεχώς μεταβαλλόμενες απαιτήσεις στον τεχνολογικό κόσμο.

Συμπερασματικά λοιπόν, έχοντας αναφέρει τους περιορισμούς που αντιμετωπίσαμε κατά την διάρκεια της διπλωματικής εργασίας, μπορούμε να πούμε πως οι περιορισμοί λειτούργησαν ως σημαντικά "μαθήματα" για την σημασία του προσεκτικού σχεδιασμού, της διαχείρισης των πόρων με σωστό τρόπο και το πιο σημαντικό, της συνεχούς μάθησης.

## 1.4 Οργάνωση της εργασίας

Στην ενότητα αυτή είναι σημαντικό τόσο για το ακαδημαϊκό ύφος της διπλωματικής, τόσο και για την σωστή ανάγνωση, όσο και για τον προσανατολισμό ενός αναγνώση, να μπορέσουμε να αποτυπώσουμε με σαφήνεια για το πώς επιλέξαμε να οργανώσουμε την διπλωματική εργασία. Έτσι λοιπόν, η διπλωματική εργασία δομείται σε διάφορα κεφάλαια, κάθε ένα από τα οποία εστιάζει σε διαφορετικές πτυχές της διαδικασίας του σεναρίου. Η οργάνωση ακολουθεί ένα λογικό μοντέλο, παρουσιάζοντας αρχικά τη θεωρητική βάση και καταλήγοντας στην πρακτική εφαρμογή, την αξιολόγηση, των συμπερασμάτων αλλά και των μελλοντικών προεκτάσεων. Παρακάτω, παρέχεται μια σύντομη επισκόπιση της δομής και του περιεχόμενου της διπλωματικής εργασίας :

### 1.4.1 Επισκόπηση Κεφαλαίων

Σε αυτή την υπονεότητα είναι χρήσιμο να δώσουμε επιγραμματικά και συνοπτικά τη βασική εστίαση και τα θέματα που καλύπτονται σε κάθε κεφάλαιο:

- **Κεφάλαιο 1** Εισαγωγή
- **Κεφάλαιο 2** Θεμελιώδες αρχές του Υπολογιστικού Νέφους
- **Κεφάλαιο 3** Εξέλιξη της Διαχείρισης Κύκλου Ζωής Λογισμικού
- **Κεφάλαιο 4** Προσεγγίσεις Infrastructure as Code
- **Κεφάλαιο 5** Μεθοδολογία - Εφαρμογή σεναρίου
- **Κεφάλαιο 6** Αποτελέσματα - Επιτεύγματα
- **Κεφάλαιο 7** Συμπεράσματα - Μελλοντικές επεκτάσεις

### 1.4.2 Ροή Κεφαλαίων και Συνδέσεις

Έχοντας δώσει επιγραμματικά τα κεφάλαια αυτής της διπλωματικής εργασίας, είναι σημαντικό να αναφέρουμε πως έχουν σχεδιαστεί με τέτοιο τρόπο ώστε να βασίζονται το ένα στο άλλο, δημιουργώντας μια συνεκτική αφήγηση που διερευνά τις καινοτόμες προσεγγίσεις στο Software Development LifeCycle [SDLC]. Συνοπτικά για κάθε κεφάλαιο μπορούμε να πούμε τα εξής:

- **Κεφάλαιο 1 : Εισαγωγή**
- **Κεφάλαιο 2 :** Σε αυτό το κεφάλαιο έχουμε δώσει έμφαση στην έννοια του cloud computing που σκοπό έχει να θέσει τα θεμέλια συζητώντας την τρέχουσα κατάσταση του cloud computing και τη συνάφεια του με το SDLC. Πιο συγκεκριμένα, το κεφάλαιο αυτό, παρέχει τις απαραίτητες βασικές γνώσεις για την πλήρη κατανόηση της εξέλιξης της διαχείρισης του κύκλου ζωής του λογισμικού μέσα από την υιοθέτηση του cloud computing.
- **Κεφάλαιο 3 :** Συνεχίζοντας στο Κεφάλαιο 3, σκοπός είναι να εμβαθύνουμε στην εξέλιξη της διαχείρισης του Κύκλου ζωής των συστημάτων λογισμικού, ζεκινώντας από την ιστορία και την ανάπτυξη του SDLC, επισημαίνοντας βασικές προόδους. Το κεφάλαιο αυτό είναι σημαντικό, καθώς χρησιμεύει να κατανοήσουμε τα παραδοσιακά μοντέλα του SDLC και χρησιμεύει ως γέφυρα μεταξύ των θεωρητικών θεμάτων του cloud computing και των καινοτόμων προσεγγίσεων με την υιοθέτηση του Infrastructure as Code.

- **Κεφάλαιο 4 :** Με αυτό το Κεφάλαιο, έχοντας αναλύσει την έννοια του SDLC και κατανοήσει την χρησιμότητα που έχει στις μεταβαλλόμενες τεχνολογικές εξελίξεις το cloud computing, αναλύουμε διεξοδικά την έννοια της Υποδομής ως Κώδικα, τα εργαλεία που υπάρχουν μέχρι τώρα, αλλά και την πρακτική υιοθέτηση τους στον πραγματικό κόσμο.
- **Κεφάλαιο 5 :** Μέσα στο Κεφάλαιο 5, δημιουργούμε πλέον ένα σαφές σενάριο και μια πρακτική εφαρμογή των όσων έχουν αναλυθεί στα προηγούμενα κεφάλαια, μέσα από τις καινοτόμες προσεγγίσεις για το SDLC. Συνεπώς, στόχος του κεφαλαίου είναι να παρέχει μια λεπτομερή μελέτη περίπτωσης, παρουσιάζοντας αποτελεσματικότητα των μεθοδολογιών, αλλά και των προκλήσεων που αντιμετωπίστηκαν.
- **Κεφάλαιο 6 :** Σε αυτό το Κεφάλαιο, δημιουργούμε μια δομημένη καταγραφή των αποτελεσμάτων που αντλήσαμε από το σενάριο καθώς και τα επιτεύγματα που είχαμε κατά την ανάπτυξη της μελέτης περίπτωσης.
- **Κεφάλαιο 7 :** Τελευταίο Κεφάλαιο, και έχοντας καταγράψει όλα τα αποτελέσματα του σεναρίου, θέτουμε τις βάσεις ώστε να εκπορεύσουμε σημαντικά συμπεράσματα της διπλωματικής εργασίας, αλλά και μελλοντικές επεκτάσεις που μπορεί να έχει το σενάριο.

Τέλος, σε όλη την διάρκεια της διπλωματικής εργασίας επισημαίνονται με σαφήνεια οι συνδέσεις μεταξύ της θεωρητικής και της πρακτικής πτυχής. Είναι σημαντικό, μέσα από τα αρχικά βήματα της θεωρίας για το SDLC και του Cloud Computing να εφαρμοστούν σε σενάρια πρακτικού κόσμου, αποδεικνύοντας με σαφήνεια τον πιθανό αντίκτυπο των καινοτόμων προσεγγίσεων SDLC με την υιοθέτηση του Infrastructure as Code. Αυτό, διασφαλίζει ότι ο αναγνώστης μπορεί εύκολα να προσανατολιστεί και να κατανοήσει πως τα διάφορα Κεφάλαια που έχουν αναπτυχθεί και συμβάλλουν στην συνολική κατανόηση της ουσίας της διπλωματικής εργασίας.



## Κεφάλαιο 2

# Θεμελιώδες αρχές του Υπολογιστικού Νέφους

Σε αυτό το κεφάλαιο θα παρέχουμε μία σφαιρική επισκόπηση για το Cloud Computing, μιας θεμελιώδους τεχνολογίας που υποστηρίζει καινοτόμες προσεγγίσεις στη διαχείριση του κύκλου ζωής του λογισμικού. Ξεκινώντας με μία ιστορική ανασκόπηση του cloud computing, καταγράφοντας την εξέλιξη του μέχρι και σήμερα. Στην συνέχεια δίνεται ένας σαφής ορισμός του cloud computing, συνοδευόμενη από τα βασικά χαρακτηριστικά. Παράλληλα συζητούνται τα οφέλη του, καθώς επίσης και τα μεινοκτήματα που μπορεί να έχει, συμπεριλαμβανομένων και των ερωτημάτων που δημιουργούνται για την ασφάλεια και την εξάρτηση από την σύνδεση στο διαδίκτυο. Επίσης, εξετάζουμε συνοπτικά τους διάφορους τύπους υπηρεσιών cloud [SaaS, IaaS, PaaS] και εξερευνά τα διαφορετικά μοντέλα ανάπτυξης στο cloud, δηλαδή τα δημόσια, ιδιωτικά, υβριδικά, και κοινοτικά clouds. Παρέχοντας μια εμπεριστατωμένη κατανόηση των εννοιών του cloud computing, αυτό το κεφάλαιο θέτει τις βάσεις για την επόμενη συζήτηση σχετικά με τις καινοτόμες προσεγγίσεις για την διαχείριση του κύκλου ζωής του λογισμικού, τονίζοντας τον κρίσιμο ρόλο του cloud computing στην επιτάχυνση, στην αποτελεσματικότητα και στην κλιμάκωση των πρακτικών ανάπτυξης λογισμικού.

### 2.1 Εισαγωγή στο Cloud Computing

Η έννοια του "Cloud Computing" τα τελευταία χρόνια έχει επεκταθεί σημαντικά, και αποτελεί πλέον μία θεμελιώδη τεχνολογική εξέλιξη στην Πληροφορική. Αυτή η εξέλιξη έχει διαμορφώσει τον τρόπο όπου οι επιχειρήσεις μπορούν να αποθηκεύουν, να διαχειρίζονται και να αναπτύσσουν τις υπηρεσίες ή τις εφαρμογές τους. Το Cloud Computing προσφέρει ένα αποτελεσματικό και ευέλικτο περιβάλλον, επιτρέποντας στις ομάδες που εργάζονται μέσα σε επιχειρήσεις πληροφορικής να ανταποκρίνονται στις σύγχρονες και μεταβαλλόμενες συνθήκες της σημερινής εποχής.

#### 2.1.1 Ιστορική Αναδρομή και Εξέλιξη

Σε αυτήν την υποενότητα αξίζει να αναφερθούμε στην ιστορία της έννοιας του cloud computing. Εισήχθη για πρώτη φορά στην δεκαετία του 1960, από τον John McCarthy, όπου είχε υποστηρίξει πως ο υπολογισμός μπορεί κάποια μέρα να οργανωθεί ως μία κοινή χρησιμότητα. Στην διάρκεια του 1966, ο Douglas Parkhill εξερεύνησε τα χαρακτηριστικά του cloud computing. Το 1999 η εταιρεία Salesforce πρωτοπορεί στον τομέα του SaaS καθώς παρουσίασε το πρώτο CRM στο "cloud". Όμως η έννοια του cloud computing ωρί-

μασε με την εισαγωγή του Amazon Web Services (AWS) το 2006, παρέχοντας υπηρεσίες μέσω διαδικτύου. Παράλληλα και άλλοι τεχνολογικοί κολλοσοί όπως η Microsoft και η Google ακολούθησαν το παράδειγμα, εισάγοντας τις δικές τους υπηρεσίες cloud. Σήμερα, το cloud computing αποτελεί βασικό κομμάτι του infrastructure των επιχειρήσεων, παρέχοντας ευελιξία, αποτελεσματικότητα και κλιμακούμενες υπολογιστικές λύσεις.

### 2.1.2 Ορισμός του Cloud Computing

Το Cloud Computing θεωρείται πως είναι ένα σύνολο τεχνολογιών και υπηρεσιών όπου με την παροχή πόρων και υποδομών, δίνεται η δυνατότητα για την αποθήκευση και την διαχείρηση δεδομένων, εφαρφμογών, διακομιστών και δικτύων μέσα από το διαδίκτυο.

Σημαντικό να αναφερθεί πως οι χρήστες μπορούν να αξιοποιούν αυτούς τους πόρους όταν είναι αναγκαίο, πληρώνοντας μόνο για την πραγματική χρήση τους, χωρίς να χρειάζεται να επενδύουν σε υποδομές. Καταλαβαίνουμε λοιπόν, πως οι επιχειρίσεις έχουν μία ευλεξία στο να διαχειρίζονται τις ανάγκες τους καθώς τους επιτρέπεται να κλιμακώνουν τους πόρους που τους δίνονται από έναν cloud provider, χωρίς να χρειάζεται να συντηρούν δικό τους φυσικό infrastructure.

Συνεπώς το cloud computing μπορεί να προσφέρει μία οικονομικά αποδοτική και ευέλικτη λύση, επιτρέποντας στις επιχειρήσεις να εστιάζουν στην ανάπτυξη και την καινοτομία χωρίς την διαχείριση των υποδομών. Σε αυτή την υποενότητα θα εξετάσουμε παράλληλα τα οφέλη του cloud, τις προκλήσεις που μπορεί να αντιμετωπίσει μία επιχείρηση κατά την μετάβαση της σε μία τέτοια τεχνολογία και τα διάφορα μοντέλα υπηρεσιών που προσφέρονται όπως το **Infrastructure as a Service (IaC)**, το **Platform as a Service (PaaS)** και το **Software as a Service (SaaS)** [1].

### 2.1.3 Βασικά χαρακτηριστικά του cloud computing

Τα βασικά χαρακτηριστικά του cloud computing, επιγραμματικά, μπορούμε να πούμε πως είναι:

- **elasticity [ελαστικότητα]** : επιτρέπει στους χρήστες να κλιμακώνουν πόρους σύμφωνα με τις ανάγκες τους, επιτρέποντας τους να αντιδρούν σε μεταβαλλόμενες απαιτήσεις χωρίς την διακοπή των υπηρεσιών.
- **resilience [ανθεκτικότητα]** : αφορά την ικανότητα του συστήματος να διατηρεί τη λειτουργικότητά ακόμα και σε περιπτώσεις αποτυχίας εξαρτώμενες από το hardware ή software.
- **automation [αυτοματοποίηση]** : επιτρέπει την αυτόματη διαχείριση και εκτέλεση εργασιών, μειώνοντας τον ανθρώπινο παράγοντα και τον κίνδυνο ανθρώπινων λαθών.
- **self-service [αυτο-εξυπηρέτηση]** : επιτρέπει στους χρήστες να αναπτύσσουν, να προσαρμόζουν και να διαχειρίζονται υποδομές και εφαρμογές. Αυτά τα χαρακτηριστικά καθιστούν το cloud computing ένα ισχυρό εργαλείο για την υποστήριξη των αναγκών των σύγχρονων επιχειρήσεων.

### 2.1.4 Οφέλη του Cloud Computing

Στην σημερινή εποχή, έχοντας αναπτυχθεί αρκετά το cloud computing, πολλές εταιρίες σε διάφορους κλάδους επιλέγουν υπηρεσίες cloud καθώς προσφέρει πολλά πλεονεκτήματα. Πιο συγκεκριμένα μπορεί να διαθέτει:

- **scalability [επεκτασιμότητα]** : Σύμφωνα με αυτό το πλεονέκτημα, μία ανάπτυξη εφαρμογής καθώς και η εγκατάσταση της μπορεί να χρειάζεται επέκταση του υλικού [hardware] που υπάρχει σε ένα φυσικό infrastructure. Όταν για παράδειγμα οι απαιτήσεις μία εφαρμογής αλλάζουν και οι συνθήκες είναι διαφορετικές, τότε μέσω των υπηρεσιών cloud μπορούμε να αυξήσουμε ή να μειώσουμε τους πόρους για να μπορέσει να γίνει πιο σωστά η διαχείριση ενός heavy traffic ή μία ενδεχόμενη ανάπτυξη της εφαρμογής μας. Συνεπώς οι επιχειρήσεις μπορούν να χρησιμοποιούν τους υπολογιστικούς πόρους [computing resources] ανάλογα με τις συνθήκες [1].
- **cost-effectiveness [σχέση κόστους-αποτελεσματικότητας]** : Σημαντικό πλεονέκτημα που αξίζει να αναφερθεί είναι η σχέση κόστους- αποτελεσματικότητας . Στην ουσία, όλα τα έξοδα του hardware, όπως για παράδειγμα servers όπου χρειάζεται η εγκατάσταση σε ένα φυσικό δωμάτιο (server room), η διαχείρηση και η διατήρηση καθώς και άλλα components, αφαιρούνται από τις επιχειρήσεις καθώς τα παρέχει ένας cloud provider. Συνεπώς οι επιχειρήσεις, χρησιμοποιώντας cloud λύσεις εξοικονομούν χρήματα και γίνεται πληρωμή μόνο για τους πόρους που έχουν δεσμεύσει [1].
- **immediate availability [άμεση διαθεσιμότητα]** : Επιπροσθέτως, με το cloud computing έχουμε την δυνατότητα της άμεσης διαθεσιμότητας καθώς με συγκεκριμένο κόστος, χρησιμοποείται άμεσα μία υπηρεσία cloud [1].
- **security [ασφάλεια]** : Τέλος σημαντική είναι και η ασφάλεια καθώς η υποδομή του cloud βρίσκεται σε προηγμένα ασφαλή κέντρα δεδομένων, έχεις την δυνατότητα να υποστηρίζονται back-up τεχνικές, και να παρέχεται η προστασία του δικτύου μέσα από τείχη προστασίας [networking firewalls], σημαντικά tools για τον εντοπισμό cyber-crime στον κυβερνοχώρο, και κρυπτογράφησης [encryption] [1].

### 2.1.5 Προκλήσεις του Cloud Computing

Στην υποενότητα αυτή είναι σημαντικό να αναφερθούν τα μειονεκτήματα που έχει και χρειάζεται να γνωρίζει ένας οργανισμός όταν τείνει να χρησιμοποιήσει τέτοιες μεθόδους για την υποδομή του αλλά και για την ανάπτυξη συστημάτων λογισμικού σε πλατφόρμες cloud:

- **Απόρρητο και Ασφάλεια** : Η αποθήκευση δεδομένων σε έναν υπηρεσία υπολογιστικού νέφους μπορεί να εκθέσει τους χρήστες σε πιθανές παραβιάσεις απορρήτου και παραβιάσεις ασφαλείας. Υπάρχει κίνδυνος μη εξουσιοδοτημένης πρόσβασης σε εναίσθητα δεδομένα από τον πάροχο νέφους ή χάκερς, οδηγώντας σε προβλήματα όπως εταιρικός ξεπουλημένος, προφίλ χρηστών ή απόσπαση δεδομένων.
- **Δυσλειτουργία Υπηρεσίας** : Οποιαδήποτε δυσλειτουργία ή χρόνος ανενεργότητας της υπηρεσίας νέφους μπορεί να επηρεάσει τον αριθμό μεγάλων χρηστών ταυτόχρονα, περιορίζοντας την πρόσβαση στα δεδομένα και στις υπηρεσίες. Αυτή η εξάρτηση από εξωτερικές υπηρεσίες μπορεί να συνεπάγεται κινδύνους όσον αφορά τη συνέχεια και τη διαθεσιμότητα της υπηρεσίας.

- **Κόστη Μεταφοράς Δεδομένων :** Η μεταφορά δεδομένων στο νέφος μπορεί να συνεπάγεται σημαντικά κόστη, ιδιαίτερα για επιχειρήσεις μεγάλης κλίμακας με εκτεταμένες απαιτήσεις αποθήκευσης δεδομένων. Τα έξοδα που συνδέονται με τη μεταφορά δεδομένων στο νέφος πρέπει να εξεταστούν και να διαχειριστούν προσεκτικά [2].
- **Νομικές και Πολιτικές Προκλήσεις :** Η υπολογιστική νέφους συνεπάγεται την εμπιστοσύνη της διαχείρισης δεδομένων και επεξεργασίας σε εξωτερικούς παρόχους υπηρεσιών, η οποία ενδέχεται να εγείρει νομικά και πολιτικά ζητήματα, ιδίως όταν ο παροχέας νέφους λειτουργεί σε διαφορετική δικαιοδοσία από τον χρήστη ή την επιχείρηση. Αυτό μπορεί να δυσκολέψει τη νομική αντιμετώπιση σε περίπτωση παραβίασης δεδομένων ή παραβίασης απορρήτου [2].

Αυτοί οι παράγοντες υπογραμμίζουν μερικά από τα κύρια μειονεκτήματα και προκλήσεις που σχετίζονται με την υπολογιστική νέφους, τονίζοντας τη σημασία της αντιμετώπισης των ανησυχιών σχετικά με το απόρρητο, την ασφάλεια και τη λειτουργικότητα σε περιβάλλοντα βασισμένα στο νέφος.

## 2.2 Τύποι Cloud Services

Αρχικά, υπάρχουν τρεις κύριες κατηγορίες υπηρεσιών cloud: το Infrastructure as a Service (IaaS), το Platform as a Service (PaaS) και το Software as a Service (SaaS). Στη συνέχεια του κεφαλαίου, θα αναλύσουμε πιο λεπτομερώς κάθε μία από τις παραπάνω κατηγορίες υπηρεσιών cloud, εξετάζοντας τα χαρακτηριστικά, Και τα πλεονεκτήματα τους. Θα διερευνήσουμε τον τρόπο με τον οποίο κάθε κατηγορία συμβάλλει στην ευελιξία και την αποδοτικότητα. Η ανάλυση αυτή θα μας βοηθήσει να κατανοήσουμε πλήρως το ρόλο του cloud computing στη σύγχρονη πραγματικότητα και τις προοπτικές του για το μέλλον.

### 2.2.1 SaaS (Software as a Service)

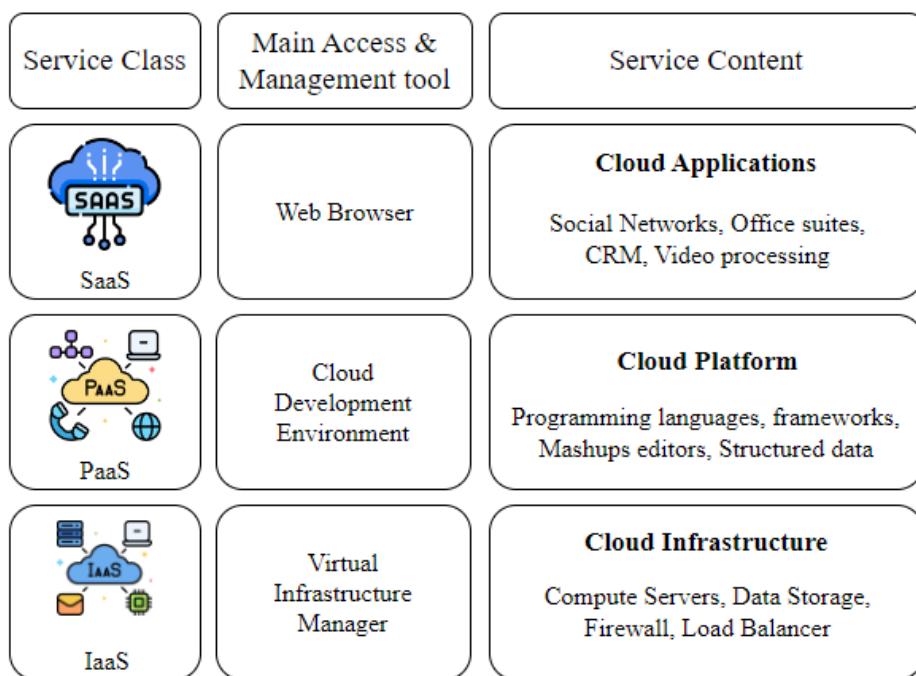
Το SaaS είναι ένας τύπος υπολογιστικού νέφους που παρέχει έτοιμες προς χρήση εφαρμογές λογισμικού μέσω του Διαδικτύου. Μέσω αυτού του μοντέλου παροχής υπηρεσιών, ο τελικός χρήστης χρησιμοποιεί τις υπηρεσίες εφαρμογών λογισμικού πάνω από το δίκτυο. Συνήθως οι πάροχοι SaaS διαχειρίζονται το software, τα updates, το security και την συντήρηση τους, και οι χρήστες έχουν πρόσβαση. Για παράδειγμα το Gmail είναι SaaS όπου η Google είναι πάροχος και εμείς οι clients. Συνοπτικά λοιπόν ως χαρακτηριστικά του SaaS είναι η πρόσβαση στο Διαδίκτυο σε εμπορικό λογισμικό, η διαχείριση του γίνεται από το κεντρική τοποθεσία, και η παράδοση του λογισμικού θεωρείται ως "ένα προς πολλά" μοντέλο [1].

### 2.2.2 IaaS (Infrastructure as a Service)

Το IaaS παρέχει εικονικούς υπολογιστικούς πόρους μέσω του Διαδικτύου. Αυτοί οι πόροι περιλαμβάνουν εικονικές μηχανές (virtual machines), αποθήκευση (storage) και δικτύωση (networking) ώς υπηρεσία κατ' απαίτηση. Είναι ο συνδιασμός δημόσιας (public) και ιδιωτικής (private) υποδομής ή και ως ατομική. Καθώς το IaaS αναπτύσσεται ραγδαία, σημαντικό να αναφερθούν τα χαρακτηριστικά του [1]: διανέμει πόρους ως υπηρεσία, επιτρέπεται η δυναμική κλιμάκωση των πόρων, το κόστος είναι ανάλογα το τι χρησιμοποιείται από τους πόρους, και πολλοί χρήστης ή πελάτες μπορούν να έχουν πρόσβαση στο ίδιο υλικό. Επιγραμματικά, ως πάροχοι IaaS είναι η Amazon (AWS), το Microsoft Azure, το Google cloud (GCP) και το Linode.

### 2.2.3 PaaS (Platform as a Service)

Το PaaS παρέχεται η πλατφόρμα για την ανάπτυξη, κωδικοποίηση και διαχείριση εφαρμογών στο Cloud. Η προσέγγιση είναι προς τα εργαλεία ανάπτυξης και το λογισμικό. Για παράδειγμα οι πάροχοι PaaS προσφέρουν διακομιστές εφαρμογών (java, .NET FrameWork), διακομιστές βάση δεδομένων (MySQL, Oracle), όπου ο client μπορεί να χρησιμοποιήσει για να δημιουργήσει τις δικές του εφαρμογές με τις ανάγκες που έχει [1]. Σε αυτό το μοντέλο, τα χαρακτηριστικά του είναι: μειώνει το κόστος ανάπτυξης και συντήρησης, αναπτύσσεται και γίνεται δοκιμή της εφαρμογής στο ίδιο περιβάλλον, παρέχει επεκτασιμότητα, αξιοπιστία και ασφάλεια. Οι πάροχοι PaaS περιλαμβάνουν το Heroku, το Google App Engine, το Azure App Service.



Εικόνα 2. Τύποι των Υπολογιστικών Υπηρεσιών

## 2.3 Τύποι Ανάπτυξης Cloud

Έχοντας αναφέρει πως ορίζεται το cloud, ποια είναι τα οφέλη του καθώς και ποιοι είναι οι τύποι ενός cloud service, τώρα θα χρειαστεί για την ανάπτυξη μίας λύσης υπολογιστικού νέφους, να αποφασίσουμε τον τύπο του cloud που θα εφαρμοστεί. Κατ’εξοχήν, υπάρχουν τέσσερα μοντέλα ανάπτυξης cloud computing που είναι διαθέσιμα στον πελάτη, και αυτά είναι:

- **Public Cloud** - Δημόσιο Νέφος
- **Private Cloud** - Ιδιωτικό Νέφος
- **Hybrid Cloud** - Υβριδικό Νέφος
- **Community Cloud** - Νέφος Κοινότητας

### 2.3.1 Δημόσιο Νέφος

Σε αυτό τον τύπο cloud, οι χρήστες έχουν την δυνατότητα μέσω κάποιων interfaces που χρησιμοποιούν προγράμματα περιήγησης ιστού, να τους δίνεται η πρόσβαση στο cloud. Η πληρωμή γίνεται μόνο για όσο κάνουν χρήση της υπηρεσίας. Κάλλιστα μπορεί να γίνει αντιπαραβολή με την ηλεκτρική ενέργεια που έχουμε σπίτι μας, καθώς και σε αυτό το σενάριο πληρώνουμε μόνο για την κατανάλωση που κάνουμε [1]. Συνεπώς, αυτό βοηθάει στην μείωση του κόστους λειτουργίας για τις δαπάνες της πληροφορικής. Σημαντικό να αναφερθεί πως το δημόσιο νέφος είναι λιγότερο ασφαλές από τα άλλα μοντέλα cloud που θα αναλύσουμε παρακάτω, καθώς όλα τα δεδομένα και οι εφαρμογές είναι επιρρεπή σε επιθέσεις από κακόβουλους χρήστες. Σαν λύση προτείνεται να γίνονται έλεγχοι ασφαλειας μέσω επικύρωσης μέσω του provider και του client. Το δημόσιο νέφος είναι εκτός προϋποθέσεων όπου οι επιχειρήσεις μπορούν να χρησιμοποιηθούν για την παροχή υπηρεσιών στον χρήστη παίρνοντας το από τρίτο μέρος.

### 2.3.2 Ιδιωτικό Νέφος

Στο ιδιωτικό νέφος η λειτουργία βρίσκεται στο εσωτερικό κέντρο δεδομένων επιχείρησης ενός ολόκληρου οργανισμού. Ένα από τα βασικά πλεονεκτήματα είναι η ευκολία στην διαχείριση της ασφάλειας, της συντήρησης και των αναβαθμίσεων και παρέχει επίσης έλεγχο στην ανάπτυξη και την χρήση. Συνεπώς μπορεί να συγκριθεί με το διαδίκτυο [1]. Συγκρίνοντας το με το δημόσιο νέφος όπου εφαρμογές και οι πόροι διαχειρόζονται από τον ίδιο τον provider, στο ιδιωτικό νέφος διατίθενται στον χρήστη σε οργανικό επίπεδο, καθώς διαχειρίζονται από τον ίδιο τον οργανισμό. Έτσι λοιπόν με αυτόν τον τρόπο η ασφάλεια ενισχύεται σημαντικά.

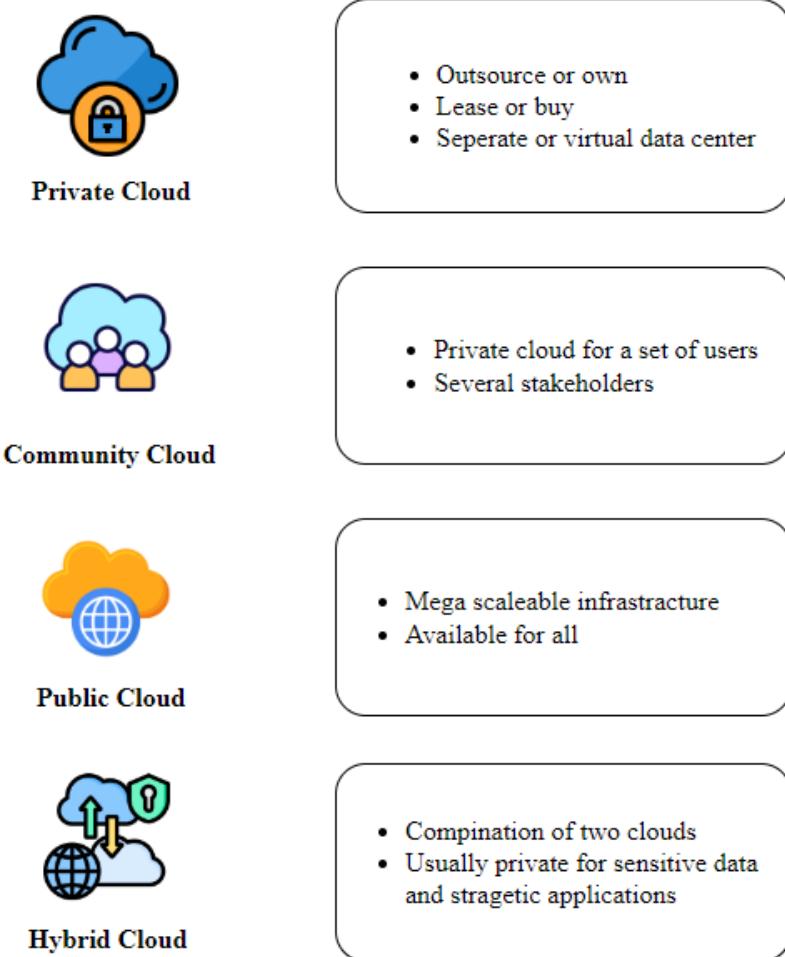
### 2.3.3 Υβριδικό Νέφος

Έχοντας αναλύσει τα δύο πρώτα μοντέλα νέφους, το υβριδικό νέφος αποτελείται από δημόσια και ιδιωτικά μοντέλα cloud, όπου η διαχείριση του υπολογιστικού περιβάλλοντος γίνεται από τρίτους, όμως συγκεκριμένοι πόροι χρησιμοποιούνται αποκλειστικά ιδιωτικά από έναν οργανισμό. Σε αυτό το μοντέλο, γίνεται σύνδεση με μία ή περισσότερες εξωτερικές υπηρεσίες cloud. Αυτός είναι ένας τρόπος, ώστε να υπάρχει ασφάλεια ελέγχου στα δεδομένα και τις εφαρμογές και επιτρέπει στο μέρος να έχει πρόσβαση στις πληροφορίες μέσω του διαδικτύου [1]. Έτσι, ο οργανισμός έχει την δυνατότητα να εξυπηρετήσει την

ανάγκη του σε ένα ιδιωτικό σύννεφο, και εάν παρουσιαστεί κάποια ανάγκη η οποία πρέπει να υλοποιηθεί, ζητά από το δημόσιο cloud για εν δυνάμῃ σύντομους υπολογιστικούς πόρους.

#### 2.3.4 Σύννεφο Κοινότητας

Κλείνοντας, με το community cloud δίνεται η δυνατότητα το περιβάλλον υπολογιστικού νέφους το οποίο είναι κοινόχρηστο ή διαχειριζόμενο από έναν αριθμό σχετικών οργανισμών, να συνδιάζουν την κατασκευή και την κοινή χρήση της υποδομής cloud [1], των απαιτήσεων και των πολιτικών τους. Η υποδομή cloud παρέχεται από άλλον πάροχο ή σε έναν από τους οργανισμούς που βρίσκεται μέσα στο σύννεφο κοινότητας.



Εικόνα 3. Τύποι Ανάπτυξης Cloud



## Κεφάλαιο 3

# Εξέλιξη της Διαχείριση Κύκλου ζωής Λογισμικού

Στο κεφάλαιο 3 εμβαθύνουμε στην ιστορική εξέλιξη και τη σημασία του Κύκλου Ζωής Ανάπτυξης Λογισμικού (SDLC), μιας δομημένης προσέγγισης για τον σχεδιασμό, τη δημιουργία, τη δοκιμή και την ανάπτυξη του λογισμικού. Παράλληλα, παρέχεται μια λεπτομερή εξέταση του τρόπου με τον οποίο οι μεθοδολογίες SDLC έχουν προσαρμοστεί για να ανταποκριθούν στις σύγχρονες απαιτήσεις των επιχειρηματικών ομάδων, θέτοντας τις βάσεις ώστε να παρέχουμε μια ολοκληρωμένη επισκόπηση των καινοτόμων τεχνολογιών που επιτρέπουν στις επιχειρήσεις να εξελιχθούν στο σημερινό γρήγορο, ανταγωνιστικό και επιχειρηματικό τοπίο. Έτσι λοιπόν, υιοθετώντας τις αρχές του DevOps και αξιοποιώντας τεχνολογίες αιχμής όπως η έννοια του IaC, και τα εργαλεία που προσφέρει που είναι το Terraform, Pulumi, Ansible και παράλληλα CI/CD pipelines, οι ομάδες μπορούν να βελτιστοποιήσουν κάθε στάδιο του κύκλου ζωής του λογισμικού, από τη διαχείριση της υποδομής μέχρι την ανάπτυξη.

### 3.1 Εισαγωγή

Στην σημερινή εποχή ο κύκλος ζωής ανάπτυξης λογισμικού έχει εξελιχθεί σημαντικά με την πάροδο των ετών, λαμβάνοντας υπόψιν τις ανάγκες που συνεχώς μεταβάλλονται, τις τεχνολογικές εξελίξεις, καθώς και την αυξανόμενη πολυπλοκότητα του ψηφιακού κόσμου. Το SDLC περιέχει ένα λεπτομερές σχέδιο για τον τρόπο ανάπτυξης, τροποποίησης, συντήρησης και ενημέρωσης του λογισμικού. Περιλαμβάνει διάφορα μοντέλα όπως waterfall, V-shaped, evolutionary prototype, spiral, iterative, και agile. Το μοντέλο SDLC ουσιαστικά διασφαλίζει την επιτυχία του έργου διαιρώντας τη διαδικασία ανάπτυξης σε εργασίες, επιτρέποντας στις εταιρίες πληροφορικής να διαχειρίζονται αποτελεσματικά τα software που δημιουργούν. Συνεπώς, μια ισχυρή προσέγγιση SDLC οδηγεί σε ένα ισχυρό τελικό προϊόν και επιτυχημένο έργο.

Αξίζει να σημειωθεί πως ο ρόλος του software developer έχει εξελιχθεί σημαντικά από την εφεύρεση των υπολογιστών, προσαρμόζοντας τις σύγχρονες τεχνολογίες σε hardware, development environment, και οργάνωση ομάδας. Σύμφωνα με την βιβλιογραφία και έρευνα που έχει γίνει, έχει διαπιστωθεί πως νέοι μέθοδοι εμφανίζονται παγκοσμίως, με στόχο την ανάπτυξη λογισμικού χωρίς μεγάλο κόστος, αποδοτικά και αποτελεσματικά. Σημαντικό να αναφερθεί πως το μοντέλο διαδικασίας ανάπτυξης λογισμικού αποτελείται από προδιαγραφές, σχεδιασμό, επικύρωση και εξέλιξη. Ο κύκλος ζωής ανάπτυξης λογισμικού περιλαμβάνει την κατανόηση του προβλήματος, τον σχεδιασμό μιας λύσης, την κωδικοποίηση της λύσης, τη δοκιμή του προγράμματος, και την συντήρηση του προιόντος. Αυτές

οι μέθοδοι στοχεύουν στην ανάπτυξη λογισμικού όσο το δυνατόν φθηνότερα, αποδοτικά και αποτελεσματικά.

### 3.2 Ιστορική εξέλιξη του SDLC

Αρχικά, η εξέλιξη του SDLC μπορεί να ανιχθευθεί κατά την διάρκεια των δεκαετιών 1940 και 1950 που έκαναν την εμφάνιση τους και οι πρώτοι υπολογιστές, όπου η ανάπτυξη του λογισμικού ήταν μία χειροκίνητη και εξαιρετικά τεχνική διαδικασία. Σε εκείνη την εποχή, οι προγραμματιστές έγραφαν οδηγίες σε επίπεδο μηχανής με το χέρι. Στις δεκαετίες του 1950 και 1960, εμφανίστηκαν για πρώτη φορά οι γλώσσες προγραμματισμού υψηλού επιπέδου όπως οι Fortran, COBOL και LISP, όπου και έφεραν την επανάσταση στην ανάπτυξη του λογισμικού, κάνοντας την κωδικοποίηση πιο προσιτή σε όλες τις απαραίτητες διαδικασίες. Στην συγκεκριμένη περίοδο είδαμε επίσης να αναπτύσσονται μεταγλωττιστές και διερμηνείς που μετέφεραζαν τον κώδικα υψηλού επιπέδου σε κώδικα μηχανής, απλοποιώντας τη διαδικασία κωδικοποίησης. Έτσι, οι εφαρμογές, τα συστήματα διαχείρισης βάσεων δεδομένων και τα πρώτα λειτουργικά συστήματα άρχισαν να αποκτούν μια δυνατή θέση κατά την διάρκεια εκείνων των ετών.

Επιπροσθέτως, όταν εμφανίστηκαν οι πρώτοι προσωπικοί υπολογιστές στις δεκαετίες 1970 και 1980, δημιούργησε την δυνατότητα να μπορεί να αναπτυχθεί το λογισμικό και τα συστήματα σε ένα ευρύ κοινό, οδηγώντας έτσι στην επέκταση στους οικιακούς υπολογιστές, τα παιχνίδια, κ.α. Παρατηρείται πως οι γραφικές διεπαφές χρήστη όπως τα Windows και το Macintosh OS βελτίωσαν την εμπειρία χρήστη, προάγοντας περαιτέρω τις πρακτικές ανάπτυξης του λογισμικού.

Στη δεκαετία του 1990, δημιουργήθηκε ο Παγκόσμιος Ιστός από τον Tim Berners-Lee και ήταν αναμενόμενο να φέρει την επανάσταση στην ανάπτυξη του λογισμικού με την γέννηση των διαδικτυακών εφαρμογών. Η JavaScript έγινε μία γλώσσα σημαντική για την ανάπτυξη ιστού και η κυκλοφορία των Windows 95 από την Microsoft έκανε τις γραφικές διεπαφές χρήστη το πρότυπο για προσωπικούς υπολογιστές.

Στην δεκαετία του 2000, η εμφάνιση του Mac OS X από την Apple, της κυκλοφορίας του Apple App Store, σηματοδότησε την εμφάνιση των εφαρμογών στα κινητά. Και φτάνοντας στην δεκαετία του 2010, παρατηρήθηκαν πως τεχνικές και προσεγγίσεις DevOps έγιναν ευρέως γνωστές, προωθώντας πρακτικά τη συνεργασία μεταξύ ανάπτυξης λογισμικού και των λειτουργιών πληροφορικής. Συνολικά, παρατηρείται πως η εξέλιξη του SDLC είναι μια διαδρομή σαφώς καινοτόμα, με τεχνολογικές προόδους και συνεχώς μεταβαλλόμενων πρακτικών και μεθόδων στην ανάπτυξη του λογισμικού, διαμορφώντας τον τρόπο με τον οποίο ένα λογισμικό σχεδιάζεται, αναπτύσσεται, δοκιμάζεται, και διατείται στο μεταβαλλόμενο τεχνολογικό κόσμο.

### 3.2.1 Φάσεις του SDLC

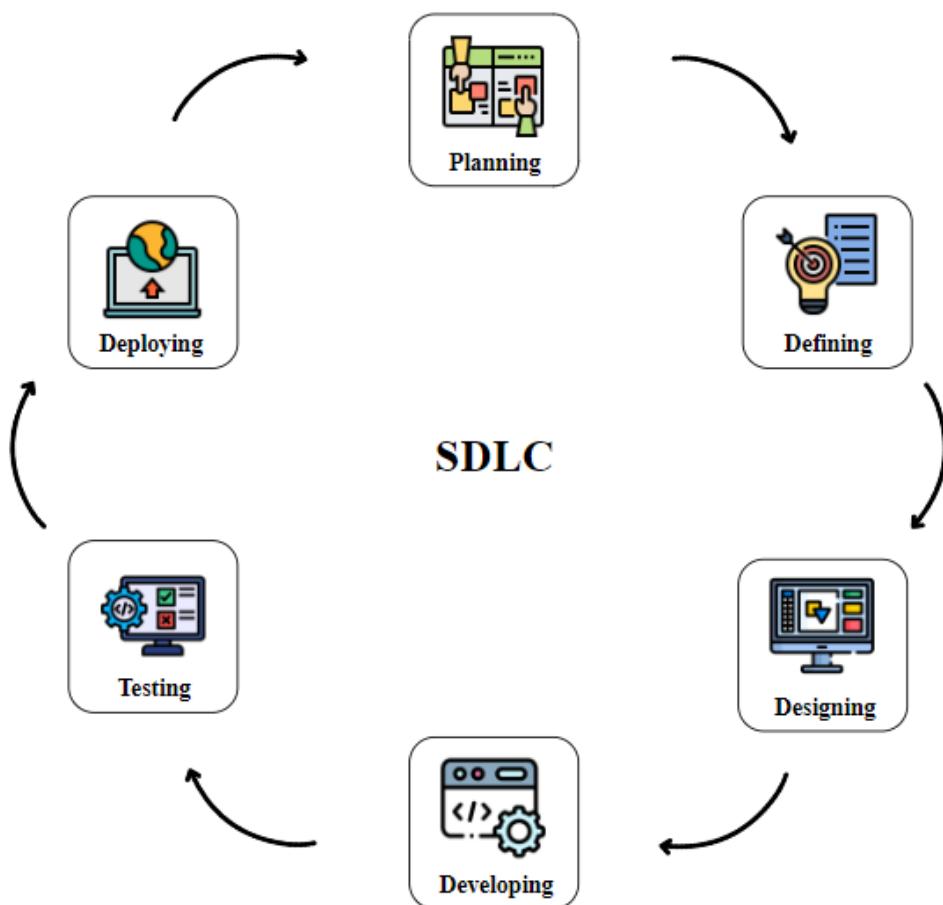
Συνεχίζοντας σε αυτήν την υποενότητα, είναι σημαντικό να αναφερθούν οι φάσεις που έχει το SDLC. Μία απλοποιημένη και ασφαλής ανάπτυξη ενός λογισμικού σίγουρα απαιτεί μια μεθοδολογία και μία καλά καθορισμένη διαδικασία μετάβασης από την ιδέα στο τελικό προϊόν. Συνεπώς, όπως αναφέραμε και πιο πάνω, το SDLC μπορούμε να πάνε πως είναι μια σειρά βημάτων που ακολουθούνται για την κατασκευή, τροποποίηση και συντήρηση ενός λογισμικού. Παρακάτω αναλύονται με σαφήνεια οι φάσεις που διέπουν το SDLC:

- **Planning :** Αρχικά, η φάση του σχεδιασμού περιλαμβάνει την ανάλυση του έργου σε συνδιασμό με τον σκοπό που θα έχει το λογισμικό. Παράλληλα σε αυτή την φάση υπάρχει η δυνατότητα της καταγραφής των αποφάσεων που θα πρέπει να διεκπερευθούν, όπως φυσικά και τα σχέδια ανάπτυξης του λογισμικού. Καταλαβαίνουμε λοιπόν πως είναι αυτό το πρώτο στάδιο είναι το πιο θεμελιώδες του SDLC και πολλές φορές ονομάζεται και ως "Συλλογή Απαιτήσεων". Η ομάδα του έργου θα χρειαστεί να συνεργαστεί με όλους τους ενδιαφερόμενους φορείς του έργου για να δημιουργήσει το συνολογικό χρονοδιάγραμμα του έργου και να προσδιοριστούν οι πόροι που απαιτούνται, καθώς και το κόστος. Στην συνέχεια, έχοντας συλλέξει όλες αυτές τις πληροφορίες θα πρέπει η ομάδα να είναι σε θέση να αναλύσει το κόστος-όφελος του έργου ώστε στην συνέχεια να μπορεί να επικυρωθεί η οικονομική και τεχνική σκοπιμότητα της συνέχισης του έργου. Σε αυτή την φάση του σχεδιασμού, εντοπίζονται οι κίνδυνοι, με την ομάδα να είναι σε θέση να αναλύσει τι μπορεί να πάει στραβά και να βρει τις λύσεις για την μείωση του αντίκτυπου αυτών των κινδύνων [3].
- **Defining :** Στην φάση του καθορισμού, με την συλλογή των απαιτήσεων που έχει γίνει στην πρώτη φάση, πλέον υπάρχει η ανάλυση τους. Πιο συγκεκριμένα, περιλαμβάνει τη συγκέντρωση λειτουργικών και μη λειτουργικών απαιτήσεων λογισμικού, ώστε να δημιουργηθεί μια πλήρη τεκμηρίωση τους. Πρακτικά δίνεται μία ανάλυση από την συλλογή πληροφοριών που έχει γίνει για το τι πρέπει να κάνει το λογισμικό, ποιες είναι οι δυνατότητες και η λειτουργικότητα του λογισμικού, αλλά και το πώς πρέπει να λειτουργεί με τυχόν άλλες απαιτήσεις που θα εμφανιστούν. Επιπλέον, στην διάρκεια αυτής της φάσης δύναται να καθορίζονται οι συγκεκριμένες απαιτήσεις υλικού και λογισμικού που απαιτούνται για την ομάδα της ανάπτυξης, να προσδιορίζεται ο τύπος περιβάλλοντος του προγραμματισμού που απαιτείται, να υπάρχει ο ορισμός της γλώσσας προγραμματισμού που θα χρησιμοποιηθεί, καθώς και να ορίσουμε τους τεχνικούς πόρους που απαιτούνται για την ολοκλήρωση του έργου. Σημαντικό σε αυτή την φάση είναι να ορίσουμε όλες εκείνες τις απαιτήσεις ασφαλείας του λογισμικού μας να προσδιοριστούν τα εργαλεία και τους πόρους που απαιτούνται για την ανάπτυξη τους. Παράλληλα θα χρειαστεί να αναλύσουμε που απαιτείται κρυπτογράφηση, ποιος θα είναι ο τύπος λειτουργιών ελέγχου πρόσβασης και ποιες είναι οι απαιτήσεις για την διατήρηση της ακεραιότητας του κώδικα μας [3].
- **Designing :** Η φάση της σχεδίασης εστιάζει σε λεπτομερείς δυνατότητες και των προδιαγραφών του λογισμικού, ώστε να δημιουργηθεί μία πραγματική διαδικασία σχεδιασμού που θα είναι σε θέση να αναλύσει όλες τις αλληλεπιδράσεις που υπάρχουν μεταξύ του λογισμικού ως προϊόν, αλλά και να χρησιμοποιηθεί το σχέδιο αυτό κατά την επόμενη φάση για να μπορέσει να είναι μία άρτια καθοδήγηση για την πραγματική ανάπτυξη και εφαρμογή του λογισμικού. Κατά την φάση του σχεδιασμού, οι προγραμματιστές, οι αρχιτέκτονες των συστημάτων και όλο το υπόλοιπο

τεχνικό προσωπικό δημιουργούν το σχεδιασμό συστήματος και λογισμικού υψηλού επιπέδου για να ανταποκρίνεται σε κάθε προσδιορισμένη απαίτηση. Κύριος στόχος σε αυτή την φάση είναι η σχεδίαση της συνολικής αρχιτεκτονικής του λογισμικού και να δημιουργηθεί το σχέδιο που θα προσδιορίζει τις τεχνικές λεπτομέριες του σχεδιασμού του λογισμικού μας. Για παράδειγμα, στην ανάπτυξη cloud, αυτή η φάση περιλαμβάνει τον καθορισμό της απαιτούμενης ποσότητας πυρήνων CPU, μνήμης RAM και εύρους ζώνης, ενώ προσδιορίζει επίσης ποιες υπηρεσίες cloud απαιτούνται για την πλήρη λειτουργικότητα του λογισμικού. Αυτό το σημείο είναι σημαντικό γιατί μπορεί να εντοπιστούν ανάγκες για πρόσθετους πόρους cloud. Το σχέδιο θα πρέπει να παρουσιάζει όλα τα στοιχεία λογισμικού που θα χρειαστεί να δημιουργηθούν, τις διασυνδέσεις με συστήματα, τη διεπαφή χρήστη στο front-end και όλες τις ροές δεδομένων τόσο εντός του λογισμικού, όσο και μεταξύ των χρηστών του λογισμικού. Τέλος, σημαντικό σε αυτό το στάδιο είναι ο προσδιορισμός πλήρως με τους πιθανούς κινδύνους [3].

- **Developing :** Κατά τη διάρκεια αυτής της φάσης του SDLC, η ομάδα των προγραμματιστών ή αλλιώς development team, διαχωρίζει όλη την ανάλυση που έγινε στις προηγούμενες φάσεις σε ενότητες που κωδικοποιούνται ξεχωριστά. Οι προγραμματιστές των βάσεων δεδομένων δημιουργούν την απαιτούμενη αρχιτεκτονική αποθήκευσης δεδομένων, οι προγραμματιστές front-end δημιουργούν τη διεπαφή του λογισμικού όπου θα αλληλεπιδρούν οι χρήστες της εφαρμογής και οι προγραμματιστές back-end κωδικοποιούν τις εσωτερικές λειτουργίες της εφαρμογής. Καταλαβαίνουμε πως αυτή η φάση του SDLC είναι η μεγαλύτερη όπου με οδηγό τα προηγούμενα βήματα, παύει να είναι περίπλοκη διαδικασία. Σε αυτή τη φάση η ομάδα ανάπτυξης θα πρέπει να ελέγχουν για ελλατώματα στον κώδικα, κάθε μεμονωμένο σημείο θα χρειαστεί να δοκιμαστεί για να επαληθεύσει την λειτουργικότητα του πριν ενσωματωθεί στο μεγαλύτερο έργο. Είναι σημαντικό να μην παρακάπτεται αυτό το βήμα ώστε να μην ελέγχεται συνολικά το λογισμικό αφού έχουν ολοκληρωθεί τα επιμέρους σημεία. Συνεπώς, εκτός από τις δοκιμές που γίνονται σε κάθε ενότητα, είναι μια σωστή στιγμή να υπάρξει και η δοκιμή της ασφάλειας πριν εισαχθεί στο έργο, ώστε να αποφευχθούν τυχόν ευπάθειες στο λογισμικό στο πυρήνα του κώδικα που μπορεί να παρουσιάσουν συνολικό κίνδυνο [3].
- **Testing :** Η φάση της δοκιμής διασφαλίζει ότι το λογισμικό το οποίο δημιουργείται, λειτουργεί όπως αναμένεται και πληροί όλες τις απαιτήσεις που χρειάζονται. Συνεπώς, μόλις αναπτυχθεί ο κώδικας του λογισμικού, η ομάδα προγραμματιστών που δημιουργούν το testing της εφαρμογής επιδιώκει να ελέγχει εάν η εφαρμογή λειτουργεί όπως έχει αποσαφηνιστεί στις προηγούμενες φάσεις με τις απαιτήσεις που έχουν καταγραφεί. Ο βασικός στόχος εδώ είναι να αποκαλυφθούν όλα τα ελλατώματα του λογισμικού και να δωθεί το κατάλληλο feedback στην ομάδα ανάπτυξης του λογισμικού για διόρθωση. Αυτή η διαδικασία συνεχίζεται μέχρις ότου επικυρωθούν όλες οι απαιτήσεις του λογισμικού και διορθωθούν όλα τα ελλατώματα που μπορεί να έχει. Είναι λοιπόν να σημειωθεί πως σε αυτό το στάδιο της δοκιμής διαπραγματεύομαστε τις πιο κρίσιμες φάσεις του SDLC, καθώς είναι η κύρια είσοδος μεταξύ της ομάδας ανάπτυξης και των πελατών μας. Οι δοκιμές στον κώδικα θα πρέπει να διεξάγονται σύμφωνα με ένα σχέδιο δοκιμών λογισμικού που θα προσδιορίζει τι και πώς να δοκιμαστεί και θα έχει εγκριθεί από τα ενδιαφερόμενα μέρη του έργου [3].

- **Deploying and Maintaining** : Τέλος, η ανάπτυξη και συντήρηση περιλαμβάνει την εγκατάσταση του λογισμικού στο περιβάλλον στο οποίο έχει αποφασιστεί, και τη διασφάλιση της σωστής και άρτιας λειτουργικότητας του. Μόλις το λογισμικό περάσει την φάση της δοκιμής είναι έτοιμη ώστε να μπορέσει να την χρησιμοποιήσει ο πελάτης. Υπάρχουν πολλά στάδια ανάπτυξης όπως το Alpha, Beta, Γενική Διαθεσιμότητα (General Availability) όπου για παράδειγμα οι εκδόσεις Alpha τείνουν να αναπτύσσονται σε επιλεγμένους πελάτες, ενώ η Γενική Διαθεσιμότητα σημαίνει πως το λογισμικό είναι έτοιμο για όλους. Μόλις το λογισμικό δοκιμαστεί και αναπτυχθεί επιτυχώς, τότε εισέρχονται στην φάση συντήρησης όπου παρακολουθούνται και ενημερώνονται συνεχώς. Σε αυτό το σημείο το λογισμικό υποβάλλεται σε έναν συνεχή κύκλο της διαδικασίας SDLC, όπου οι ενημερώσεις κώδικα περνούν πάλι από όλες τις φάσεις που συζητήθηκαν στα προηγούμενα στάδια [3].



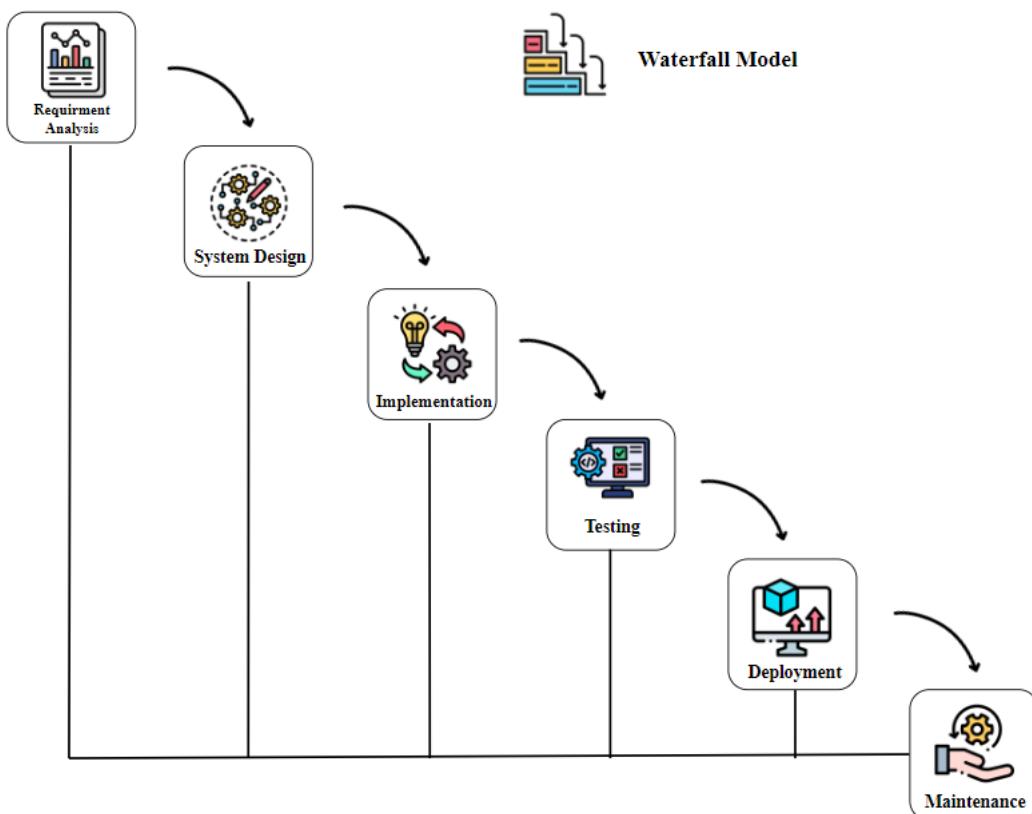
Εικόνα 4. Φάσεις του Software Development LifeCycle

### 3.3 Εξέλιξη των μεθοδολογιών του SDLC

#### 3.3.1 Παραδοσιακά μοντέλα του SDLC

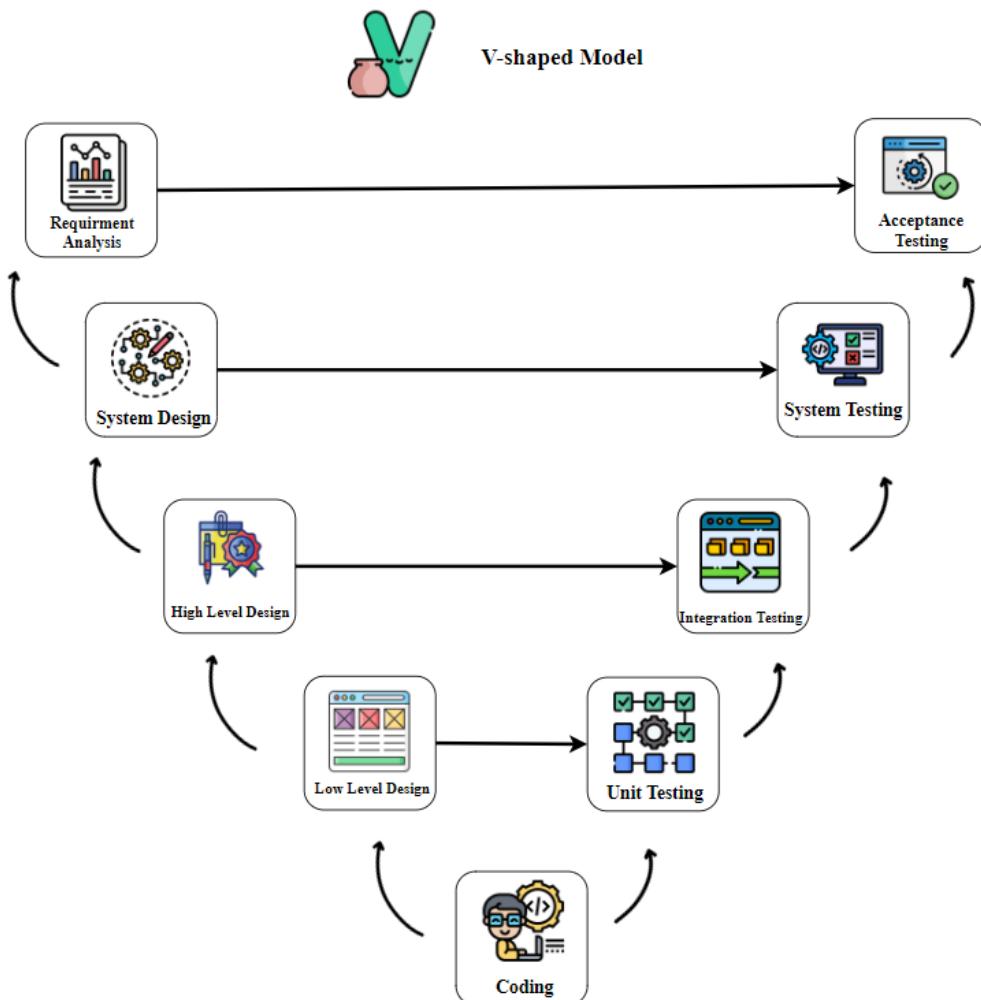
Σε αυτή την υποενότητα και αφού έχουμε διερευνήσει τις φάσεις του Κύκλου ζωής ανάπτυξης συστημάτων λογισμικού (SDLC), είναι απαραίτητο να εξεταστούν τα παραδοσιακά μοντέλα. Αυτά τα μοντέλα έχουν υιοθετηθεί ευρέως και έχουν παίξει ένα σημαντικό ρόλο στη διαμόρφωση της βιομηχανίας της ανάπτυξης του λογισμικού.

- **Waterfall Model :** Αυτό το μοντέλο είναι το πιο διαδεδομένο και αποδεκτό του SDLC. Με το waterfall η διαδικασία ανάπτυξης των συστημάτων του λογισμικού χωρίζεται σε διάφορα στάδια. Χαρακτηρίζεται ως ένα μοντέλο συνεχούς ανάπτυξης του λογισμικού καθώς φαίνεται πως η διαδικασία ανάπτυξης είναι συνεχώς προς τα κάτω μέσω της ανάλυσης των απαιτήσεων, του σχεδιασμού, της υλοποίησης, της δοκιμής, της ολοκλήρωσης και όλων των βημάτων ενσωμάτωσης για την διαχείριση. Πιο συγκεκριμένα, περιλαμβάνει τον διαχωρισμό της διαδικασίας της ανάπτυξης σε διακριτά στάδια, με κάθε στάδιο να βασίζεται στο προηγούμενο. Κάθε στάδιο ολοκληρώνεται και μετέπειτα προχωράμε στο επόμενο και αυτό δίνει την δυνατότητα της άρτιας διαδικασίας ανάπτυξης του λογισμικού. Σημαντικό να αναφερθεί πως σε διάφορα έργα μεγάλης κλίμακας όπου οι απαιτήσεις είναι καθορισμένες και το πεδίο εφαρμογής είναι σαφές, η προσέγγιση του waterfall χρησιμοποιείται για την υλοποίηση του έργου [4].



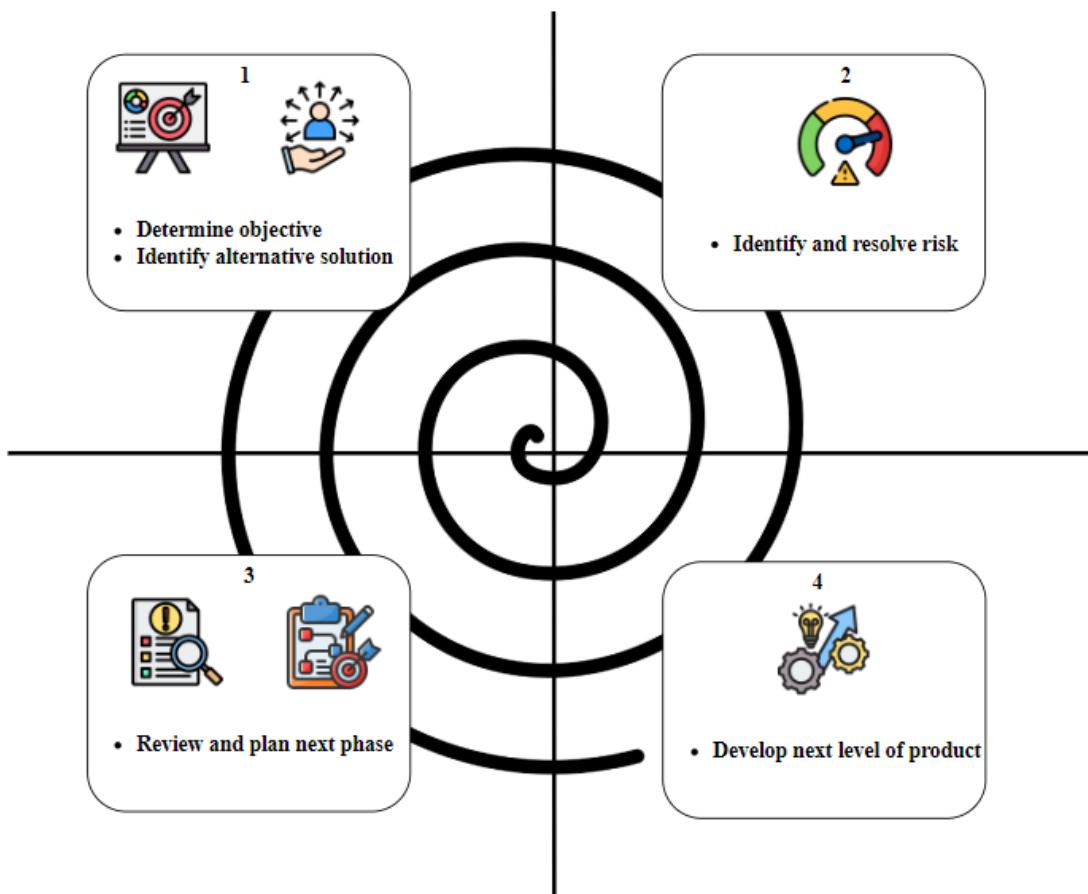
Εικόνα 5. Waterfall μοντέλο

- **V-model** : Η μεθοδολογία V-model είναι ένας συνδιασμός των μοντέλων Warerfall και Spiral. Πιο συγκεκριμένα, αυτό το μοντέλο του SDLC περιλαμβάνει τη διάσπαση της διαδικασίας ανάπτυξης των συστημάτων του λογισμικού σε συγκεκριμένες φάσεις, με κάθε φάση να βασίζεται στην προηγούμενη. Η βασική διαφοροποίηση όμως είναι πως το V-model επιτρέπει την επαναληπτική και σταδιακή διαδικασία της ανάπτυξης, με κάθε φάση να αναπτύσσεται και να δοκιμάζεται πριν προχωρίσουμε στην επόμενη [4].



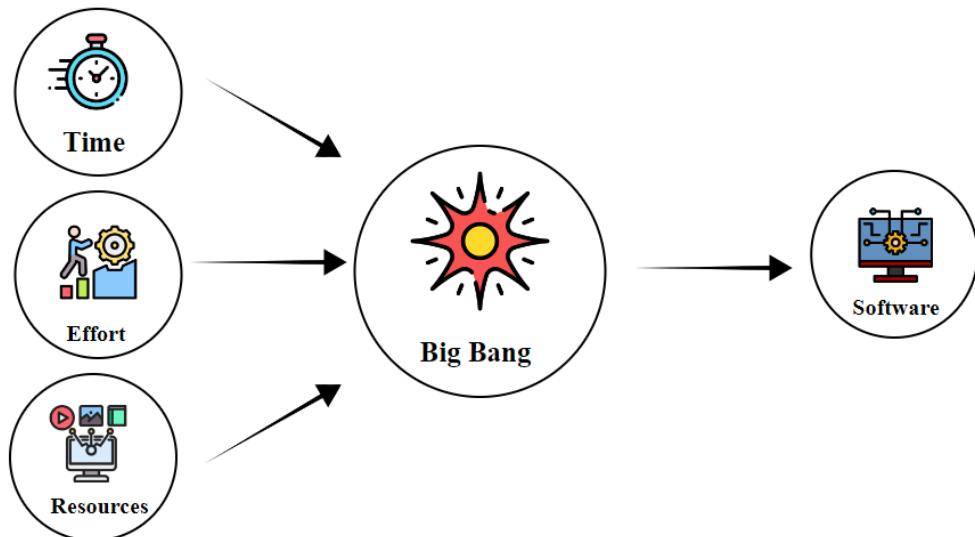
Εικόνα 6. V-shaped μοντέλο

- **Spiral Model :** Αυτό το μοντέλο βασίζεται σε μοντέλο κινδύνου. Αυτό το μοντέλο του SDLC επιτρέπει στην ομάδα να επιλέξει τη διάσπαση της διαδικασίας ανάπτυξης σε μικρότερα στάδια, τα οποία είναι διαχειρίσιμα, και με κάθε στάδιο να αναπτύσσεται και να δοκιμάζεται πριν προχωρήσουμε στο επόμενο. Η μεθοδολογία spiral χαρακτηρίζεται από την ευελιξία και την προσαρμοστικότητα της, επιτρέποντας αλλαγές και τροποποιήσεις σε όλη τη διαδικασία ανάπτυξης. Συνεπώς, είναι ένας συνδιασμός γρήγορου σχεδιασμού και ενσωμάτωσης στο σχεδιασμό και την κατασκευή του έργου που είναι προς υλοποίηση. Έτσι, κάθε στάδιο στο spiral μοντέλο, ξεκινάει με τον καθορισμό του στόχου του σταδίου, των πιθανών μεταβλητών για την επίτευξη του στόχου και των περιορισμών που μπορεί να υπάρχουν [4].



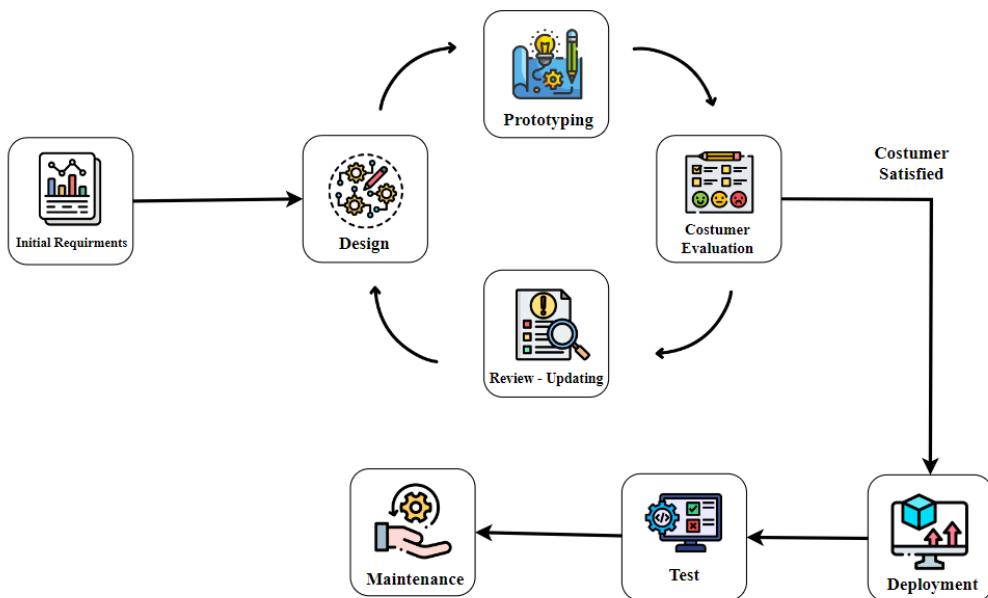
Εικόνα 7. Spiral μοντέλο

- **Big Bang Model :** Αυτό το μοντέλο επικεντρώνεται στην ανάπτυξη των συστημάτων λογισμικού και κωδικοποίησης με ελάχιστο ή και καθόλου σχεδιασμό. Καταλαβαίνουμε λοιπόν πως αυτό το μοντέλο είναι πιο χρήσιμο και καλύτερο για μικρότερης κλίμακας έργα όπου μικρές ομάδες συνεργάζονται. Είναι επίσης χρήσιμο όταν η ζήτηση είναι άγνωστη ή δεν υπάρχει τελική ημερομηνία κυκλοφορίας συνολικότερα του έργου που έχει ανατεθεί στις ομάδες υλοποίησης [4].



Εικόνα 8. Big Bang μοντέλο

- **Prototype Model :** Αρχικά στο μοντέλο αυτό γίνονται όλες οι απαραίτητες ενέργειες ώστε να γίνει η συλλογή των απαιτήσεων. Έτσι οι προγραμματιστές και οι χρήστες καθορίζουν τον σκοπό των συστημάτων του λογισμικού, ορίζουν κανόνες και θέτουν τις συνολικότερες βάσεις για την υλοποίηση [4].

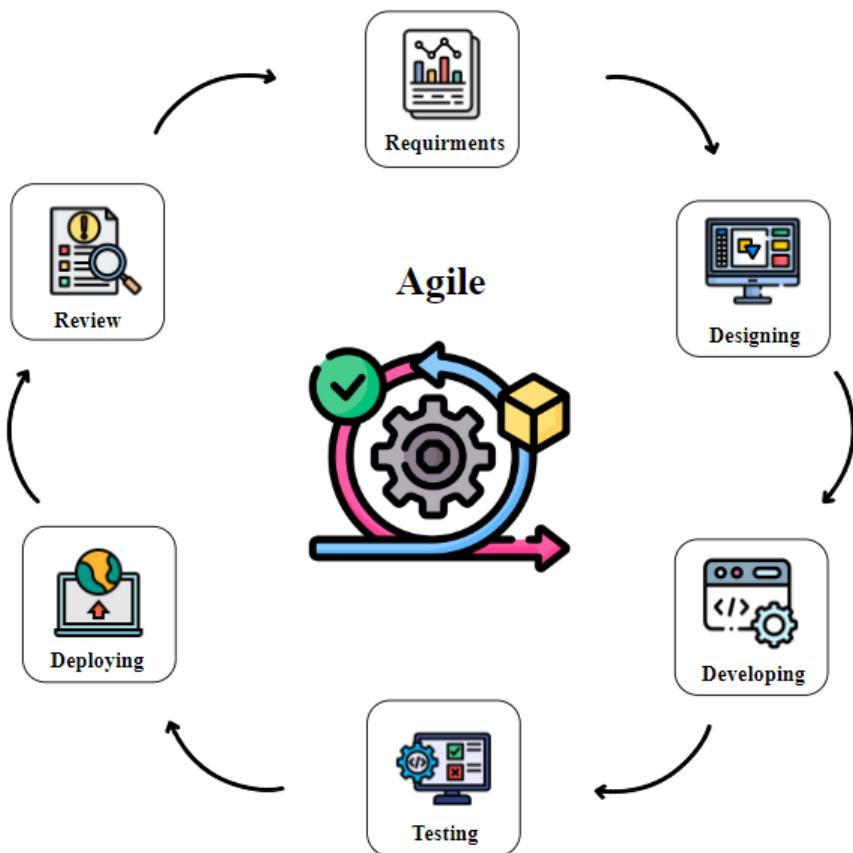


Εικόνα 9. Prototype μοντέλο

### 3.3.2 Μοντέρνα μοντέλα του SDLC

Συνεχίζοντας την ανάλυση, είναι σημαντικό να αναφερθεί πως οι παραδοσιακές μεθοδολογίες SDLC, όπως είναι το Waterfall, το Spiral και το RAD έχουν χρησιμοποιηθεί ευρέως στην ανάπτυξη των συστημάτων του λογισμικού. Ωστόσο, αυτές οι μέθοδοι έχουν τους περιορισμούς τους, ιδιαίτερα όσον αφορά την ευελιξία και την προσαρμοστικότητα. Συνεπώς, τα τελευταία χρόνια έχουν δημιουργηθεί νέες ανάγκες και έχουν εμφανιστεί πιο σύγχρονες μεθοδολογίες του SDLC για την αντιμετώπιση των περιορισμών και των αναγκών. Αυτά τα σύχρονα μοντέλα δίνουν προτεραιότητα στην ικανοποίηση των πελατών που ζητούν να δημιουργηθεί ένα έργο, την ομαδική συνεργασία και την επαναληπτική ανάπτυξη, επιτρέποντας έτοι μεγαλύτερη ευελιξία στην ανάπτυξη των συστημάτων λογισμικού. Σε αυτήν την υποενότητα θα αναλύσουμε μερικές από τις νέες μεθοδολογίες του SDLC που έχουν γίνει δημοφιλείς τα τελευταία χρόνια.

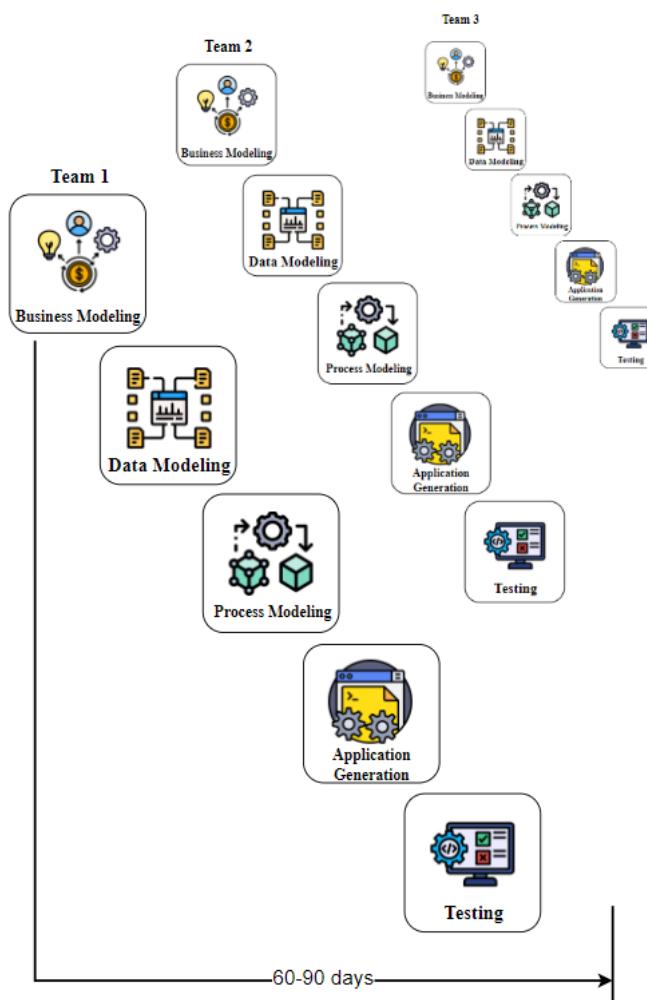
- **Agile Model :** Η μέθοδος Agile είναι μια πρακτική όπου επιτρέπει μεταξύ ανάπτυξης και δοκιμών των συστημάτων λογισμικού κατά την διάρκεια του SDLC να υπάρχει αλληλεπίδραση. Έτοι σε αυτό το μοντέλο, το συνολικό έργο αναπτύσσεται σε μικρά βήματα και τα σχέδια παραδίδονται σε επαναλήψεις, με κάθε επανάληψη να διαρκεί μία έως τρεις εβδομάδες. Σημαντικό να αναφερθεί είναι ότι, τα χαρακτηριστικά της Agile μεθόδου θα αλλάξουν, καθώς είναι δύσκολο να προβλεφθεί πώς θα αλλάξουν οι βασικοί πελάτες καθώς προχωρά το έργο [4].



Εικόνα 10. Agile μοντέλο

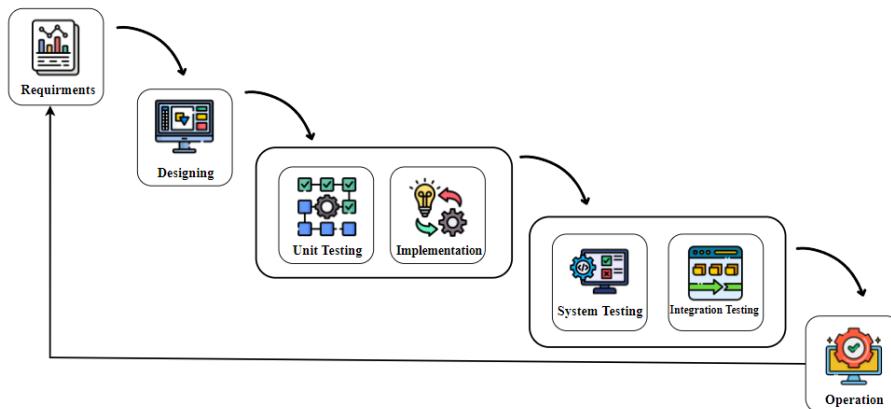
- **RAD Model** : Η μεθοδολογία RAD ή αλλιώς ταχείας ανάπτυξης είναι η υιοθέτηση του μοντέλου Waterfall. Βασικός σκοπός είναι να γίνει ανάπτυξη του λογισμικού σε σύντομο χρονικό διάστημα [4]. Το μοντέλο RAD χρησιμοποιεί την ιδέα ότι καλύτερα συστήματα μπορούν να αναπτυχθούν και να υλοποιηθούν σε πιο λίγο χρόνο χρησιμοποιώντας ομάδες εστίασης για τη καταγραφή των απαιτήσεων του εκάστοτε έργου. Πιο συγκεκριμένα, οι ομάδες αυτές είναι:

| Groups                 | Ομάδες                       |
|------------------------|------------------------------|
| (Business Modeling)    | Επιχειρηματική μοντελοποίηση |
| (Data Modeling)        | Μοντελοποίηση δεδομένων      |
| (Process Modeling)     | Μοντελοποίηση διαδικασιών    |
| (Application Creation) | Δημιουργία εφαρμογών         |
| (Testing and Return)   | Δοκιμή και επιστροφή         |



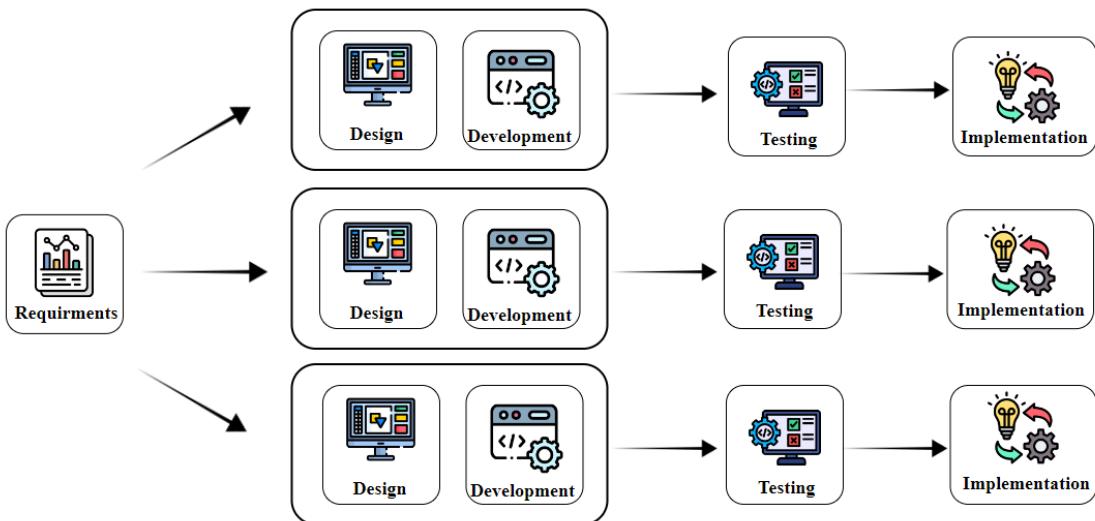
Εικόνα 11. RAD μοντέλο

- **Incremental Model :** Σε αυτό το μοντέλο θα πρέπει να υπάρχει μια σειρά βημάτων Waterfall. Αρχικά, οι απαιτήσεις του συνολικού έργου χωρίζονται σε ομάδες. Το λογισμικό για κάθε ομάδα αναπτύσσεται σύμφωνα με το μοντέλο SDLC. Επαναλαμβάνεται η διαδικασία SDLC, προσθέτοντας χαρακτηριστικά με κάθε έκδοση, μέχρι να ικανοποιηθούν όλες οι απαραίτητες απαιτήσεις του έργου. Έτσι με αυτόν τον τρόπο, κάθε κύκλος λειτουργεί ως στάδιο συντήρησης για την προηγούμενη έκδοση του λογισμικού. Οι αλλαγές σε αυτό το μοντέλο επιτρέπουν την επικάλυψη της ανάπτυξης του λογισμικού και μετέπειτα ο επόμενος κύκλος ξεκινάει πριν ολοκληρωθεί ο προηγούμενος κύκλος [4].



Εικόνα 12. Incremental μοντέλο

- **Iterative Model :** Στο μεντέλο αυτό, ο κύκλος ζωής ανάπτυξης των συστημάτων λογισμικού επικεντρώνεται στην αρχικοποίηση, και στην απλή υλοποίηση της. Μετά, υπάρχει η σταδιακή αύξηση της πολυπλοκότητας και του πεδίου εφαρμογής μέχρι να ολοκληρωθεί η συνολική ανάπτυξη του έργου (refactoring) [4].



Εικόνα 13. Iterative μοντέλο

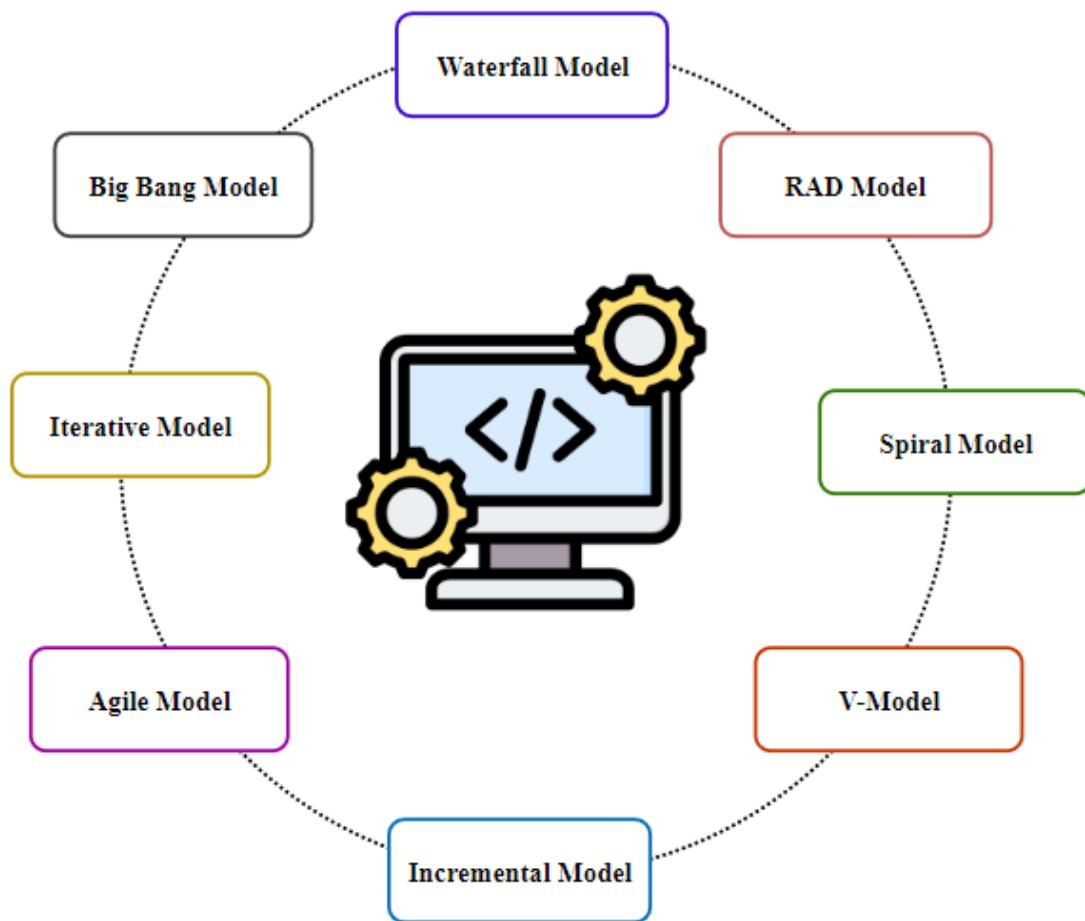
### 3.3.3 Σύγκριση παραδοσιακών - μοντέρνων Μοντέλων SDLC

Έχοντας αναλύσει τα μοντέλα SDLC (παραδοσιακά και μοντέρνα) που υπάρχουν, παρατηρούμε πως έχουν δυνατά και αδύνατα σημεία. Αρχικά, θα μπορούσε να πει κανείς πως τα παραδοσιακά μοντέλα SDLC έχουν αν μη τι άλλο παίξει σημαντικό ρόλο στην διαμόρφωση του "πώς" και "γιατί" μέσα στην βιομηχανία ανάπτυξης των συστημάτων λογισμικού, και η επιλογή του μοντέλου που θα χρησιμοποιηθεί εξαρτάται αρκετά από τις συγκεκριμένες ανάγκες και απαιτήσεις του έργου.

Οι Agile μεθοδολογίες παρατηρούμε πως διαφέρουν αρκετά από την λογική της παραδοσιακής SDLC ως προς την ιεράρχηση των προτεραιοτήτων του πελάτη, και την αναμονή αλλαγών ανά πάσα στιγμή, ώστε να μπορεί να ικανοποιήσει τις νέες απαιτήσεις. Έτσι, αντί να συγκεντρώσει όλες τις απαιτήσεις, το σύστημα χρειάζεται ως πρώτο βήμα, οι Agile μέθοδοι να μπορέσουν να συγκεντρώσουν όλα τα δεδομένα για να ξεκινήσουν και να δείξουν κάτι στους πελάτες. Με αυτόν τον τρόπο, δίνεται η δυνατότητα στον πελάτη να επέμβει και να παρέχει σχόλια και διορθώσεις που τυχόν χρειάζονται όσο τον δυνατόν νωρίτερα, κατανοώντας από την αρχή την πορεία της εφαρμογής σε όλα τα στάδια που έχει προς υλοποίηση. Συνεπώς, η μεθοδολογία απαιτεί τη συμμετοχή των πελατών κατά την ανάπτυξη και την διατήρηση του συστήματος για την παροχή έγκαιρης ανατροφοδότησης.

Από την άλλη μεριά, μια πιο παραδοσιακή μεθοδολογία SDLC φαίνεται να είναι πιο κατάλληλη όταν ο πελάτης δεν είναι άμεσα διαθέσιμος να απαντήσει σε ερώτησεις που τυχόν θα προκύψουν κατά την διάρκεια όλης της διαδικασίας. Παρατηρώντας την αγορά του σήμερα, είναι εύκολο να αντιληφθεί κανείς πως με τις ραγδαίες εξελίξεις που υπάρχουν συνολικότερα σε όλους τους τομείς, δεν μπορεί να περιμένει πολύ για να παραδοθεί ένα σύστημα, γεγονός που είναι και ο λόγος όπου οι περισσότερες εφαρμογές κατασκευάζονται και αναπτύσσονται σήμερα, με Agile μεθόδους. Έτσι, η λήψη σημαντικών και στοχευμένων σχολίων από την αγορά και η συστηματική παροχή αξίας σε κάθε πελάτη, είναι ο μόνος τρόπος για να γνωρίσουμε εάν αυτό το οποίο έχει υλοποιηθεί είναι σωστό.

Συμπερασματικά λοιπόν, όταν κάτι είναι στην κυκλοφορία άμεσα, δημιουργεί αξία στους πελάτες, και προφανώς συγκεντρώνονται περισσότερα σχόλια για την βελτίωση της συνολικής εφαρμογής. Για παράδειγμα η έγκαιρη κυκλοφορία της Microsoft με το Word, της έδωσε την δυνατότητα να μπορέσει να λάβει τα κατάλληλα σχόλια από την αγορά, καθώς κυκλοφόρησε νωρίς τα Microsoft 1.0 και Word 2.0. Έτσι η Microsoft κατάφερε να ακούσει τους πελάτες και να συγκεντρώσει όλα τα απαραίτητα σχόλια για την ανάπτυξη του Word 3.0 που εκθρόνισε το WordPerfect που τότε το 1992 ήταν ένας κορυφαίος επεξεργαστής κειμένου. Τότε δεν υπήρχαν πουθενά Agile μοντέλα, και όμως η Microsoft κατάφερε να εισχωρήσει άμεσα στην αγορά για να μάθει από τους χρήστες. Οι Agile μεθοδολογίες μας επιτρέπουν να φτάσουμε στην αγορά νωρίτερα από τις παραδοσιακές μεθοδολογίες που αναλύθηκαν προηγουμένως, αποφεύγοντας παράλληλα να γεμίσει η ανάπτυξη του συστήματος λογισμικού με ελλατώματα και σφάλματα.



Εικόνα 14. Μοντέλα του Software Development LifeCycle

## 3.4 Μεταβατικές Προσεγγίσεις του SDLC

Στην προηγούμενη ενότητα δώσαμε έμφαση στα διάφορα μοντέλα SDLC και τις φάσεις που έχουν χρησιμοποιηθεί στην ανάπτυξη συστημάτων του λογισμικού. Παρόλα αυτά, έχουμε φτάσει σε ένα τεχνολογικό σημείο όπου οι μεθοδολογίες του SDLC έχουν αλλάξει αρκετά με την πάροδο των ετών, λόγω της ανάγκης για πιο αποτελεσματική, ευέλικτη και προσαρμοστική προσέγγιση στην ανάπτυξη λογισμικού. Επομένως, είναι σημαντικό σε αυτή την ενότητα να διερευνήσουμε την εξέλιξη των μεθοδολογιών του Κύκλου Ζωής Ανάπτυξης Συστημάτων Λογισμικού (SDLC), ακολουθώντας την εξέλιξη από τις παραδοσιακές προσεγγίσεις στις σύγχρονες μεθοδολογίες που υπάρχουν σήμερα.

Μια βασική εξέλιξη ήταν η εμφάνιση του όρου DevOps, μιας προσέγγισης που συνδιάζει την ανάπτυξη του λογισμικού (Dev) με τις λειτουργίες της πληροφρικής (Ops) για τον εξορθολογισμό του κύκλου ζωής ανάπτυξης συστημάτων λογισμικού. Το DevOps δίνει έμφαση στην συνεργασία, την αυτοματοποίηση και τη συνεχή παράδοση για να επιταχύνει την καινοτομία και την παράδοση συστημάτων λογισμικού και υπηρεσιών.

Το Infrastructure as Code (IaC) είναι μία πρακτική DevOps που στοχεύει στο να περιγράψει και να καθορίσει την ανάπτυξη της υποδομής, όπως τα δίκτυα (networks), τις εικονικές μηχανές (virtual machines) κ.α. Το IaC συμβάλλει σημαντικά στην επιβολή της συνέπειας, στην αποφυγή της μη αυτόματης διαμόρφωσης και στην ενεργοποίησης της γρήγορης παροχής περιβάλλοντων για να γίνουν δοκιμές.

Συνεπώς, οι ομάδες DevOps μπορούν να συνεργαστούν με ένα ενοποιημένο σύνολο πρακτικών και εργαλείων για την παροχή εφαρμογών και υποδομών, γρήγορα και αξιόπιστα σε μεγάλης κλίμακας έργα. Αυτή η ενότητα θα εισάγει συνοπτικά αυτές τις έννοιες, για το πως η υιοθέτηση των DevOps και IaC έχει αλλάξει τον τρόπο με τον οποίο αναπτύσσονται και παραδίδονται τα συστήματα λογισμικού, επιτρέποντας στους οργανισμούς να επιταχύνουν το SDLC και να προσαρμοστούν στις συνεχώς μεταβαλλόμενες απαιτήσεις της αγοράς.

### 3.4.1 Παραδοσιακή χρήση πόρων στο SDLC

Αρχικά, στα προηγούμενα χρόνια οι προγραμματιστές βασίζονταν σε μεγάλο βαθμό σε φυσικούς διακομιστές (servers), εικονικές μηχανές (VMs), και άλλους πόρους για την ανάπτυξη των συστημάτων λογισμικού ώστε να μπορέσουν να φέρουν εις πέρας ένα έργο μεγάλης κλίμακας. Αυτή η προσέγγιση περιελάμβανε τη παροχή και τη διαμόρφωση στοιχείων υλικού και λογισμικού για την κάλυψη των απαιτήσεων των εφαρμογών, όχι με αυτόματο τρόπο.

Ωστόσο, αυτό το παραδοσιακό μοντέλο χρήσης των πόρων που υπήρχαν διαθέσιμοι σε φυσικό επίπεδο, δημιουργούσε όσο άλλαζαν οι απαιτήσεις αρκετούς περιορισμούς και σημαντικές προκλήσεις που εμπόδισαν την αποτελεσματικότητα και επεκτασιμότητα των διαδικασιών ανάπτυξης των συστημάτων λογισμικού. Οι φυσικοί διακομιστές (physical servers) και τα VMs είχαν και έχουν περιορισμένη υπολογιστική ισχύ, αποθήκευση και μνήμη κάτι που συχνά οδηγούσε σε σοβαρή συμφόρηση απόδοσης και αδυναμία των πόρων που ήταν διαθέσιμοι να ανταπεξέλθουν με αποτελεσματικό τρόπο στην αποστολή που είχε το έργο.

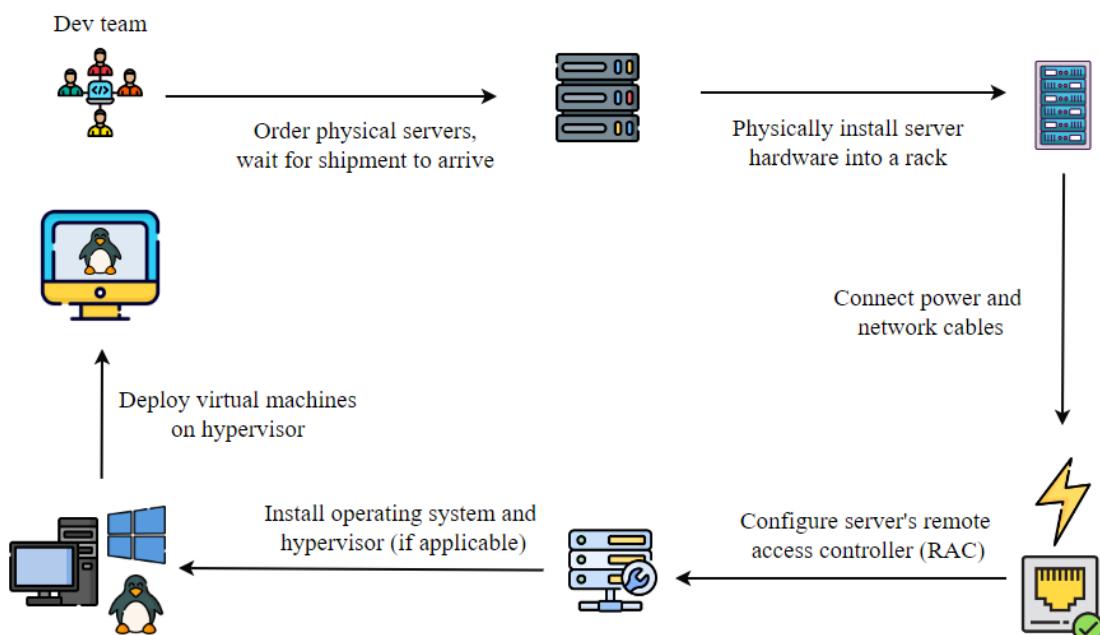
Έτσι οι προγραμματιστές έπρεπε να σχεδιάσουν προσεκτικά και να διαθέσουν πόρους για να διασφαλίσουν ότι τα συστήματα λογισμικού θα μπορούσαν να λειτουργήσουν ομαλά, που όμως ήταν ένα γεγονός που στην διάρκεια ανάπτυξης ακόμα και τώρα προσθέτει πολυπλοκότητα στην διαδικασία ανάπτυξης των συστημάτων λογισμικού. Ένας άλλος περιορισμός της παραδοσιακής χρήσης των πόρων ήταν μη αυτόματη διαμόρφωση των πόρων. Η εγκατάσταση και η συντήρηση των physical servers και VMs απαιτεί σημα-

ντικό χρόνο και προσπάθεια, καθώς και εξιδικευμένες τεχνικές γνώσεις. Αυτή η μη αυτόματη διαδικασία είναι επιρρεπής σε σφάλματα και ανθρώπινα λάθη, πράγμα που μπορεί να οδηγήσει σε πιθανά ζητήματα συμβατότητας και αποτυχίας στην ανάπτυξη.

Επιπροσθέτως, το παραδοσιακό μοντέλο χρήσης των resources δεν είχε άμεση επεκτασιμότητα. Η προσθήκη ή η αφαίρεση πόρων για την αντιμετώπιση των μεταβαλλόμενων απαιτήσεων του έργου ή των απαιτήσεων των χρηστών που χρησιμοποιούν το εκάστοτε λογισμικό, ήταν μία χρονοβόρα διαδικασία. Συνεπώς, αυτή η έλλειψη ευελιξίας σε διάφορες συνθήκες, δυσκόλευε αρκετά τους οργανισμούς και ακόμα και σήμερα αρκετές εταιρίες δυσκολεύονται να ανταποκριθούν γρήγορα στις αλλαγές της αγοράς.

Για τον λόγο αυτού του μεταβαλλόμενου τοπίου στην ανάπτυξη λογισμικού, ώστε να μπορέσουμε να αντιμετωπίσουμε τις προκλήσεις των συνθηκών σήμερα, αλλά και των περιορισμών που αναλύσαμε παραπάνω, έχουν εξελιχθεί σύγχρονες πρακτικές ανάπτυξης συστημάτων λογισμικού, δίνοντας έμφαση στη χρήση των πόρων (resources) που βασίζονται σε cloud computing και μπορούν άμεσα οι cloud providers (πάροχοι cloud) όπως είναι οι AWS, Microsoft Azure, Google Cloud και Linode να προσφέρουν πόρους και αυτοματοποιημένες διαδικασίες για την ανάπτυξη των συστημάτων του λογισμικού (software).

Επομένως, η επόμενη υποενότητα θα ξεκινήσει να δώσει μια ανάλυση για το πώς η υιοθέτηση του cloud computing (υπολογιστικού νέφους) που αναλύσαμε στο Κεφάλαιο 2, καθώς και του Infrastructure as Code (IaC) με τεχνικές DevOps που έχει μεταμορφώσει τον τρόπο με τον οποίο οι προγραμματιστές χρησιμοποιούν resources για την ανάπτυξη λογισμικού.



Εικόνα 15. Παραδοσιακή χρήση resources

### 3.4.2 Σύγχρονη χρήση πόρων στο SDLC

Συνεχίζοντας, είναι σημαντικό να αναφερθεί πως με την χρήση του cloud computing και των resources κατ'απαίτηση έχει αλλάξει σε μεγάλο βαθμό τον τρόπο με τον οποίο οι προγραμματιστές χρησιμοποιούν τους πόρους. Όπως είπαμε και στην προηγούμενη υποενότητα, οι ομάδες ανάπτυξης των συστημάτων λογισμικού βασίζονταν αρκτετά σε physical servers, VMs και γενικά τα resources που ήταν διαθέσιμα. Πλέον το τοπίο ανάπτυξης των συστημάτων λογισμικού έχει υποστεί σημαντικό μετασχηματισμό λόγω της ταχείας υιοθέτησης του cloud computing, των πρακτικών DevOps και της έννοιας του Infrastructure as Code (IaC).

Το cloud computing έχει φέρει την επανάσταση και προσφέρει πολλά πλεονεκτήματα τα οποία έχουμε αναλύσει και στο Κεφάλαιο 2, όπως είναι η επεκτασιμότητα, η ευελιξία, η σχέση κόστους-αποτελεσματικότητας και η προσβασιμότητα. Αξιοποιώντας λοιπόν τους πόρους του cloud, οι εταιρίες μπορούν να προσαρμοστούν στις αυξανόμενες απαιτήσεις και να παραμείνουν ανταγωνιστικοί στο σημερινό ψηφιακό τοπίο με γρήγορο ρυθμό.

Παράλληλα με το υπολογιστικό νέφος, το DevOps δίνει έμφαση στη συνεργασία μεταξύ των ομάδων που διαχειρίζονται ένα έργο, την αυτοματοποίηση και τη συνεχή παράδοση που έχει ως αποτέλεσμα την άμεση παράδοση του λογισμικού και των υπηρεσιών του. Επίσης, είναι σημαντικό να αναφερθεί πως το IaC είναι μία βασική πρακτική DevOps, καθώς μέσα από αυτή τη πρακτική οι ομάδες DevOps μπορούν να συνεργαστούν με εργαλεία και ένα σύνολο πρακτικών ώστε να αναπτύξουν τα συστήματα λογισμικού γρήγορα και αξιόπιστα.

Συνεπώς καταλαβαίνουμε πως η υιοθέτηση των DevOps και IaC έχει αλλάξει τον τρόπο ανάπτυξης του λογισμικού, επιτρέποντας να χρησιμοποιούνται νέες μεθόδους στο κύκλο ζωής του λογισμικού. Στο κεφάλαιο 4 θα εμβαθύνουμε αρκετά στην ανάλυση των εργαλείων IaC, όπως είναι το Terraform, Ansible, Pulumi, της έννοιας του DevOps καθώς και των CI/CD αγωγών.



## Κεφάλαιο 4

# Προσεγγίσεις Infrastructure as Code

Στα προηγούμενα κεφάλαια 2 και 3 αναλύσαμε και διερευνήσαμε τόσο το έννοια του cloud computing, όσο και την έννοια του κύκλου ζωής συστημάτων λογισμικού με τις φάσεις που έχει καθώς και τα μοντέλα που απαρτίζουν το SDLC, καθώς επίσης και την εξέλιξη του. Επίσης, τονίσαμε τη σημασία της προσαρμογής που χρειάζεται και έχει ανάγκη η αγορά στις μεταβαλόμμενες απαιτήσεις για την ανάπτυξη του λογισμικού. Αυτό το κεφάλαιο, έχει στόχο στο να εμβαθύνει στην έννοια της Υποδομής ως Κώδικα (IaC), μιας βασικής πρακτικής στην μεθοδολογία του DevOps. Παράλληλα, αυτό το κεφάλαιο θα διερευνήσει τα οφέλη, τις προκλήσεις και τα εργαλεία που έχει το IaC, καθώς και πως ενσωματώνονται με πρακτικές DevOps και CI/CD pipelines.

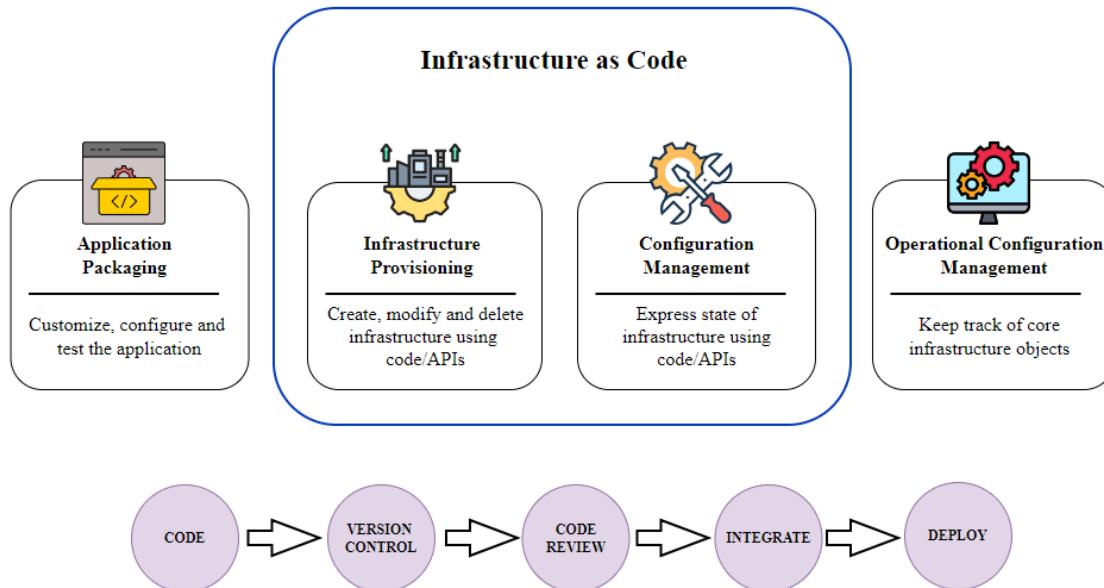
### 4.1 Εισαγωγή στο Infrastructure as Code

Η υποδομή ως κώδικα (Infrastructure as Code) έφερε την επανάσταση στον τρόπο όπου οι εταιρείες σχεδιάζουν και κατασκευάζουν υποδομές πληροφορικής, παρέχοντας ταυτόχρονα ένα αξιόπιστο και ισχυρό τρόπο. Επιτρέπει στις ομάδες του DevOps να δημιουργούν πόρους υποδομής όπως για παράδειγμα εικονικές μηχανές και δίκτυα, χρησιμοποιώντας περιγραφικά μοντέλα και γλώσσες προγραμματισμού. Για να δώσουμε μια οπτική πάνω σε αυτό, παλαιότερα για να γίνει η ρύθμιση σε ένα μεγάλο αριθμό φυσικών διακομιστών (physical servers) διαρκούσε αρκετές ώρες ή και ημέρες. Τώρα με το σωστό εργαλείο IaC, μπορούμε να έχουμε αυτούς τους servers πλήρως διαμορφωμένους σε πολύ λίγο χρονικό διάστημα.

#### 4.1.1 Ορισμός του Infrastructure as Code

Το IaC είναι ένα σύνολο εφαρμογών που χρησιμοποιεί "Κώδικα" αντί για εντολές που εισάγονται χειροκίνητα για την ρύθμιση εικονικών μηχανών, δικτύων, και πακέτων λογισμικού και για την διαμόρφωση των περιβαλλόντων που απαιτούν οι εφαρμογές. Σημαντικό να αναφερθεί πως στις τεχνολογίες IaC, η παροχή υποδομής και η εγκατάσταση συστημάτων λογισμικού διαχειρίζονται με αυτοματισμούς χρησιμοποιώντας κώδικα.

Στους παρόχους υπηρεσιών cloud, η ικανονοποίηση των απαιτήσεων των χρηστών σε διαφορετικές χρονικές στιγμές χρησιμοποιώντας παραδοσιακές μεθόδους μπορεί να είναι χρονοβόρα και επιρρεπής σε σφάλματα. Όμως, με το IaC η κοινή χρήση και η αποτελεσματική διανομή πόρων μπορεί να υλοποιηθεί σε λιγοστό χρόνο. Συνεπώς, η κοινή χρήση πόρων, η διαμόρφωση συσκευών, η προετοιμασία περιβαλλόντων ανάπτυξης και δοκιμών των εφαρμογών, η ανάπτυξη τους μπορούν να επιτευχθούν με τα εργαλεία που έχει το IaC, όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 16. Δομή του Infrastructure as Code

#### 4.1.2 Βασικές προσεγγίσεις του IaC

Σε αυτή την υποενότητα είναι σημαντικό να αναφερθεί, καθώς έχει δοθεί ο ορισμός του τι είναι το infrastructure as Code, ποιές είναι οι δύο βασικές προσεγγίσεις που υπάρχουν και ανάλογα τις απαιτήσεις του έργου ή και τις δυνατότητες που χρειάζεται να έχει ένα έργο σε εξέλιξη, καλείται η εκάστοτε ομάδα να επιλέξει. Καθώς το cloud computing, το Internet of Things και το Big Data, γίνονται οι νέες τεχνολογικές τάσεις στον κόσμο της πληροφορικής, ο λόγος για το ότι η υποδομή (infrastructure) είναι αμετάβλητη αυξάνεται. Αναφορικά λοιπόν θα δούμε τι μπορεί να είναι μια αμετάβλητη και τι μια μεταβλητή υποδομή:

- **Mutable Infrastructure as Code [Μεταβλητή Υποδομή ως κώδικα] :**

Με αυτή την έννοια, μεταβλητή υποδομή, σημαίνει να μπορεί να δέχεται αλλαγές και να μεταλλάσσεται εύκολα ανάλογα τις συνθήκες και τις απαιτήσεις. Συνεπώς, οι ομάδες πληροφορικής θα πρέπει να αντιμετωπίζουν την κάθε ανάγκη ενός server. Αυτή η διαδικασία, αν πάρουμε ως παράδειγμα ότι μιλάμε για μεγάλης κλίμακας υποδομής, μπορεί να είναι μια χρονοβόρα διαδικασία, καθώς ο στόχος είναι στο να εντοπίζονται και να επιλύονται προβλήματα, αντί στο να στοχεύουν στην καινοτομία για τη δημιουργία ενός συστήματος χωρίς περιορισμούς [5].

Η Μεταβλητή Υποδομή ως Κώδικα, δημιουργεί ένα σύστημα που τα προβλήματα σε διακομιστές επιλύονται από συγκεκριμένο σύνολο εξειδικευμένων επαγγελματιών. Συνεπώς, η διαδικασία επίλυσης μπορεί να καθυστερήσει εάν η ομάδα δεν είναι διαθέσιμη κάθε στιγμή. Παράλληλα θέτει κινδύνους παραμόρφωσης, καθώς κάθε φορά που γίνονται αλλαγές στην διαμόρφωση υπηρεσιών στις εφαρμογές, αυτό μπορεί να επιφρεάσει αρκετά τις επιδόσεις του έργου μακροπρόθεσμα. Κλείνοντας, η μεταβλητή υποδομή δημιουργεί ένα περιβάλλον ικανό να εντοπίσει ζητήματα, να τα επιλύσει γρήγορα και ταιριάζει αρκετά όταν χρειάζονται να καλυθφούν συγκεκριμένες απαιτήσεις.

- **Immutable Infrastructure as Code [Αμετάβλητη Υποδομή ως κώδικα]** :

Από την άλλη μεριά, έχουμε την αμετάβλητη υποδομή η οποία δεν μπορεί να αλλάξει και έτσι γίνεται πιο ανθεκτική. Το Immutable IaC δεν αντιμετωπίζει μεμονομένα ζητήματα, άλλα όταν χρειαστεί κάποια αλλαγή, ολόκληρη η υποδομή επανεξοπλίζεται για να καλύψει τις απαιτήσεις που έχουν εμφανιστεί [5]. Παρατηρείται πως είναι εξαιρετικά αποτελεσματική τακτική καθώς επιτυγχάνει ομοιομορφία και συνέπεια στο IaC.

Επιπροσθέτως, το Immutable IaC είναι το αποτέλεσμα του virtualization, καθώς το μεγαλύτερο μέρος βρίσκεται και υποστηρίζεται πάνω στο cloud computing. Έτσι, οι επιχειρήσεις δεν χρειάζεται να ανησυχούν για τις μεταβαλλόμενες απαιτήσεις και την παροχή νέας υποδομής όταν αυτό κριθεί αναγκαίο, γιατί τόσο τα συστήματα λογισμικού που αναπτύσσονται, όσο και το υλικό (hardware) διατηρούνται στο cloud.

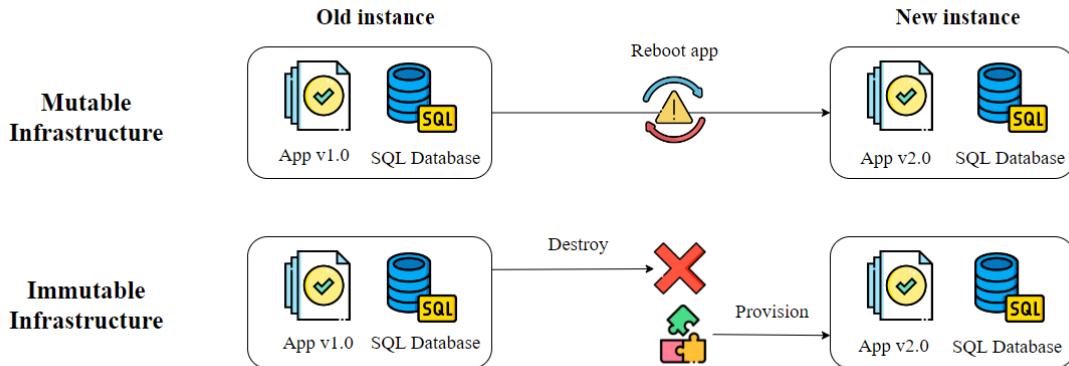
Παράλληλα, καθώς υπάρχει κατανόηση και συλλογή των απαιτήσεων υποδομής αλλά και των βημάτων που απαιτούνται για την ολοκλήρωση τους, δημιουργούνται διάφορα σενάρια κώδικα τα οποία είναι ικανά να συναρμολογούν τα απαραίτητα στοιχεία και έτσι μπορεί να αυτοματοποιηθεί ολόκληρη η διαδικασία.

#### 4.1.3 Σύγκριση Mutable IaC και Immutable IaC σε σχέση με το SDLC

Έχοντας δώσει την ανάλυση για τις δύο προσεγγίσεις IaC που υπάρχουν, είναι σημαντικό να υπάρξει σύγκριση σε συνδιασμό με τον Κύκλο Ζωής Ανάπτυξης συστημάτων λογισμικού (SDLC) καθώς η επιλογή εξαρτάται κυρίως από τις απαιτήσεις και τους περιορισμούς που μπορεί να έχει το SDLC.

Αρχικά, όσον αφορά το Immutable IaC, σύμφωνα και με αυτά που αναλύσαμε στην προηγούμενη υποενότητα, είναι επωφελές για την διασφάλιση της συνέπειας και της προβλεψιμότητας καθώς διασφαλίζει τον έλεγχο της έκδοσης, και απλοποιεί αρκετά την αντιμετώπιση των προβλημάτων και της συντήρησης. Σε αυτό το είδος υποδομής, εφόσον παρέχεται, δεν μπορεί να τροποποιηθεί. Έαν για παράδειγμα χρειαστεί οποιαδήποτε αλλαγή, τότε το υπάρχον σύστημα καταστρέφεται και ένα νέο σύστημα δημιουργείται. Αυτή η υποδομή τείνει να είναι πιο **αξιόπιστη**.

Από την άλλη πλευρά, το Mutable IaC, φαίνεται να είναι πιο κατάλληλο για σενάρια όπου απαιτείται από το έργο ευελιξία και σταδιακές αλλαγές ώστε να φτάσουμε στο επιθυμητό αποτέλεσμα, καθώς είναι ικανό να επιλύσει προβλήματα και να τροποποιηθεί μετά την παροχή άμεσα στο σύστημα που έχει δημιουργηθεί για την ανάπτυξη συστημάτων λογισμικού. Επιτρέπει στις ομάδες να επιλύουν ζητήματα ασφάλειας και να ενημερώνουν τις απαιτήσεις εφαρμογών.



Εικόνα 17. Mutable-Immutable IaC

#### 4.1.4 Βασικές προσεγγίσεις προγραμματισμού του IaC

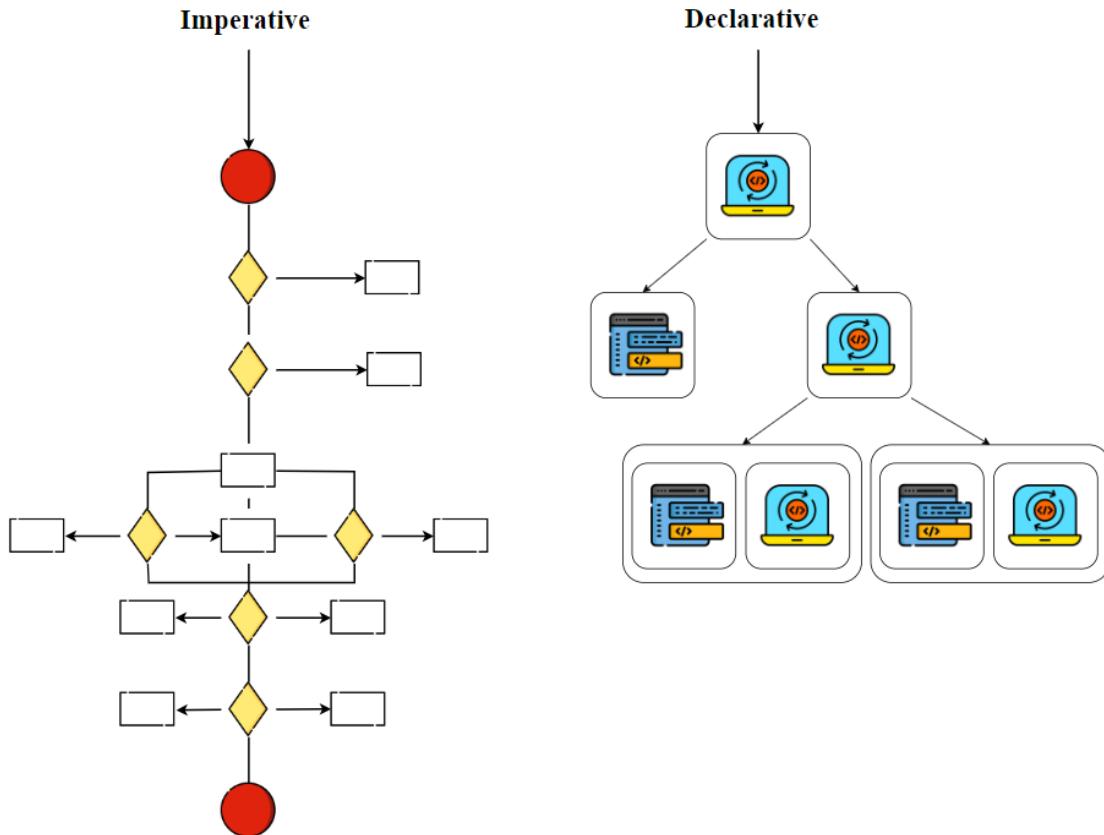
Σε αυτή την υποενότητα είναι σημαντικό να αναφερθεί, καθώς έχει δοθεί ο ορισμός του τι είναι το Infrastructure as Code, ποιές είναι οι δύο βασικές προσεγγίσεις προγραμματισμού που υπάρχουν, και ανάλογα τις απαιτήσεις του έργου ή και τις δυνατότητες που χρειάζεται να έχει ένα έργο σε εξέλιξη, να καλείται η εκάστοτε ομάδα να επιλέξει. Αναφορικά λοιπόν έχουμε:

- **Declarative Approach (functional) [Δηλωτική προσέγγιση] :**

Με αυτή την προσέγγιση, οι ομάδες ανάπτυξης έχουν την δυνατότητα να ορίσουν την απαιτούμενη κατάσταση του συστήματος και να συμπεριλάβουν τα χαρακτηριστικά και τους πόρους που θα πρέπει να διαθέτουν. Στην συνέχεια, ένα εργαλείο IaC θα φροντίσει να εφαρμόσει αυτό το σύστημα που θα έχει αρχικοποιηθεί ώστε να επιτύχει από μόνο του τα επιθυμητά αποτελέσματα. Πιο διαδεδομένα εργαλεία IaC που συνηθίζουν οι προγραμματιστές να χρησιμοποιούν είναι το Terraform, το AWS Cloud Formation, το Ansible και το Puppet [6].

- **Imperative Approach (procedural) [Επιτακτική προσέγγιση] :**

Αυτή η προσέγγιση, απαιτεί από τις ομάδες ανάπτυξης των συστημάτων λογισμικού να υπάρξει καταγραφή των βημάτων που πρέπει να ακολουθήσει ένα εργαλείο IaC κατά την παροχή ενός πόρου. Με αυτόν τον τρόπο αυτές οι σειρές εντολών καθοδηγούν το εργαλείο να δημιουργήσει κάθε περιβάλλον [6]. Αξίζει να σημειωθεί πως ένα σημαντικό εργαλείο σε αυτή την προσέγγιση είναι το Chef. Ωστόσο, παρατηρείται πως τα εργαλεία IaC είναι συμβατά και με την Δηλωτική και με την Επιτακτική προσέγγιση.



Εικόνα 18. Προσεγγίσεις προγραμματισμού του Infrastructure as Code

## 4.2 Βασικά Χαρακτηριστικά του IaC

Σε αυτή την υποενότητα είναι σημαντικό να αναφερθούν τα βασικά χαρακτηριστικά που έχει το IaC. Αυτή η μεθοδολογία του Infrastructure of Code όπως είπαμε και στον ορισμό του, διαχειρίζεται και παρέχει την υποδομή της πληροφορικής χρησιμοποιώντας κώδικα και όχι χειροκίνητη διαμόρφωση. Παρακάτω αναλύουμε ορισμένα βασικά χαρακτηριστικά του IaC [7]:

- **Automation [Αυτοματισμός]** : Το IaC έχει το χαρακτηριστικό να μπορεί να αυτοματοποιεί τη διαμόρφωση και την παροχή της υποδομής, μειώνοντας τον χρόνο και τα ανθρώπινα σφάλματα.
- **Repeatability [Επαναληψιμότητα]** : Όλα τα σενάρια του IaC μπορούν να χρησιμοποιηθούν επανειλημμένα, και αυτό συμβάλει στην εύκολη αναδημιουργία της ίδιας υποδομής σε πολλαπλά περιβάλλοντα.
- **Version Control [Ελεγχος έκδοσης]** : Ο κώδικας που δημιουργείται, αποθηκεύεται σε συστήματα ελέγχου έκδοσης όπως το Git, γεγονός που διευκολύνει την παρακολούθηση αλλαγών, την επαναφορά σε προηγούμενες εκδόσεις και τη συνεργασία με άλλες ομάδες μέσα στον οργανισμό.
- **Scalability [Επεκτασιμότητα]** : Επιτρέπει την κλιμάκωση των πόρων υποδομής προς τα πάνω ή προς τα κάτω με βάση τις απαιτήσεις που υπάρχουν, διασφαλίζοντας με αυτόν τον τρόπο ότι η υποδομή είναι πάντα διαθέσιμη και έχει βέλτιστη απόδοση.

- **Transparency [Διαφάνεια]** : Το IaC καθιστά την υποδομή κατανοητή, καθώς ο κώδικας ορίζει τα στοιχεία της υποδομής και τις σχέσεις που έχουν μεταξύ τους.
- **Compliance and Security [Συμμόρφωση και Ασφάλεια]** : Το IaC είναι σε θέση να διασφαλίσει ότι τα πρότυπα συμμόρφωσης, οι βέλτιστες πρακτικές και οι πολιτικές ασφαλείας εφαρμόζονται με σαφήνεια σε όλη την υποδομή, μειώνοντας αισθητά τον κίνδυνο για ευπάθειες.

## 4.3 Πλεονεκτήματα και Προκλήσεις του IaC

Σε αυτή την ενότητα, είναι εξίσου σημαντικό να αναφερθούν τα πλεονεκτήματα που έχει η προσέγγιση Infrastructure as Code (IaC), αλλά και οι προκλήσεις που δημιουργούνται, ώστε να μπορέσουμε να ενσωματώσουμε με σωστή βάση την μετάβαση μετέπειτα στα εργαλεία που έχει το IaC.

### 4.3.1 Πλεονεκτήματα του IaC

Αναλυτικά λοιπόν για τα πλεονεκτήματα έχουμε:

- **Reduced costs [Μειωμένο κόστος]** :

Ένα από τα βασικά πλεονεκτήματα του IaC είναι ότι υπάρχει μείωση του κόστους, καθώς κάθε εργασία που γινόταν, οδηγούσε τις περισσότερες φορές στην αύξηση του χρόνου μέχρι την ολοκλήρωση της εργασίας, και αυτό με το IaC έχει αλλάξει σε θετική κλίμακα, δραματικά.

Οι σημαντικές διεργασίες, όπως η χρονονοβόρα διαμόρφωση της υποδομής με το IaC αυτοματοποιούνται και οι μηχανικοί πληροφορικής μπορούν πλέον να ολοκληρώσουν αυτές τις εργασίες σε σύντομο χρονικό διάστημα και να επικεντρωθούν σε άλλες χρήσιμες διεργασίες κατά την ανάπτυξη συστημάτων λογισμικού.

Με αυτόν τον τρόπο, ένας οργανισμός είναι σε θέση να πετύχει την ελασχιστοποίηση του κόστους από διάφορες διαδικασίες και να τις διαθέσει εξ ολοκλήρου είτε σε εξοικονόμηση χρημάτων για κρίσιμες αποφάσεις στο μέλλον είτε για αναβάθμιση και εξέλιξη όλων των λειτουργιών μέσα σε έναν οργανισμό [8].

- **Better Disaster Recovery [Καλύτερη Ανάκτηση Καταστροφών]** :

Μια επιπλέον πρόκληση είναι ότι το IaC είναι σε θέση να βοηθήσει στην αποκατάσταση από καταστροφές, μπορεί επίσης να δημιουργήσει και σημαντικές προκλήσεις [8]. Σαν παράδειγμα μπορούμε να αναφέρουμε την διασφάλιση ότι για τις διαμορφώσεις υποδομής πρέπει να έχουν δημιουργηθεί κατάλληλα αντίγραφα ασφαλείας, τα οποία μπορούν να αποκατασταθούν γρήγορα σε περίπτωση καταστροφής.

- **Less human errors [Λιγότερα ανθρώπινα λάθη]** :

Ένα σημαντικό όφελος που παρατηρείται έιναι πως με το IaC, μειώνεται σε μεγάλο βαθμό το ανθρώπινο λάθος. Πιο συγκεκριμένα, υπάρχουν δύο κατηγορίες ανθρώπινων λαθών: το πρώτο είναι όταν ένας άνθρωπος δηλώνει λάθη που έγιναν από τις ομάδες προγραμματιστών σε ένα έργο κατά την διάρκεια κατασκευής υποδομής ή στις φάσεις του SDLC [8]. Σαν δεύτερη κατηγορία λαθών είναι η προσαρμογή νέων υπαλλήλων στην ανάπτυξη των συστημάτων λογισμικού και στην υποδομή που είχε κατασκευάσει κάποιος άλλος μηχανικός πληροφορικής.

- **Better documentation [Καλύτερη τεκμηρίωση] :**

Το IaC είναι σε θέση να δημιουργήσει λεπτομερείς αναφορές και τεκμηρίωση για κάθε διαδικασία. Για παράδειγμα, όταν οι εργαζόμενοι αποχωρούν από μια εταιρεία ή μια ομάδα απόμων ρυθμίζουν το μοντέλο κώδικα για άλλες ομάδες, και το IaC καταγράφει κάθε αλλαγή και ενέργεια που έχει γίνει [8].

- **Increase scalability [Αύξηση επεκτασιμότητας] :**

Είναι σημαντικό να αντιληφθούμε πως χρησιμοποιώντας το IaC, οι ομάδες μπορούν να διαμορφώσουν τα περιβάλλοντα τους σύμφωνα με τις προδιαγραφές τους με γρήγορο ρυθμό και σε μεγάλη κλίμακα [8]. Με την συνέπεια που αναλύθηκε πιο πάνω, οι ομάδες προγραμματιστών και πληροφορικής μπορούν να αναπτύξουν τη διαμόφωση και να δημιουργήσουν αυτά τα περιβάλλοντα πολλές φορές, καθιστώντας το σταθερό και αποτρέποντας την ασυμβατότητα που προκαλείται από μετατόπιση διαμόρφωσης.

- **Increased speed and efficiency [Αυξημένη ταχύτητα και αποτελεσματικότητα] :**

Είναι εύκολο να παρατηρηθεί πως πολλές διεργασίες που υπάρχουν και πρέπει να γίνουν, όπως η παρακολούθηση και η διαχείριση υποδομών, με το IaC η κάθε εργασία μπορεί να γίνει δύο φορές πιο γρήγορή καθώς το Infrastructure as Code έχει φροντίσει στο να αυτοματοποιήσει σχεδόν κάθε διαδικασία [8]. Ο αυτοματισμός επεκτείνεται από σημαντικές διαδικασίες όπως η εικονοποίηση έως την διαχείριση λογαριασμών χρήστη, τις βάσεις δεδομένων, τη διαχείριση δικτύου και ακόμη και μικρές λειτουργίες όπως η προσθήκη environments και resources όταν χρειάζεται.

Συνεπώς, αυτή η αυτοματοποίηση δίνει την δυνατότητα για περισσότερη ταχύτητα και απλούστερη διαδικασία στις ομάδες προγραμματιστών καθώς ο απλός κώδικας βελτιώνει τη σαφήνεια, την αποτελεσματικότητα και την ταχύτητα σε ένα έργο.

Το IaC έρχεται να εξαλείψει σημαντικά αυτά τα δύο ζητήματα, καθώς με την αυτοματοποίηση μειώνει του κινδύνους ανθρωπογενών λαθών περιορίζοντας τις μακροχρόνιες διαδικασίες. Παράλληλα, το IaC τυποποιήσει και κατέγραψε όλες τις διαδικασίες κατά το στάδιο κατασκευής [8]. Αυτό δημιουργεί λεπτομερείς αναφορές και τεκμηρίωση για πως λειτουργεί μία υποδομή, πως διαχειρίζεται, αναπτύσσεται και αυτή αλλά και τα συστήματα λογισμικού κατά την διάρκεια του κύκλου ζωής λογισμικού, και έτσι οι νέοι υπάλληλοι είναι σε θέση να μπορούν να διαχειριστούν και να συνεχίσουν το έργο χωρίς προβλήματα.

- **Improved consistency [Βελτιωμένη συνέπεια] :**

Καθώς το IaC είναι σε θέση να απομονώσει τα ανθρώπινα λάθη, παράλληλα βοηθάει και στη συνοχή των όλων συστημάτων που αναπτύσσονται κατά την διάρκεια του έργου [9]. Το IaC βοηθάει στο να υπάρχει συνοχή στις διαδικασίες, βελτιώνοντας την, αποτρέποντας την σπατάλη πολύτιμων πόρων, και τις καθυστερήσεις που μπορεί να προκαλέσουν οι ασυνέπειες στη διαμόρφωση.

- **Eliminate configuration drift [Κατάργηση της μετατόπισης της διαμόρφωσης] :**

Ένα άλλο σημαντικό πλεονέκτημα του Infrastructure as Code είναι ότι είναι ”ανίκανη”. Πιο συγκεκριμένα, κάποιος μπορεί να αναπτύξει πολλές φορές τον κώδικα, όμως η πρώτη ανάπτυξη να είναι η επικρατέστερη και οι επόμενες αναπτύξεις να μην έχουν στην ουσία κανένα αποτέλεσμα [9].

Συνεπώς, το πιο εντυπωσιακό είναι πως το IaC αποτρέπει τη μετατόπιση της διαμόρφωσης. Για παράδειγμα, εάν κάποιος αλλάξει ένα resource πρέπει να γίνει επαναφορά το συντομότερο δυνατόν ώστε να μην δημιουργηθούν σφάλματα. Με το IaC οι προδιαγραφές της διαμόρφωσης του περιβάλλοντος και κατ'επέκταση και στην ανάπτυξη των συστημάτων λογισμικού, βρίσκονται ήδη μέσα στον κώδικα που έχει δημιουργηθεί. Αυτό σημαίνει ότι η διόρθωση γίνεται αυτόματα εάν εφαρμοστεί η αλλαγή.

- **Improved security strategies [Βελτιωμένες στρατηγικές ασφάλειας] :**

Μεταξύ άλλων πλεονεκτημάτων του IaC, είναι ότι μπορεί να βοηθήσει στη βελτίωση των στρατηγικών ασφάλειας [9]. Με το IaC, οι υπηρεσίες υπολογιστών, αποθήκευσης, και δικτύωσης παρέχονται με κώδικα και αναπτύσσονται με τον ίδιο τρόπο συχνά χρησιμοποιώντας ιδιωτικό ή δημόσιο cloud [**private or public cloud**].

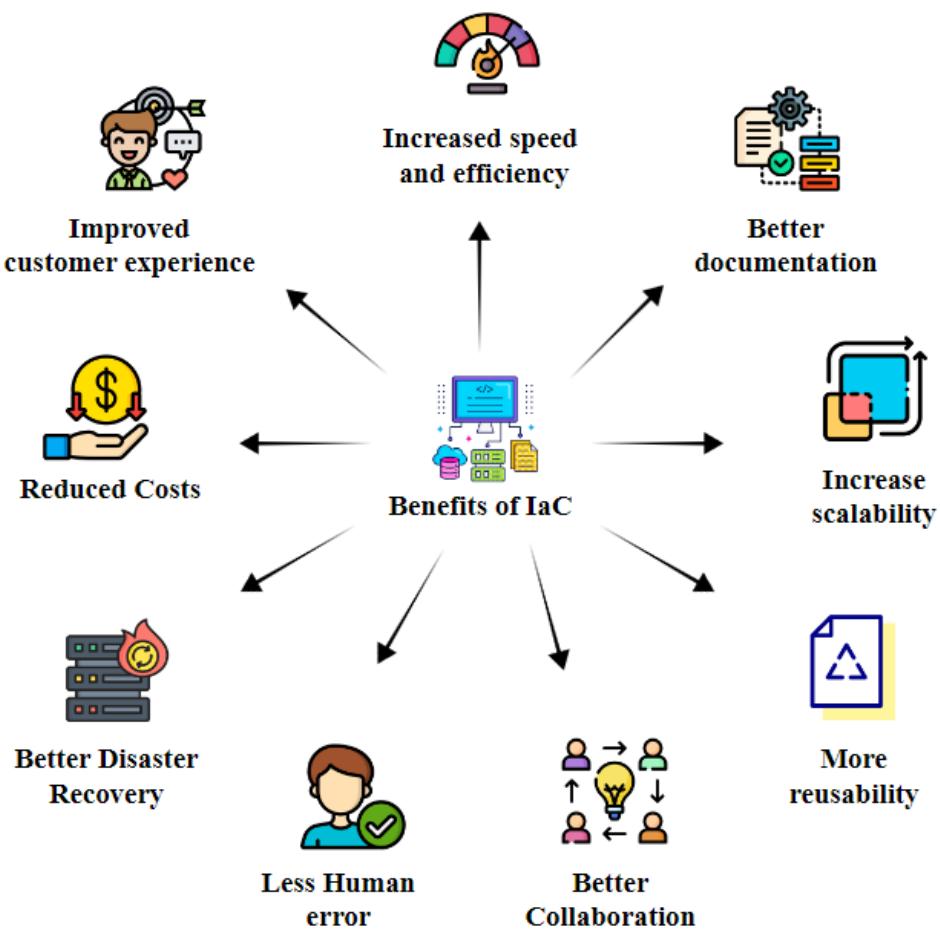
Αυτό ισχύει και για τα πρότυπα ασφάλειας στα σενάρια που θα έχουν συλλεχθεί κατά την διάρκεια των φάσεων για την ανάπτυξη των συστημάτων λογισμικού, και μπορούν να δημιουργηθούν εύκολα. Συνεπώς, αυτό θα αποδεικνύταν πιο ωφέλιμο για υποδομές που απαιτούν αυστηρή ασφάλεια με αποτέλεσμα πολλά περάσματα ασφάλειας.

- **Accountability [Υπευθυνότητα] :**

Αυτό το πλεονέκτημα της Υποδομής ως Κώδικα είναι αποτέλεσμα διαφόρων χαρακτηριστικών. Αρχικά υπάρχει η δυνατότητα της τεκμηρίωσης για τις ενέργειες που έγιναν κατά την διαμόρφωση της υποδομής και υπάρχει η πρόσβαση στον κώδικα. Έτσι αυτή η δυνατότητα καθιστά τον κώδικα εύκολα προσβάσιμο και αρκετά διαφανή ώστε να εξηγεί γιατί οι χρήστες έκαναν συγκεκριμένες αλλαγές όταν πραγματοποιήθηκαν και το αποτέλεσμα αυτών των αλλαγών [9].

Έτσι το IaC, βελτιώνει την λογοδοσία και την ορατότητα των αυτοματοποιημένων μεθόδων διαμόρφωσης και ανάπτυξης στα άλλα μέλη της ομάδας του έργου, γεγονός που με τη σειρά του βελτιώνει και την παραγωγικότητα της ομάδας.

Συμπερασματικά, η συνεργασία με τα εργαλεία και τις βέλτιστες πρακτικές του IaC, δίνει την δυνατότητα τόσο στις ομάδες των μηχανικών πληροφορικής, όσο και στις επιχειρίσεις να δημιουργήσουν τέτοια περιβάλλοντα, σταθερά, με τα χαρακτηριστικά που έχει το IaC.



Εικόνα 19. Πλεονεκτήματα του Infrastructure as Code

### 4.3.2 Προκλήσεις στο πλαίσιο του IaC

Σε αυτό το σημείο θα χρειαστεί να διερευνήσουμε και την άλλη όψη του νομίσματος όσον αφορά το Infrastructure as Code, θέτοντας ζητήματα που δημιουργούν προκλήσεις σε οργανισμούς και ομάδες που υιοθετούν την προσέγγιση του IaC και θα πρέπει να ληφθεί υπόψη κατά την εφαρμογή της. Επιγραμματικά λοιπόν έχουμε :

- **Coding Language Dependency [Εξάρτηση γλώσσας κωδικοποίησης] :**

Μια πρόκληση του IaC στην υιοθέτηση του είναι ότι χρειάζεται ισχυρή κατανόηση γλωσσών προγραμματισμού όπως είναι η JSON, η HashiCorp Configuration Language (HCL) που είναι γλώσσα του εργαλείου Terraform, η YAML, η Ruby κλπ. Έτσι, είναι εμφανές πως δημιουργείται μια πρόκληση για τις ομάδες που δεν έχουν σημαντική εξοικίωση με τον προγραμματισμό [10].

- **Security Assessment Processes [Διαδικασίες αξιολόγησης ασφάλειας] :**

Είναι σημαντικό να αναφερθεί πως τα εργαλεία καθώς και οι διαδικασίες ασφαλείας που προ-υπάρχουν, ενδέχεται να μην επαρκούν για το IaC. Έτσι απαιτούνται χειροκίνητοι έλεγχοι για να διασφαλιστεί ότι οι προβλεπόμενοι πόροι είναι λειτουργικοί και ότι χρησιμοποιούνται από τις σωστές εφαρμογές [10].

Συμπερασματικά λοιπόν, δημιουργείται μία πολυπλοκότητα που προκαλείται από τον νέο τύπο κώδικα, καθώς δημιουργούνται οι εξής προκλήσεις: για το ποιές είναι οι βέλτιστες πρακτικές για τον κώδικα, πώς θα πρέπει να εφαρμόσουμε τις αλλαγές στην υποδομή, αποφεύγοντας σημαντικές διακοπές λειτουργίας, καθώς και πώς θα γίνει η δοκιμή για όλες αυτές τις αλλαγές.

- **IaC Monitoring [Παρακολούθηση IaC] :**

Επίσης άλλη μία πρόκληση στην υιοθέτηση του IaC είναι και η δυσκολία που μπορεί να έχει, ειδικά σε αναπτύξεις έργου μεγάλης κλίμακας. Συνεπώς, απαιτούνται εργαλεία παρακαλούθησης ικανά ώστε να υπάρχει σωστή δομή καταγραφής για το τι πόροι παρέχονται, ποιο είναι το κόστος και πόσο συχνά γίνονται αλλαγές στα παρεχόμενα resources. Αυτή η πρόκληση περιτλέκει αρκετά το workflow στις ομάδες καθώς μπορεί να είναι δύσκολο να κατανοήσουν χωρίς την απαραίτητη γνώση.

- **Role-Based Access Control (RBAC) [Ελεγχος πρόσβασης βάση ρόλου] :**

Σε έναν οργανισμό που έχει δημιουργήσει διαδικασίες IaC για την ανάπτυξη συστημάτων λογισμικού και το κύκλο ζωής του λογισμικού (SDLC), να μπορεί να υπάρχει σωστή ρύθμιση ρόλων και αδειών σε διάφορα μέρη των ομάδων. Ωστόσο, αυτό μπορεί να θεωρηθεί απαιτητικό καθώς δημιουργεί δυσκολίες στην διαχείριση της πρόσβασης σε σενάρια IaC και στην διασφάλιση πως μόνο εξουσιοδοτημένα άτομα μπορούν να κάνουν τις αλλαγές που πρέπει για όλη την υποδομή.

- **Feature Lag [Καθυστέρηση Χαρακτηριστικών] :**

Άλλη μία πρόκληση που πρέπει να είναι καταγεγραμμένη ώστε να μπορέσει να μας βοηθήσει στις αποφάσεις συνολικά της Υποδομής ως Κώδικα και των προσεγγίσεων που έχει, είναι και ότι τα εργαλεία IaC πιθανόν να υστερούν σε σχέση με εκδόσεις χαρακτηριστικών που μπορεί να δίνουν οι cloud providers [11].

Συνεπώς, δημιουργείται μια δυσκολία στην εφαρμογή νέων δυνατοτήτων cloud, εκτός και αν οι δημιουργοί των χαρακτηριστικών αυτών, είναι σε θέση να ενημερώσουν επαρκώς τους cloud παρόχους, ώστε να μπορέσουν με την σειρά τους να ενσωματώσουν αυτές τις δυνατότητες. Αυτό βέβαια, μπορεί να οδηγήσει σε καθυστερήσεις ως προς τα χαρακτηριστικά και πρόσθετη πολυπλοκότητα.

- **Conventions and Logic [Συμβάσεις και Λογική] :**

Οι ομάδες DevOps χρειάζεται να κατανοούν πλήρως τα σενάρια του IaC και τα οποία υιοθετούνται σε ένα έργο, καθώς μπορεί να προκαλέσει για άτομα χωρίς τεχνογνωσία, δυσκολία στην ταχύτητα της κλιμάκωσης του έργου, αλλά και της ενσωμάτωσης των σεναρίων μέσα σε έργο μεγάλης κλίμακας.

- **Maintenance [Συντήρηση] :**

Επίσης σημαντικό είναι και η διατήρηση όλων των ρυθμίσεων σε μια Infrastructure as Code προσέγγιση. Παρόλα αυτά, δημιουργείται μια πρόκληση να διατηρηθούν σε μεγάλης κλίμακας έργα, και να υπάρξει η απαραίτητη παρακολούθηση των αλλαγών, αλλά και στο να διασφαλιστεί ότι οι αλλαγές στην υπόδομή είναι ενημερωμένες και ακολουθούν όλες τις διαδικασίες που έχουν καταγραφεί.

### 4.3.3 Συμπεράσματα για το IaC

Τα σενάρια του IaC μπορούν να προσφέρουν, όπως είδαμε, πολλά πλεονεκτήματα στην διαχείριση της υπόδομής ενός έργου κατά την διάρκεια του κύκλου ζωής ανάπτυξης των συστημάτων λογισμικού και τις υιοθέτησης αυτής της προσέγγισης, ωστόσο είναι σημαντικό να ληφθούν υπόψιν και όλες οι προκλήσεις που αναλύσαμε στην προηγούμενη ενότητα.

Οι μηχανικοί πληροφορικής και οι οργανισμοί θα μπορούν να προετοιμαστούν καλύτερα καθώς θα πρέπει να κατανοούν ξεκάθαρα τα πλεονεκτήματα και τις προκλήσεις του Infrastructure as Code, να έχουν κάνει την απαραίτητη ανάλυση για το κόστος (χρόνος, άνθρωποι, υποδομές, φήμη) του, πώς να μειώνουν το κόστος, και να κατανοούν τον αντίκτυπο της στρατηγικής που επιλέγουν στο έργο, αλλά και να έχουν ένα σχέδιο για το πώς να αλλάξουν τη στρατηγική εάν διάλεξαν λάθος δομή στην αρχή.

## 4.4 Καλύτερες πρακτικές IaC

Το Infrastructure as Code έχει αρχίσει να αποδεικνύει πως είναι μια κρίσιμη προσέγγιση της σύγχρονης ανάπτυξης του κύκλου ζωής των συστημάτων λογισμικού (SDLC), επιτρέποντας στις ομάδες να διαχειρίζονται και να παρέχουν infrastructure resources χρησιμοποιώντας κώδικα. Έτσι οι εταιρίες μπορούν να επιτύχουν μεγαλύτερη αποτελεσματικότητα, επεκτασιμότητα και αξιοπιστία στις λειτουργίες πληροφορικής τους. Ωστόσο είναι σημαντικό να εφαρμόσουν αποτελεσματικά το IaC και τις βέλτιστες πρακτικές του και επιγραμματικά είναι οι παρακάτω:

- **Code Everything [Κωδικοποιήστε τα πάντα] :**

Η έννοια του Infrastructure as Code θα χάσει την δυναμική της, εάν λείπει το στοιχείο του κώδικα από την συνολική υποδομή. Για αυτόν τον λόγο χρειάζεται οι ομάδες πληροφορικής να προβούν στην κωδικοποίηση όλων των προδιαγραφών υποδομής. Με αυτή την διαδικασία, θα βοηθήσει αρκετά στον καθορισμό των στοιχείων του cloud που πιθανόν να χρησιμοποιηθούν, του τρόπου με τον οποίο διαμορφώνεται το περιβάλλον του cloud και του τρόπου με τον οποίο τα στοιχεία σχετίζονται μεταξύ τους. Και με αυτόν τον τρόπο θα μπορέσει το infrastructure να υλοποιηθεί και να αναπτυχθεί ως κώδικας, γρήγορα και με συνέπεια.

- **Minimize Documentation [Ελαχιστοποίηση τεκμηρίωσης] :**

Το IaC είναι σε θέση να παρακολουθεί κάθε βήμα και διαδικασία που γίνεται και να το τεκμηριώνει με μεγάλη λεπτομέρεια. Συνεπώς είναι εύκολη η απομάκρυνση από παραδοσιακές πρακτικές, όπου το να συμπληρώνουμε κάθε ενέργεια που γίνεται με πρόσθετες οδηγίες τεκμηρίωσης που πρέπει να εκτελέσουν οι ομάδες πληροφορικής είναι μηδενικές.

- **Test, Integrate, Deploy, Repeat :**

Η διαχείριση των αλλαγών στον κώδικα υποδομής και η διατήρηση στην βέλτιστη λειτουργία, γεννάει την ανάγκη να πρέπει να επενδύσουμε σε επαναλαμβανόμενες και συνεχείς δοκιμές, ενοποίηση και ανάπτυξη. Αυτό είναι ένα σημαντικό στοιχείο, καθώς μας βοηθάει στην αποφυγή κινδύνων ασφαλείας και ζητημάτων μετά το πέρας της ανάπτυξης. Στο σύνολο της, αυτή η πρακτική είναι σε θέση να εντοπίσει εύκολα απειλές και σφάλματα νωρίς στον κύκλο ζωής ανάπτυξης των συστημάτων λογισμικού, ελαχιστοποιώντας τον κίνδυνο να συμβεί αυτό μετά τη μετάδοση του προιόντος σε live μορφή.

- **Make Infrastructure Code Modular :**

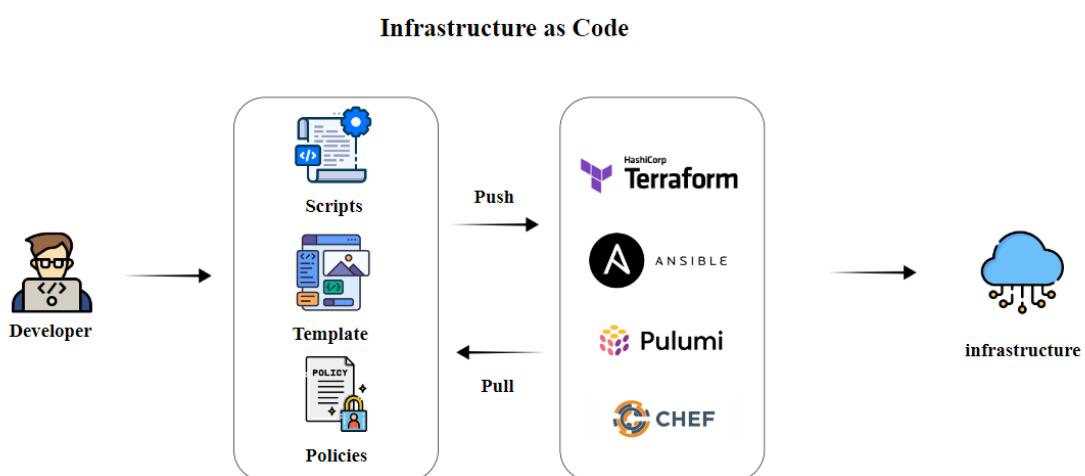
Μια δημοφιλής τάση στον κόσμο της ανάπτυξης λογισμικού είναι η αρχιτεκτονική των **Microservices**, και αυτό γιατί έχει την ικανότητα να δημιουργεί ανεξάρτητες μονάδες [12]. Έτσι, εφαρμόζοντας την έννοια αυτή στο IaC, μπορούμε εύκολα να δημιουργήσουμε στοίβες ή λειτουργικές μονάδες ως microservices, και μετά να μπορούν αυτόματα να συνδυαστούν. Μια τέτοια πρακτική προσέγγιση μας δίνει καλύτερο έλεγχο σε όποιον έχει πρόσβαση σε διαφορετικά μέρη του κώδικα υποδομής μας. Παράλληλα, η διάσπαση της υποδομής σε microservices μπορεί να διευκολύνει κατά πολύ, στον εντοπισμό και τη διόρθωση σφαλμάτων, κάνοντας έτσι τις λειτουργίες μας πιο ευέλικτες.

- **Version Control [Ελεγχος έκδοσης]** : Χρειάζεται να διατηρηθεί ο έλεγχος έκδοσης (**Git**) του κώδικα υποδομής, καθώς όλες οι λεπτομέρειες διαμόρφωσης υπάρχουν ως κώδικας. Είναι εύκολο λοιπόν να έρθει σε συμπέρασμα κανείς πως, η διατήρηση μιας διαδρομής ελέγχου για αλλαγές στον κώδικα μπορεί να μας βοηθήσει να παρακολουθούμε και να διαχειρίζομαστε τις αλλαγές σε όλη την υποδομή [12].

- **Build Immutable Infrastructure [Δημιουργία Αμετάβλητης υποδομής]** :

Μια άλλη σημαντική πρακτική είναι η ιδέα της αντικατάστασης των στοιχείων υποδομής αντί της αλλαγής τους σε μία συγκεκριμένη χρονική στιγμή, καθώς μπορεί να συμβάλει σημαντικά στην επίτευξη της συνέπειας στις λειτουργίες μας, αλλά και στην ανθεκτικότητα της υποδομής [13]. Έτσι με αυτή την πρακτική, ο κώδικας που δημιουργείται από τις ομάδες πληροφορικής για μια υποδομή μπορεί να επαναληφθεί πολλές φορές χωρίς κανέναν κίνδυνο μετατόπισης της διαμόρφωσης, όπως έχουμε αναλύσει στην 4.1.2 υποενότητα. Συνεπώς, όταν χρειάζεται, αποσύρουμε την παλιά υποδομή και την αντικαθιστούμε με την καινούργια.

Συμπερασματικά λοιπόν, οι βέλτιστες πρακτικές που αναλύσαμε θεωρούνται οι σημαντικότερες στην έννοια του Infrastructure as Code. Αυτό συμβαίνει γιατί, σε πολλά cloud περιβάλλοντα και ομάδες πληροφορικής χρησιμοποιώντας αυτές τις πρακτικές, μπορούμε να μειώσουμε τις μη αυτόματες διαδικασίες και να εξασφαλίζουμε κάθε φορά ταχύτερη παράδοση λογισμικού. Το **Infrastructure as Code** είναι ένα από τα βασικότερα στοιχεία του DevOps. Στις μέρες μας, όπου ο τεχνολογικός κόσμος μεταμορφώνεται και εξελίσσεται ραγδαία μέσω των νέων τεχνολογιών που κάνουν την εμφάνιση τους με τις απαραίτησεις που υπάρχουν, το IaC είναι ένα βήμα ώστε όλες οι λειτουργίες για την υποδομή και το SDLC να είναι έτοιμες για το μέλλον, γρήγορα. Παράλληλα το IaC βοηθάει σε μεγάλο βαθμό στο να ανακαλύψουμε όλες τις δυνατότητες που έχει το cloud computing, μία έννοια που έχει αναλυθεί στο Κεφάλαιο 2, καθώς ο συνδυασμός αυτών των εννοιών βοηθάει σημαντικά στην εξάλειψη σφαλμάτων που σχετίζονται με τη μη αυτόματη διαχείριση τεχνολογιών υποδομών και προσφέρει ευελιξία στον κύκλο ζωής της ταχείας ανάπτυξης των συστημάτων λογισμικού, με μείωση του λειτουργικού κόστους.



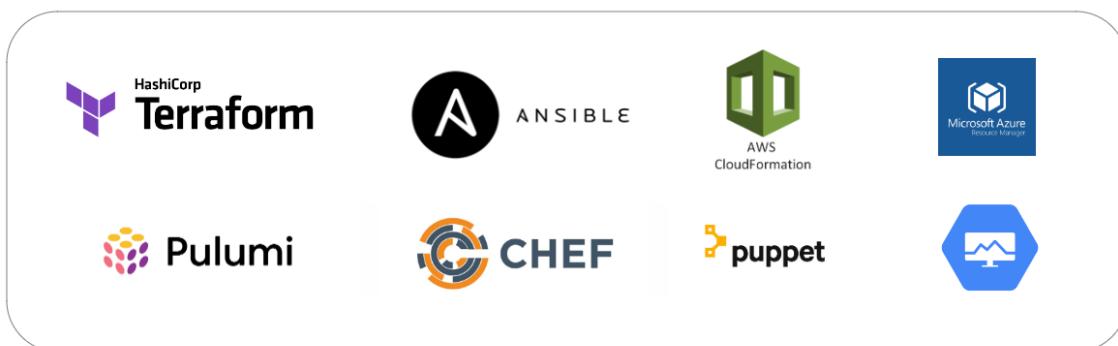
Εικόνα 20. Εφαρμογή του Infrastructure as Code

## 4.5 Εργαλεία του IaC

### 4.5.1 Εισαγωγή στα εργαλεία του IaC

Στις προηγούμενες ενότητες έχουμε διερευνήσει ενδελεχώς τις διαφορετικές προσεγγίσεις στο Infrastructure as Code, συμπεριλαμβανομένων των πλεονεκτημάτων, των προκλήσεων που μπορεί να έχει μια επιλογή υιοθέτησης του IaC για την δημιουργία υποδομής και ανάπτυξης του κύκλου ζωής των συστημάτων λογισμικού, καθώς συζητήσαμε και για τις βέλτιστες πρακτικές της εφαρμογής του. Είναι σημαντικό να έχουμε καταγεγραμένα και να κατανοήσουμε πως τα βασικά εργαλεία IaC έχουν σχεδιαστεί για να διευκολύνουν την εφαρμογή και τη διαχείριση των πρακτικών του. Κάθε εργαλείο IaC που θα αναλυθεί στις επόμενες υποενότητες, έχει μοναδικά χαρακτηριστικά και πλεονεκτήματα, καθιστώντας απαραίτητη την αξιολόγηση παραγόντων όπως η ευκολία χρήσης, η κάλυψη πόρων, η υποστήριξη της κοινότητας μέσα στον τεχνολογικό κόσμο, αλλά και το κόστος.

Συνεπώς, η ενσωμάτωση με την υπάρχουσα υποδομή και άλλες αλυσίδες εργαλείων είναι ζωτικής σημασίας για την υιοθέτηση τους, καθώς τα εργαλεία IaC μπορούν να κατηγοριοποιηθούν σε λύσεις για συγκεκριμένες πλατφόρμες και λύσεις σε προβλήματα που ήδη υπάρχουν. Σκοπός λοιπόν αυτής της ενότητας είναι να αναλύσει με σαφήνεια όλα τα σημαντικά εργαλεία **Υποδομής ως Κώδικα** που υπάρχουν, να αναλύσει την αρχιτεκτονική εφαρμογής τους, να δώσει έμφαση στα πλεονεκτήματα και μειονεκτήματα τους, να συγκρίνει αυτά τα εργαλεία, να δώσει μετέτειτα μια ανάλυση για το πώς συνδέονται η έννοια του **Infrastructure as Code** με το **DevOps** και τις πρακτικές **CI/CD pipelines**. Με αυτόν τον τρόπο θα δημιουργηθεί το συμπέρασμα που θα είναι και η συνέχεια για το Κεφάλαιο 5, όπου θα δημιουργηθεί ένα πρακτικό case study ικανό να εντάξει όσα έχουν αναλυθεί από την αρχή αυτής της εργασίας.



Εικόνα 21. Εργαλεία IaC

## 4.5.2 Terraform



### Εισαγωγή στο Terraform

Το Terraform<sup>1</sup> είναι ένα εργαλείο Infrastructure as Code, ανοιχτού κώδικα [open-source] που αναπτύχθηκε από τη HashiCorp και επιτρέπει σε ομάδες προγραμματιστών να ορίσουν και να διαχειριστούν cloud και on-premise υπολογιστικούς πόρους, χρησιμοποιώντας ένα ευανάγνωστο από τον άνθρωπο αρχείο διαμόρφωσης και ρυθμίσεων. Αυτό το εργαλείο χρησιμοποιεί μια δηλωτική σύνταξη που είναι το HashiCorp Configuration Language (HCL), που σημαίνει ότι οι χρήστες έχουν την δυνατότητα να ορίζουν την επιθυμητή κατάσταση της υποδομής και το Terraform είναι σε θέση να καθορίσει τις ενέργειες που απαιτούνται για την επίτευξη αυτής της κατάστασης.

Στην συνέχεια αυτής της υποενότητας θα αναλύσουμε ενδελεχώς κάθε στοιχείο του Terraform, τα βασικά χαρακτηριστικά που έχει, που μπορεί να υλοποιηθεί, σε ποιες πλατφόρμες, τις τυπικές ροές εργασίας Terraform, καθώς και το πώς χρησιμοποείται το εργαλείο. Παράλληλα, θα αναλυθεί η δομή του κώδικα HCL, το Cloud Development Kit (CDL) ανάπτυξης και διασύνδεσης Terraform σε πολλαπλές υψηλού επιπέδου γλώσσες προγραμματισμού, και οι εναλλακτικές λύσεις στο Terraform, ώστε μετά το πέρας της υποενότητας να υπάρχει μια συνολική εικόνα του εργαλείου Terraform.

### Βασικά χαρακτηριστικά του Terraform

Αρχικά, είναι σημαντικό σαν πρώτο βήμα αυτής της υποενότητας για το εργαλείο Terraform να αναφέρουμε και να δώσουμε μια πρώτη εικόνα για τα χαρακτηριστικά του. Πιο συγκεκριμένα έχουμε:

- Declarative Syntax [Δηλωτική Σύνταξη]:**

'Οπως αναλύσαμε και σε προηγούμενη ενότητα για την δηλωτική προσέγγιση υποδομής, αυτή την έννοια χρησιμοποιεί και το Terraform. Δίνει την δυνατότητα στους χρήστες να ορίσουν από την αρχή την κατάσταση της υποδομής και η Terraform καθορίζει όλες τις ενέργειες που απαιτούνται για την επίτευξη του σεναρίου που έχει δοθεί [14]. Με αυτό τον τρόπο διασφαλίζεται πως το Infrastructure είναι πάντα σε συνεπή και αναπαραγωγική κατάσταση.

- State Management [Διαχείριση κατάστασης] :**

Το Terraform μπορεί να διατηρεί ένα αρχείο κατάστασης (terraform state file) που υποδηλώνει σε ζωντανή στιγμή την κατάσταση που έχει η υποδομή. Αυτό το αρχείο καταγράφει σημαντικές πληροφορίες σχετικά με τα resources που παρέχει το Terraform, τις εξαρτήσεις που μπορεί να έχει μέσα στην υποδομή και τα χαρακτηριστικά τους [14]. Έτσι το εργαλείο του IaC είναι σε θέση να παρακολουθεί με ακρίβεια τις αλλαγές στην υποδομή και να εκτελεί λειτουργίες όπως η τροποποίηση, διαγραφή και δημιουργία νέων πόρων μέσα από ένα φάσμα ελέγχου και πρόβλεψης.

<sup>1</sup><https://www.terraform.io/>

- **Resource Provisioning [Παροχή πόρων] :**

Πολύ σημαντικό χαρακτηριστικό του Terraform είναι ότι μπορεί να παρέχει πόρους αυτόμata σε cloud περιβάλλοντα, έχοντας αλληλεπίδραση με τα **API [Application Programming Interface]** που παρέχονται από τους cloud παρόχους. Συνεπώς, ο χρήστης δημιουργεί αρχεία διαμόρφωσης (.tf files) ανάλογα το σενάριο που καλείται να υλοποιήσῃ η ομάδα, και το μεταφράζει στο επιθυμητό αποτέλεσμα μέσα από κλήσεις API για την παροχή των απαραίτητων πόρων [14]. Με αυτό τον τρόπο, αυτό που πετυχαίνει αυτό το εργαλείο είναι η παροχή πόρων χωρίς πολυπλοκότητα, δίνοντας την ευκαρία στους χρήστες να εργαστούν με τον καθορισμό της επιθυμητής διαμόρφωσης υποδομής.

- **Modularity and Reusability :**

Επίσης σημαντικό χαρακτηριστικό είναι πως έχει δημιουργηθεί στο να χωρίζεται σε πολλά κομμάτια κώδικα σε αρχεία .tf και όλα μαζί να ενώνονται ώστε να παράξουν το απαραίτητο σενάριο που έχει γραφτεί στον κώδικα. Αυτό γίνεται με την χρήση **modules**, οι οποίες είναι αυτόνομες μονάδες διαμόρφωσης Terraform που ενσωματώνουν το εκάστοτε σενάριο για τους πόρους που θα χρειαστούμε. Έτσι αυτά τα μέρη μπορούν να χρησιμοποιηθούν εκ νέου σε νέα έργα και περιβάλλοντα και να μπορούν να απλοποιούν τη διαχείριση της υποδομής ώστε να βοηθούν άμεσα και γρήγορα στην κλιμάκωση του έργου καθώς βρίσκεται στη διαδικασία της ανάπτυξης [14].

## Πλεονεκτήματα του Terraform

- **Consistency and Reproducibility [Συνέπεια και Αναπαραγωγμότητα] :**

Το εργαλείο Terraform δίνει την δυνατότητα για δημιουργίας μιας αναπαραγωγικής διαδικασίας στην ανάπτυξη της υποδομής, ενισχύοντας με αυτόν τον τρόπο πανομοιότυπα περιβάλλοντα σε όλη την ανάπτυξη, την σταδιακή διεργασία και την παραγωγή. Έτσι, ελαχιστοποιούνται απρόβλεπτα ζητήματα που πιθανόν να προκύπτουν κατά την ανάπτυξη. Παράλληλα, δίνει την δυνατότητα εισαγωγής υπάρχουσας υποδομής και διασφαλίζει πως το σύστημα διαχείρισης κατάστασης αγγίζει με ακρίβεια την τρέχουσα κατάσταση των πόρων.

- **Unified Configuration [Ενοποιημένη διαμόρφωση] :**

Οι εταιρίες πληροφορικής έχουν την δυνατότητα με το Terraform να εισάγουν πόρους που υπάρχουν ήδη στην υποδομή τους, να ενοποιήσουν τη διαμόρφωση τους, απλοιώντας έτσι τη διαχείριση και παράγοντας την κατάλληλη συνοχή όλης της υποδομής.

- **Resource Monitoring [Παρακολούθηση πόρων] :**

Το Terraform είναι αποτελεσματικό καθώς έχει την δυνατότητα να διαχειρίζεται την κατάσταση των πόρων, να υπάρχει λεπτομερής καταγραφή παρακολούθησης των αλλαγών που γίνονται στα resources μέσα στην υποδομή, και διευκολύνει στο να υπάρχει έγκαιρη αντιμετώπιση ζητημάτων που δημιουργούνται κατά την ανάπτυξη. Από αυτό καταλαβαίνουμε πως το Terraform σαν εργαλείο IaC, στοχεύει στο να μπορεί να ελέγχει ολόκληρη την υποδομή.

- **Compatibility on platforms [Συμβατότητα στις πλατφόρμες] :**

Επίσης, ένα πολύ σημαντικό χαρακτηριστικό του Terraform, είναι πως μπορεί να υιοθετήσει υπηρεσίες υποδομής από διάφορους cloud providers, όπως για παράδειγμα είναι οι **AWS, Microsoft Azure, Google Cloud, DigitalOcean, Linode, κ.α.** Αυτό το κάνει ισχυρό, καθώς οι ομάδες προγραμματιστών και πληροφορικής είναι σε θέση να επιλέξουν τον πάροχο που ταιριάζει καλύτερα στις ανάγκες τους.

- **Collaboration and Transparency [Συνεργασία και Διαφάνεια] :**

Η εντολή "plan" στο Terraform παρέχει μια προεπισκόπηση των ενεργειών που θα εκτελέσει το Terraform, προωθώντας έτσι την ομαδική συνεργασία μεταξύ των μελών μιας ομάδας.

- **Community [Κοινότητα] :**

Το Terraform σαν εργαλείο, πέρα από όσα αναφέραμε παραπάνω, έχει και μια μεγάλη και ενεργή κοινότητα που συμβάλλει στο οικοσύστημα, καθώς υπάρχει σημαντική δυνατότητα συνεργασίες με βέλτιστες πρακτικές που έχουν δημιουργηθεί και έχουν διατεθεί σε όλους. Συνεπώς, οι ομάδες προγραμματιστών και πληροφορικής μπορούν να επωφεληθούν από τη συλλογική γνώση και εμπειρία που προσφέρει η κοινότητα, δίνοντας παραδείγματα στον πραγματικό κόσμο.

## Προκλήσεις του Terraform

Αρχικά το Terraform είναι σίγουρο πως έχει παίξει βασικό ρόλο στη διάδοση της έννοιας του Infrastructure as Code. Ωστόσο, είναι αναγκαίο να συμβάλλουμε σε αυτή την εργασία ώστε να καταγραφούν με σαφήνεια και οι προκλήσεις που μπορεί να υπάρχουν. Πιο συγκεκριμένα μπορούμε να πούμε τα εξής:

- **Learning Curve [Καμπύλη εκμάθησης] :**

Το Terraform ειδικά για αρχάριους ή νέους χρήστες στην Υποδομή ως Κώδικα έχει μια σημαντική δυσκολία στην εκμάθηση. Οι χρήστες που επιλέγουν αυτό το εργαλείο IaC θα πρέπει να είναι σε θέση να μάθουν και να κατανοήσουν την δηλωτική σύνταξη, τον τρόπο εγγραφής λειτουργικών μονάδων (microservices), αλλά και την διαχείριση των αρχείων κατάστασης.

- **Maturity of Providers [Ωριμότητα παρόχων] :**

'Όπως έχουμε αναφέρει σε προηγούμενες υποενότητες, το Terraform είναι συμβατό με πολλούς cloud παρόχους. Όμως, το σύνολο των χαρακτηριστικών των μεμονωμένων παρόχων ενδέχεται να διαφέρει αρκετά. Αυτό συμβαίνει καθώς αρκετοί cloud providers πιθανόν να μην είναι σε θέση να υποστηρίζουν συγκεκριμένους πόρους και δυνατότητες σε μια υποδομή σε σύγκριση με άλλους παρόχους.

- **Resources Abstraction Complexity [Πολυπλοκότητα αφαίρεσης πόρων]**

Το Terraform είναι σε θέση να μπορεί να αφερεί πόρους από διάφορους παρόχους cloud, όμως μερικές φορές μπορεί να οδηγεί σε δυσκολία στην κατανόηση των υπηρεσιών μιας υποδομής, και αυτό μπορεί να είναι σημαντικό στη βέλτιστη απόδοση και στις διαμορφώσεις που προυπήρχαν.

- **Limited Control Over Deployment Order [Περιορισμένος έλεγχος στην εντολή ανάπτυξης] :**

Μερικές φορές, η ανάλυση εξάρτησης του Terraform ενδέχεται να μην λειτουργεί όπως έχει αρχικοποιηθεί, οδηγώντας έτσι σε προκλήσεις στον έλεγχο της σειράς ανάπτυξης των πόρων, ειδικά σε σενάρια που έχουν αρκετή πολυπλοκότητα.

- **Third-Party Plugin Dependencies [Εξαρτήσεις προσθηκών τρίτων] :**

Είναι σημαντικό να αναφέρουμε πως μια βασική πρόκληση κατά την υιοθέτηση του Terraform είναι πως ορισμένες δυνατότητες πιθανόν να βασίζονται σε προσθήκες τρίτων. Ενώ αυτές οι προσθήκες επεκτείνουν τις δυνατότητες του Terraform, μπορούν σε σύντομο διάστημα μέσα στην υποδομή να εισάγουν ζητήματα συμβατότητας και να απαιτούν πρόσθετη προσπάθεια για τη διαχείριση της υποδομής.

Συμπερασματικά, το Terraform έχει κερδίσει επάξια την έννοια του ισχυρού εργαλείου IaC σε συνδιασμό με το κύκλο ζωής ανάπτυξης συστημάτων λογισμικού (SDLC), προσφέροντας πολλαπλά πλεονεκτήματα όσον αφορά την υποστήριξη το **automation**, **multi-cloud environments**, και την υποστήριξη της κοινότητας που έχει δημιουργηθεί. Από την άλλη πλευρά, οι προκλήσεις, όπως είναι η δυσκολία πολλές φορές στην εκμάθησης για αρχάριους και οι περιορισμοί που μπορεί να έχει ένας πάροχος στην υποστήριξη των πόρων και της υποδομής, δίνει μια κατεύθυνση στο να υπάρξει η προσπάθεια της κατανόησης των πλεονεκτημάτων και των προκλήσεων, ώστε να μπορούν οι ομάδες με τα σενάρια έργων που έχουν προς υλοποίηση, να αποφασίζουν για τις ανάγκες διαχείρισης υποδομής, εάν το Terraform είναι ιδανικό.

## Αρχιτεκτονική του Terraform

Το Terraform όπως αναλύθηκε, είναι ένα εργαλείο ανοιχτού κώδικα, και δίνει τη δυνατότητα στις ομάδες προγραμματιστών να παρέχουν την υποδομή που είναι αναγκαία με απλό, αποτελεσματικό και δηλωτικό τρόπο μέσω επαναλαμβανόμενου κώδικα, είτε πρόκειται για περιβάλλοντα **cloud** είτε για περιβάλλοντα εσωτερικού χώρου **on-premises environments**, όπου οι οργανισμοί στεγάζουν στις δικές τους εγκαταστάσεις και συντηρούνται από τους ίδιους. Συνεπώς είναι σημαντικό να αναλυθεί η αρχιτεκτονική που ακολουθεί το Terraform και φαίνεται στο σχήμα 21 για να ολοκληρώσει τις απαραίτητες διεργασίες του IaC. Το Terraform λοιπόν περιλαμβάνει δύο στοιχεία που είναι ικανά να ρυθμίσουν ολόκληρη την υποδομή: το **Terraform Core** και τους **Providers**. Παρακάτω θα δούμε πως μέσα από αυτά στα στοιχεία γίνεται η υλοποίηση που χρειαζόμαστε από το Terraform:

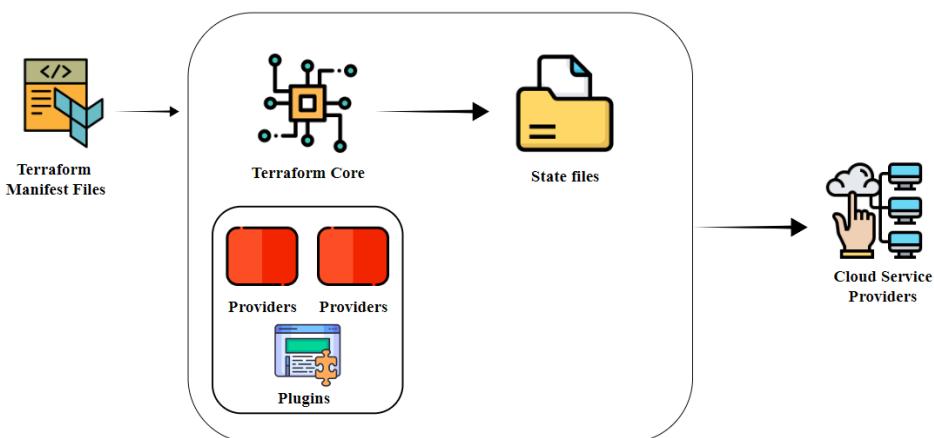
**Terraform Core :** Το Terraform Core, ο πυρήνας του δηλαδή, είναι υπεύθυνος στο να μπορεί να κάνει ανάγνωση των διαμορφώσεων και να δημιουργήσει το γράφημα εξάρτησης, καθώς βασίζεται σε μεγάλο βαθμό στη θεωρία των γραφημάτων. Είναι σημαντικό να αναφερθεί πως χρησιμοποιεί στην ουσία δύο πηγές εισόδου, ώστε να μπορεί να λειτουργεί σωστά. Οπότε παρακάτω δίνουμε μια πρώτη εικόνα στο ποια είναι η λειτουργία του [14]:

- **Initialization [Αρχικοποίηση] :** Η πρώτη και κύρια πηγή εισόδου είναι όταν ο χρήστης δίνει την διαμόρφωση του Terraform και την περιγραφή των πόρων για το ποιοι πόροι χρειάζεται να χρησιμοποιηθούν στην υποδομή. Η δεύτερη πηγή εισόδου είναι όλα τα δεδομένα που παρέχονται στην Terraform σχετικά με την υπάρχουσα ρύθμιση υποδομής.

- **Planning [Σχεδιασμός]** : Εφόσον έχει γίνει η αρχικοποίηση αυτών των εισόδων, το Terraform έχει την δυνατότητα ώστε να αποφασίσει ποιες είναι οι ενέργειες οι οποίες πρέπει να γίνουν για την υποδομή. Παράλληλα, αναλύει την κατάσταση που επιθυμεί και καθορίζει ο χρήστης, την συγκρίνει με την υπάρχουσα κατάσταση, και ρυθμίζει την αρχιτεκτονική που εξαλείφει τα κενά στην υποδομή.
- **Applying [Εφαρμογή]** : Έτσι, το Terraform Core προσδιορίζει τις ενέργειες που πρέπει να πραγματοποιηθούν, να τροποποιηθούν ή να διαγραφούν ώστε να πετύχει την πλήρη αξιοποίηση της υπάρχουσας υποδομής που μπορεί να έχουμε.

**Providers/Provisioners [Πάροχοι]** : Το δεύτερο στοιχείο του Terraform που είναι οι Providers. Οι πάροχοι συχνά αναφέρονται ως μεμονωμένα εξωτερικά στατικά δυαδικά αρχεία. Ο πυρήνας του Terraform επικοινωνεί με τα plugins (πρόσθετα) μέσω της κλήσης απομακρυσμένης διαδικασίας (**Remote Procedure Call (RDC)**), όταν η ανάπτυξη και ο σχεδιασμός της υποδομής και της εφαρμογής βρίσκονται σε εξέλιξη. Όλα τα Plugins είναι γραμμένα στην γλώσσα προγραμματισμού Go. Στο configuration file κάθε πάροχος είναι και ένα plugin. Παρακάτω δίνουμε μια εικόνα για το ποια είναι η λειτουργία των Providers/Provisioners [14]:

- **Initialization [Αρχικοποίηση]** : Στο Terraform οι πάροχοι είναι cloud providers. Όλοι αυτοί οι cloud providers μπορούν να προσφέρουν άμεση πρόσβαση στους πόρους. Αν πάρουμε ως παράδειγμα έναν δημοφιλή πάροχο που είναι οι Amazon Web Services, οι χρήστες μπορούν να έχουν πρόσβαση σε EC2 Instances και ότι άλλο εμπεριέχει, και να ξεκινήσει η ομάδα προγραμματιστών σε έναν οργανισμό, να ξεκινήσουν να παρέχουν την υποδομή που επιθυμούν.
- **Applying and Planning [Εφαρμογή και Σχεδιασμός]** : Οι πάροχοι είναι υπεύθυνοι ώστε να καθορίσουν τους πόρους που μπορεί να διαχειριστεί το Terraform σε μια υπηρεσία, όπως για παράδειγμα σε ένα EC2 Instance, και για την μετέπειτα μετάφραση των configuration files του Terraform μέσω κλήσων API που είναι ειδικά για αυτή την υπηρεσία που ζητήθηκε.



Εικόνα 22. Αρχιτεκτονική του Terraform

## Ροή εργασίας του Terraform

Στις προηγούμενες υποενότητες για το εργαλείο Terraform, εξερευνήσαμε τα βασικά στοιχεία και τα οφέλη του Terraform, συμπεριλαμβανομένης της αρχιτεκτονικής, και των χαρακτηριστικών του. Τώρα, θα εμβαθύνουμε στη ροή εργασίας Terraform [**Terraform workflow**], όπου είναι ένα κρίσιμο συστατικό του συνολικού οικοσυστήματος του και το οποίο είναι στην ουσία η διαδικασία χρήσης του Terraform για τη διαχείριση και την παροχή υποδομής. Παρακάτω θα αναλύσουμε τα τρία στάδια του Terraform workflow [14] :

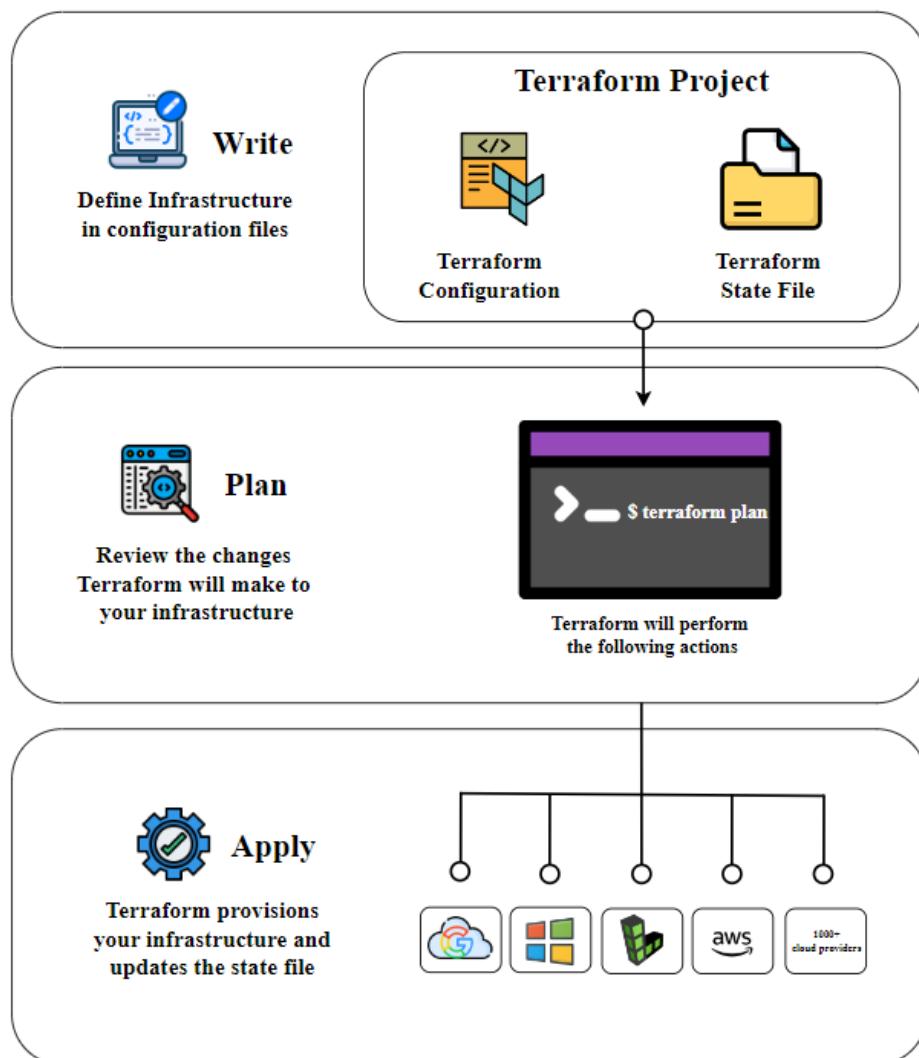
- **Write [Εγγραφή]**: Αυτό είναι το πρώτο στάδιο της ροής εργασίας του Terraform, το στάδιο της εγγραφής. Κυρίως εστιάζει στην ανάπτυξη του κώδικα και την δημιουργία πόρων υποδομής, που χρειάζεται το σενάριο που είναι προς υλοποίηση. Είναι σημαντικό πως δίνει την δυνατότητα να υπάρχει η διαμόρφωση υποδομής για πολλούς cloud παρόχους με τους πόρους που διαθέτουν. Τα αρχεία διαμόρφωσης που δημιουργούνται είναι στην μορφή .tf ή υπάρχει και η ευελιξία να αποθηκεύονται σε μορφή JSON, και πιο συγκεκριμένα με επέκταση .tf.json.

Ας πάρουμε ένα παράδειγμα για να γίνει πιο κατανοητό: έστω ότι ένας DevOps Engineer εργάζεται σε μία εταιρεία και σκοπεύει να δημιουργήσει ένα mini-cluster. Θα πρέπει πρώτα απ'όλα να χρησιμοποιήσει την καθορισμένη επέκταση .tf στο αρχείο διαμόρφωσης του. Οι εικονικές μηχανές (virtual machines) που θα χρησιμοποιήσει για παράδειγμα είναι 3 VMs, ένα εικονικό ιδιωτικό cloud δίκτυο (**Virtual Private Cloud** network), μία ομάδα ασφαλείας (**security group**) και μια υπηρεσία εξισορρόπησης φορτίου (**Load Balancer**). Όλο αυτό το σύμπλεγμα των πόρων, το Terraform μπορεί να το υλοποιήσει σε configuration files.

- **Plan [Σχέδιο]** : Συνεχίζοντας στο δεύτερο στάδιο του Terraform workflow, με το παράδειγμα που αναφέραμε πιο πάνω, αφού γράψουμε τον κώδικα διαμόρφωσης, το επόμενο βήμα περιλαμβάνει την εκτέλεση του εργαλείου Terraform για τη δημιουργία ενός σχεδίου. Αυτό το εργαλείο μπορεί να εκτελεστεί χρησιμοποιώντας την τοπική διεπαφή γραφικής εντολών (**CLI**) ή μέσω μιας γλώσσας υψηλού επιπέδου που συνδέεται με το πλαίσιο Terraform μέσω του kit ανάπτυξης cloud (**CDK**), που θα αναλύσουμε αργότερα. Στην συνέχεια, κατά την εκτέλεση του σχεδίου Terraform, σαρώνει τοπικούς καταλόγους για αρχεία διαμόρφωσης με επεκτάσεις .tf ή tf.json και τα επεξεργάζεται για να δημιουργήσει μια λίστα ενεργειών που θα εκτελεστούν από τους πάροχο/ους. Αυτή η λίστα, γνωστή ως σχέδιο εκτέλεσης, περιλαμβάνει ενέργειες για την δημιουργία, την ενημέρωση ή την καταστροφή πόρων, προκειμένου να ευθυγραμμιστεί η υποδομή, η οποία έχει αρχικοποιηθεί στο αρχείο κατάστασης (**Terraform State File**).

Το Terraform state αρχείο, περιέχει όλες τις λεπτομέρειες όλων των πόρων υποδομής και μπορεί να υπάρχει είτε τοπικά στο σύστημα αρχείων είτε σε απομακρυσμένη τοποθεσία. Ενδεικτικά, για να είμαστε πιο σαφείς, μέσα στην ρύθμιση υπάρχει μια εικονική μηχανή "Virtual Machine 1" που λειτουργεί στην πλατφόρμα Elastic Compute (EC2). Όταν το Terraform εκτελείται για τη δημιουργία του σχεδίου, λαμβάνει υπόψη την υπάρχουσα κατάσταση και αποφασίζει για να τις ενέργειες που πρέπει να γίνουν. Σκοπός λοιπόν, είναι να προσφέρει την εφαρμογή ενημερώσεων όπου χρειάζεται, είτε η υποδομή θα παραμείνει αμετάβλητη.

- **Apply [Εφαρμογή]** : Έχοντας φτάσει λοιπόν στο τρίτο και τελευταίο στάδιο της ροής εργασίας Terraform, το εργαλείο όταν λειτουργεί όπως έχει προγραμματιστεί, κάθε λειτουργία θα υλοποιείται από τον αντίστοιχο πάροχο που έχει δηλωθεί. Το Terraform αλληλεπιδρά με προσθήκες του παρόχου του και στη συνέχεια με το αντίστοιχο API του παρόχου cloud (π.χ Linode). Συνεπώς, ως μέρος της εκάστοτε εφαρμογής είναι στο να αντιπροσωπεύει τις αλλαγές που έγιναν ή δημιουργήθηκαν στην υποδομή. Παρακάτω, παρατηρούμε τα τρία στάδια του Terraform workflow σε σχεδιάγραμμα:



Εικόνα 23. Ροή εργασίας του Terraform

## HashiCorp Configuration Language [Γλώσσα Διαμόρφωσης]

Όπως έχουμε αναφέρει και στην αρχή της υποενότητας για το εργαλείο Terraform, χρησιμοποιεί μια δηλωτική σύνταξη για να δημιουργεί τα configuration αρχεία, όπου είναι το HashiCorp Configuration Language ή σε συντομογραφία HCL. Σε αυτό το σημείο, είναι σημαντικό να εμβαθύνουμε περισσότερο σε αυτό το σημαντικό στοιχείο του Terraform, για να μπορούμε μετά να κατανοούμε τέτοια αρχεία [14].

Πιο συγκεκριμένα μετά την σάρωση τέτοιων configuration αρχείων, το Terraform όπως είπαμε δημιουργεί το κατάλληλο πλάνο/σχέδιο για την υποδομή. Η γλώσσα λοιπόν HashiCorp Configuration Language χρησιμοποιείται για αυτά τα αρχεία. Σε σύγκριση με την δηλωτική προσέγγιση που έχουμε αναφέρει σε προγενέστερη ενότητα, η δηλωτική έχει το πλεονέκτημα ότι η κωδικοποίηση της υποδομής γίνεται απευθείας στην επιθυμητή κατάσταση [14]. Είναι σημαντικό να αναφερθεί πως πρέπει να καθορίζεται κάθε ενδιάμεσο βήμα που απαιτείται για την επίτευξη της κατάστασης στόχου της υποδομής.

Επιπρόσθετα, πρέπει ο βασικός στόχος της HCL να είναι ο καθορισμός των πόρων. Συνεπώς, κάθε block κώδικα μέσα στο configuration αρχείο αντιπροσωπεύει ένα στοιχείο της υποδομής. Όλο αυτό το αρχείο HCL στην ουσία είναι η καθοδήγηση που χρειάζεται το Terraform ώστε να γνωρίζει πώς να χειρίζεται ένα συγκεκριμένο σύνολο πόρων υποδομής.

Για παράδειγμα, ένα block κώδικα σε ένα αρχείο .tf διαμόρφωσης του Terraform, μπορεί να αντιστοιχεί σε διάφορα μέρη (components) ενός VPC (Virtual Private Cloud), χρειάζεται και την προσθήκη του παρόχου που έχει αποφασιστεί ότι θα χρησιμοποιηθεί. Βέβαια, προκειμένου να μετατραπούν αυτά τα block κώδικα που περιγράφουν τους πόρους της υποδομής σε κλήσεις API προς τον πάροχο υποδομής, ένας πάροχος καθορίζει ποια προσθήκη παρόχου θα χρησιμοποιηθεί. Μέσα στην υποδομή του παρόχου, ένας πόρος ορίζεται χρησιμοποιώντας το block πόρων [14]. Αυτά τα block μετά μετατρέπονται σε κλήσεις API προς την υπηρεσία υποδομής προορισμού του παρόχου για δημιουργία, επεξεργασία ή και διαγραφή. Ένα τέτοιο απλό παράδειγμα θα μπορούσε να ήταν:

Πίνακας 1. Παράδειγμα terraform.tf αρχείου

| terraform.tf   |
|--|
| <pre>#provider . tf  resource "aws_vpc" "main" {   cidr_block = "10.0.0.0/16"    tags = {     Name = "Project-VPC"   } }</pre> |

## Framework Environments

Σε αυτό το σημείο, έχοντας δώσει μια σαφή εικόνα για όλα τα χαρακτηριστικά του Terraform ως Infrastructure as Code εργαλείο, καθώς και τα πλεονεκτήματα/προκλήσεις που μπορεί να έχει η υιοθέτηση του, αλλά και όλα στοιχεία που απαρτίζουν αυτό το εργαλείο, είναι πολύ σημαντικό να μπορέσουμε τώρα να δώσουμε μια αναφορά για τα framework περιβάλλοντα που μπορεί το Terraform να εκτελεστεί.

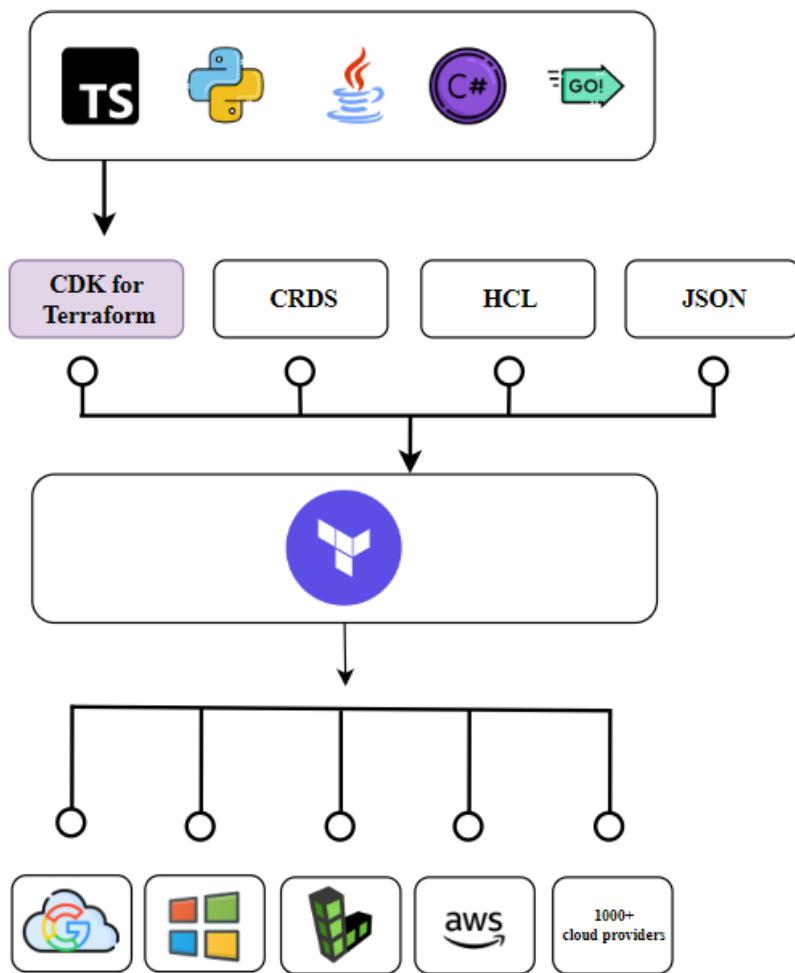
Αρχικά, το Terraform έχει την δυνατότητα να εκτελεστεί στα ακόλουθα περιβάλλοντα: **CDK, Terraform Cloud, Terraform Enterprise** και **CLI**. Το πιο δημοφιλές είναι να χρησιμοποιείται το CLI (Command-Line Interface). Το Terraform εκτελείται από την ίδια θέση που δημιουργείται το αρχείο κατάστασης (Terraform State file). Πιο συγκεκριμένα παρακάτω έχουμε μια αναφορά στα framework environments [14]:

- **Terraform Cloud :** Το Terraform Cloud είναι μια υπηρεσία που βασίζεται σε σύννεφο και επιτρέπει στους προγραμματιστές να διαχειρίζονται και να παρέχουν το infrastructure στο cloud. Παρέχει μια κεντρική πλατφόρμα για τη διαχείριση πολλών περιβαλλόντων, συμπεριλαμβανομένης της ανάπτυξης, των σταδίων και της παραγωγής της υποδομής.
- **Terraform Enterprise :** Από την άλλη πλευρά, υπάρχει μια αυτο-φιλοξενούμενη έκδοση του Terraform Cloud και ονομάζεται Terraform Enterprise. Το προϊόν παρέχει ένα πανομοιότυπο σύνολο λειτουργιών που βασίζονται σε cloud, αλλά προορίζεται για ανάπτυξη στο ιδιωτικό cloud ενός οργανισμού. Το kit ανάπτυξης cloud είναι μια πρόσθετη έκδοση του CLI ή του τοπικού περιβάλλοντος (CDK).
- **Terraform CLI (Command-Line Interface) :**

Το Terraform CLI είναι ένα εργαλείο διεπαφής γραμμής εντολών που δίνει τη δυνατότητα στις ομάδες προγραμματιστών να διαχειρίζονται την Υποδομή ως Κώδικα (IaC) χρησιμοποιώντας τη γλώσσα διαμόρφωσης HashiCorp (HCL). Έτσι, με το Terraform CLI, μπορούν να ορίζουν, να παρέχουν και να διαχειρίζονται πόρους υποδομής σε μορφή αναγνώσιμη από τον άνθρωπο, η οποία μπορεί να τροποποιηθεί, να επαναχρησιμοποιηθεί και να μοιραστεί μεταξύ των ομάδων.

- **Configuration Development Kit :** Το CDK επιτρέπει σε πέντε υποστηριζόμενες γλώσσες υψηλού επιπέδου να αναπτύσσουν και να εφαρμόζουν τον καθορισμό της υποδομής με το Terraform αντί να εκτελούν απευθείας το εργαλείο CLI. Γ'αυτό τον λόγο είναι ένας τρόπος για την ανάπτυξη υποδομής χρησιμοποιώντας διαφορετικές γλώσσες προγραμματισμού χρησιμοποιώντας το Terraform Cloud Development Kit.

Έτσι δεν χρειάζεται να γίνει η ανάπτυξη στη γλώσσα διαμόρφωσης HashiCorp και εκτέλεση μέσω του εργαλείου CLI, καθώς το CDK παρέχει πρόσβαση σε ολόκληρο το οικοσύστημα Terraform. Επιπλέον, είναι ευκολότερο για έναν χρήστη να συνδεθεί με μια υπάρχουσα αλυσίδα εργαλείων δοκιμών. Οι γλώσσες που υποστηρίζονται αυτήν τη στιγμή περιλαμβάνουν τα ακόλουθα: `Go`, `TypeScript`, `Python`, `Java`, `C Sharp`. Ενώ το Terraform CLI απαιτείται για τον κώδικα διαμόρφωσης σε HCL ή JSON, το CDK μπορεί να κληθεί από οποιαδήποτε από τις πέντε υποστηριζόμενες γλώσσες του. Οι διαφορετικές διαδρομές εισόδου Terraform απεικονίζονται στο παρακάτω σχήμα:



Εικόνα 24. Terraform CDK

### 4.5.3 Ansible



#### Εισαγωγή στην Ansible

Σε αυτή την υποενότητα θα συνεχίσουμε με ένα άλλο Infrastructure as Code εργαλείο, που είναι το Ansible. To Ansible<sup>2</sup> είναι ένα εργαλείο αυτοματισμού ανοιχτού κώδικα [open-source], που επιτρέπει τη διαχείριση και την ενορχήστρωση πολύπλοκων υποδομών και εφαρμογών πληροφορικής. Είναι σημαντικό να αναφέρουμε ότι αναπτύχθηκε από τη Red Hat, και είναι μια δημοφιλής επιλογή μεταξύ των ομάδων DevOps και των διαχειριστών συστημάτων λόγω της ευκολίας χρήσης, της ευελιξίας και της επεκτασιμότητας του.

Στην συνέχεια της ενότητας θα προσπαθήσουμε να θέσουμε τις βάσεις για την ανάλυση του Ansible ως εργαλείο Infrastructure as Code, καθώς η ενσωμάτωση που προσφέρει με πολλούς cloud providers και των πλατφορμών τους, με εργαλεία container orchestration και άλλα εργαλεία DevOps, είναι σίγουρα ένα εργαλείο που το καθιστά βασικό στοιχείο της σύγχρονης διαχείρισης της υποδομής. Αυτή η εισαγωγή τείνει να παρέχει μια σύντομη επισκόπηση των βασικών χαρακτηριστικών, των πλεονεκτημάτων και προκλήσεων, της αρχιτεκτονικής του αλλά και use cases [περιπτώσεις χρήσης] του Ansible.

#### Βασικά χαρακτηριστικά του Ansible

Παρακάτω δίνουμε μια σαφή εικόνα για τα χαρακτηριστικά του Ansible [15] :

- **Declarative Configuration Management:** Το Ansible χρησιμοποιεί και εκείνο μια δηλωτική προσέγγιση διαχείρισης διαμόρφωσης μιας υποδομής, όπου οι ομάδες ορίζουν την επιθυμητή κατάσταση της υποδομής και των εφαρμογών τους και το Ansible έχει την δυνατότητα διασφάλισης ότι το σενάριο επιτυγχάνεται.
- **Agentless Architecture :** Είναι σημαντικό να αναφέρουμε πως το Ansible δεν απαιτεί να γίνει κάποια εγκατάσταση λογισμικού στα συστήματα προορισμού, αλλά χρησιμοποιεί πρωτόκολλα όπως το SSH, και το PowerShell. Αυτό το χαρακτηριστικό το καθιστά εξαιρετικά ευέλικτο εργαλείο για τη διαχείριση συγκεκριμένων περιβάλλοντων.
- **Playbooks:** Το Ansible χρησιμοποιεί αρχεία **YAML**, γνωστά και ως **playbooks**. Αυτά τα playbooks αναπαραγωγής εκτελούνται στη συνέχεια σε συστήματα προορισμού, τα οποία μπορούν να περιλαμβάνουν servers, virtual machines, containers, και cloud instances, για να διασφαλιστεί στο μέγιστο η καθορισμένη κατάσταση σε μια υποδομή.

<sup>2</sup><https://www.ansible.com/>

- **Modules :** Επίσης το Ansible είναι σε θέση να παρέχει ένα ευρύ φάσμα λειτουργικών μονάδων που μπορούν να χρησιμοποιηθούν για την εκτέλεση συγκεκριμένων εργασιών, όπως η εγκατάσταση λογισμικού, η διαμόρφωση αρκετών ρυθμίσεων δικτύου και η διαχείριση βάσεων δεδομένων.
- **Roles :** Οι Ansible roles είναι μονάδες του οργανισμού που διευκολύνουν την κοινή χρήση κώδικα αυτοματισμού.
- **Inventories :** Τα αποθέματα Ansible χρησιμοποιούνται για τη διαχείριση της λίστας των μηχανημάτων που χρησιμοποιούνται με το Ansible.
- **Idempotence :** Ένα εξίσου χαρακτηριστικό του Ansible, είναι ότι έχει δημιουργηθεί για να είναι "ανίκανο". Πιο συγκεκριμένα, αυτό σημαίνει πως η εκτέλεση του ίδιου playbook που έχει δημιουργηθεί για ένα σενάριο, μπορεί να χρησιμοποιηθεί και να εκτελεστεί πολλές φορές καθώς δεν θα προκαλέσει αλλαγές εάν έχει ήδη επιτευχθεί η επιθυμητή κατάσταση.

## Πλεονεκτήματα του Ansible

Σε αυτή την ενότητα είναι σημαντικό να δώσουμε έμφαση στα πλεονεκτήματα που έχει το Ansible το οποίο είναι ευρέως διαδεδομένο εργαλείο αυτοματισμού ανοιχτού κώδικα και έχει γίνει μια επιλογή που δίνει λύσεις ως προς την διαχείριση της Υποδομής ως Κώδικα (IaC). Έτσι, θα μας βοηθήσει να κατανοήσουμε καλύτερα τα οφέλη του και θα μας δώσει μια σαφή εικόνα για το τι μπορεί να προσφέρει έαν κάποια ομάδα DevOps το χρησιμοποιήσει. Αναλυτικότερα λοιπόν έχουμε [16]:

- **Productivity - Efficiency [Παραγωγικότητα - Αποτελεσματικότητα] :** Το Ansible έχει την δυνατότητα να αυτοματοποιήσει την υποδομή και επιτρέπει στις ομάδες που εργάζονται σε ένα συγκεκριμένο έργο να επαναλαμβάνουν τις εργασίες, όπως την παροχή πόρων, την διαχείριση διαμόρφωσης και την ανάπτυξη των συστημάτων λογισμικού. Συνεπώς, το Ansible καταργεί τις μη αυτόματες διαδικασίες, και βοηθά τις ομάδες να εξοικονομούν χρόνο και πόρους, επιτρέποντας η εστίαση των ομάδων να γίνει μέσω στρατηγικών πρωτοβουλιών.
- **Reliability [Αξιοποστία] :** Όπως αναφέραμε πιο πάνω, το Ansible ακολουθεί μια δηλωτική προσέγγιση στη διαχείριση της υποδομής, διασφαλίζοντας έτσι ότι η επιθυμητή κατάσταση της υποδομής, εφαρμόζεται με συνέπεια σε όλα τα περιβάλλοντα.
- **Scalability [Επεκτασιμότητα] :** Έπίσης ένα βασικό χαρακτηριστικό που έχει αποτυπωθεί προηγουμένως, είναι ότι το Ansible έχει agentless αρχιτεκτονική. Αυτό το καθιστά μια εξαιρετικά επεκτάσιμη και ευέλικτη λύση. Οι εταιρίες είναι σε θέση να διαχειρίζονται και να αναπτύσσουν υποδομές σε πολλούς cloud παρόχους, σε physical servers, και virtual machines, χρησιμοποιώντας ένα ενιαίο πλαίσιο αυτοματισμού.
- **Version Control :** Φυσικά και το Ansible επιτρέπει στις ομάδες να διαχειρίζονται το configuration τους μέσα από συστήματα ελέγχου έκδοσης, όπως το Git. Έτσι, προωθεί την συνεργασία, την δυνατότητα αλλαγής στον κώδικα εύκολα και γρήγορα, καθώς και τις αλλαγές που μπορεί να γίνονται.
- **Fast Resolution [Γρήγορη λύση] :** Το Ansible δίνει την δυνατότητα μέσω των αυτοματισμών να προσφέρει ελαχιστοποίηση των κινδύνων από λάθη του συστήματος, καθώς εντοπίζει και επιλύει άμεσα τα προβλήματα. Συνεπώς, η δυνατότητα

γρήγορης ανάπτυξης ενημερώσεων και διαμορφώσεων σε όλη την υποδομή μπορεί να οδηγήσει σε ταχύτερη αντιμετώπιση όποιων προβλημάτων.

- **Security [Ασφάλεια]** : Το Ansible όπως θα δούμε και σε ενότητα παρακάτω, δίνει την δυνατότητα να αυτοματοποιεί τις διεργασίες που έχουν να κάνουν με την ασφάλεια, καθώς μπορεί να σαρώσῃ για τυχόν ευπάθειες και να διαχειριστεί αυτόματα τις ενημερώσεις κώδικα. Έτσι, δίνει την δυνατότητα να μπορεί να ελέγχει και να διασφαλίζει την ασφάλεια σε όλη την υποδομή.
- **Cost Savings [Εξοικονόμηση κόστους]** : Το Ansible είναι ένα εργαλείο open source και εξαλείφει την ανάγκη για δαπάνη σε χρεώσεις που σχετίζονται με τον αυτοματισμό. Παράλληλα με την αποδοτικότητα, την αυτοματοποίηση και την παραγωγικότητα, το Ansible δίνει την δυνατότητα μακροπρόθεσμα σε οργανισμούς για σημαντική εξοικονόμηση κόστους, που είναι πολύ σημαντικό.

## Προκλήσεις του Ansible

Σε αυτή την υποενότητα είναι σημαντικό να δώσουμε και μια αποτύπωση από τις προκλήσεις που μπορεί να υπάρχουν για το Ansible. Παρόλο που είναι ένα ισχυρό εργαλείο IaC, έχει ορισμένα μειονεκτήματα που πρέπει να γνωρίζουμε. Αυτοί οι περιορισμοί μπορεί να είναι ο λόγος για να μην το χρησιμοποιήσουν οι ομάδες που αναπτύσσουν ένα έργο ή μια περίπτωση χρήστης, και θα πρέπει να λαμβάνονται υπόψη κατά την επιλογή μια λύσης αυτοματισμού. Πιο αναλυτικά έχουμε [16]:

- **Limited Windows Support [Περιορισμένη υποστήριξη των Windows]** : Το Ansible έχει περιορισμένη δυνατότητα για υποστήριξη σε Windows. Αν και υποστηρίζει κόβμους Windows, βασίζεται στην απομακρυσμένη λειτουργία PowerShell αντί του ssh. Συνεπώς, χρειάζεται μια μηχανή ελέγχου Linux που εκτελεί Python, για τη διαχείριση κεντρικών υπολογιστών των Windows, κάτι που μπορεί να είναι ένας σημαντικός περιορισμός για ομάδες που λειτουργούν σε περιβάλλοντα Windows.
- **Limited Enterprise Support Experience [Περιορισμένη εμπειρία υποστήριξης επιχειρήσεων]** : Άλλη μια πρόκληση που φαίνεται να υπάρχει είναι η έλλειψη υποστήριξης σε μεγάλες επιχειρήσεις, σε αντίθεση με τους ανταγωνιστές της όπως είναι το Puppet, το Chef και το Terraform. Συνεπώς, ο περιορισμός αυτός μπειώνει αρκετά την υπευθυνότητα του Ansible ως προς την επιλογή για επιχειρήσεις και έργα μεγάλης κλίμακας.
- **Debugging Challenges [Προκλήσεις εντοπισμού σφαλμάτων]** : Σε προηγούμενη ενότητα αναφέραμε πως υπάρχει η δυνατότητα της γρήγορης λύσης σε σφάλματα, που είναι σημαντικό κατά την ανάπτυξη υποδομής, όμως είναι περιορισμένη. Το πρόγραμμα εντοπισμού σφαλμάτων μπορεί μόνο να δώσει ή να ορίσει μεταβλητές, όμως δεν παρέχει λεπτομερείς πληροφορίες σχετικά με τις εσωτερικές λειτουργίες των μονάδων. Συνεπώς, αυτή η πρόκληση μπορεί να προκαλέσει μια χρονοβόρα διαδικασία για την διάγνωση σφαλμάτων σε πολύπλοκα Playbooks.
- **Stateless [Ελλειψη Κατάστασης]** : Ένα από τα βασικά ζητήματα και πρόκληση του Ansible είναι πως υπάρχει έλλειψη λεπτομερής κατάστασης, σε αντίθεση με άλλα εργαλεία αυτοματισμού. Το Ansible δεν διατηρεί έναν ολοκληρωμένο κατάλογο καταστάσεων και εξαρτήσεων συστήματος, όπως κάνει για παράδειγμα το **Terraform**. Αντιθέτως, απλά εκτελεί τις εργασίες που έχει διαδοχικά μέχρι να ολοκληρωθούν, να αποτύχουν ή να παρουσιάσουν κάποιο σημαντικό σφάλμα. Συνεπώς,

δεν είναι εύκολο σε πολύπλοκα περιβάλλοντα να διασφαλιστεί η τρέχουσα κατάσταση της υποδομής.

- **Insufficient User Interface [Ανεπαρκής διεπαφή χρήστη]** : Επίσης σημαντική πρόκληση για το Ansible είναι πως η αρχική διεπαφή γραμμής εντολών του Ansible ήταν ανεπαρκής. Παρόλο που υπήρξε βελτίωση με την εισαγωγή του AWX και μετέπειτα του Ansible Tower<sup>3</sup>, για την αντιμετώπιση του ζητήματος, η διεπαφή χρήστη εξακολουθεί να θέλει σημαντικές βελτιώσεις. Αυτή η πρόκληση μπορεί να παρέχει σημαντικούς περιορισμούς κατά την διαχείριση της υποδομής.

### **Αρχιτεκτονική του Ansible**

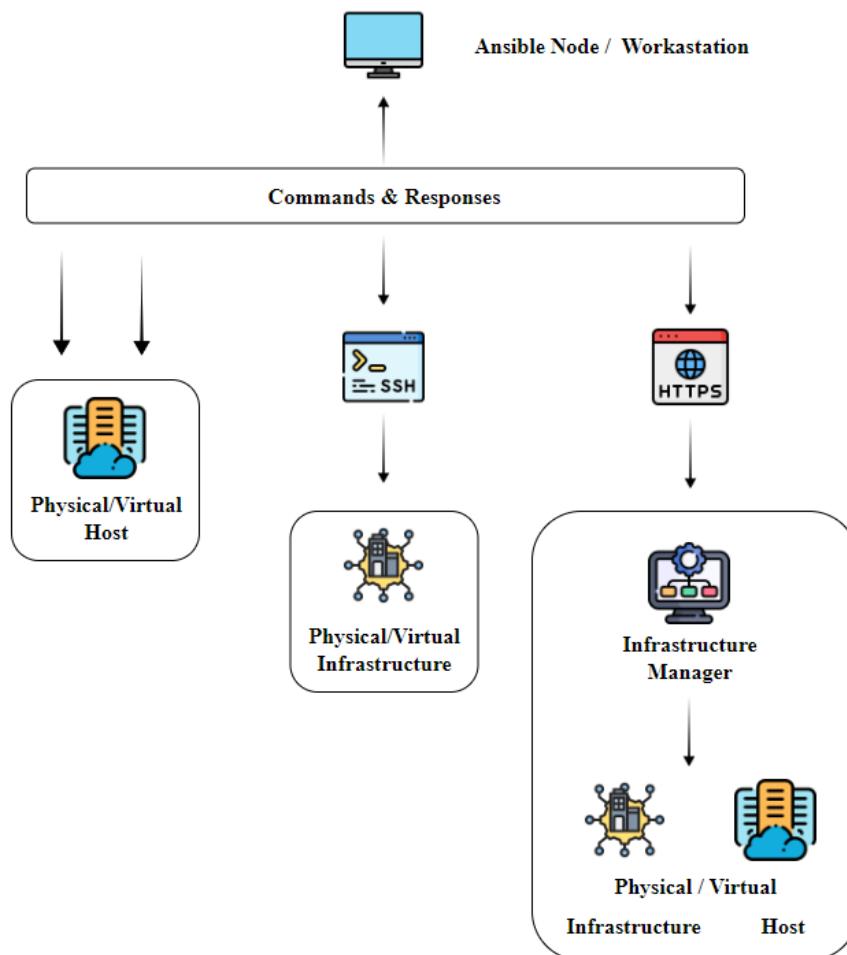
Το Ansible είναι ένα εργαλείο αυτοματισμού ανοιχτού κώδικα (**open source**), που έχει σχεδιαστεί, - όπως έχουμε αναφέρει - για την απλοποίηση της διαχείρισης της υποδομής. Είναι σημαντικό σε αυτό το σημείο να αναλύσουμε την αρχιτεκτονική αυτού του εργαλείου IaC. Η αρχιτεκτονική του όπως παρατηρούμε και στο σχεδιάργραμμα 24 που αποτυπώνει την αρχιτεκτονική του Ansible της υποενότητας αυτής, είναι χτισμένη γύρω από έναν κόμβο ελέγχου και τους διαχειριζόμενους κόμβους, διασφαλίζοντας αποτελεσματικό και επεκτάσιμο αυτοματισμό. Παρακάτω δίνεται μια σαφή ανάλυση της αρχιτεκτονικής [17]:

- **Ansible Node / Workstation [Ansible Κόμβος / Σταθμός Εργασίας]** : Είναι ένα κεντρικό σημείο ελέγχου όπου εκκινούνται και λαμβάνονται όλες οι εντολές και οι απαντήσεις. Για παράδειγμα, ένα μηχάνημα όπου είναι εγκατεστημένο το Ansible και από το οποίο εκτελούνται όλα τα Playbooks που έχουν αναπτυχθεί. Στην αρχιτεκτονική του Ansible χρησιμοποιούνται τα εξής:
- **Communication Protocols [Πρωτόκολλα Επικοινωνίας]** :
  - ! **SSH** : To SSH [**Secure Shell**] χρησιμοποιείται για την ασφαλή επικοινωνία με φυσικούς ή εικονικούς κεντρικούς υπολογιστές και υποδομές.
  - ! **HTTPS / REST**: Χρησιμοποιούνται για την ασφαλή και ομαλή επικοινωνία με τους διαχειριστές μιας υποδομής. Επίσης, διευκολύνει τις αλληλεπιδράσεις που βασίζονται σε API για τη διαχείριση φυσικών και εικονικών υποδομών.
- **Target Nodes [Κόμβοι στόχοι]** :
  - ! **Physical / Virtual Hosts [Φυσικοί / Εικονικοί διακομιστές]** : Σε αυτό το σημείο έχουμε άμεση διαχείριση των διακομιστών ή των εικονικών μηχανών που υπάρχουν στην υποδομή.
  - ! **Physical / Virtual Infrastructure [Φυσική / Εικονική Υποδομή]** : Στην αρχιτεκτονική του Ansible γίνεται διαχείριση πιο περίπλοκων ρυθμίσεων, όπως είναι για παράδειγμα **Kubernetes clusters** ή άλλα περιβάλλοντα cloud. Όλες οι εντολές αποστέλλονται μέσα από το secure shell protocol, στους κόμβους ελέγχου.
- **Infrastructure Managers [Υπεύθυνοι Υποδομής]** :
  - ! Υπάρχουν διάφορα εργαλεία όπως για παράδειγμα το OpenStack, το VMware ή οι πάροχοι cloud [AWS, Azure, Linode, DigitalOcean, GCP]

<sup>3</sup><https://docs.ansible.com/ansible-tower/3.8.4/html/>

- ! Όπως αναφέρθηκε παραπάνω, οι αλληλεπιδράσεις γίνονται μέσω HTTPS/REST API.
- ! Επίσης είναι σε θέση να μπορούν να λειτουργούν ομαλά ώστε να διαχειρίζονται την φυσική και εικονική υποδομή, αλλά και κεντρικούς υπολογιστές.
- **Command Flow [Ροή εντολών]** : Είναι σημαντικό σε αυτό το σημείο να αναφέρουμε και πως λειτουργεί η αρχιτεκτονική του Ansible. Πρακτικά με το Ansible η διαχείριση των πόρων υποδομής επιτυγχάνεται μέσω συνδέσεων SSH που δημιουργούνται μέσω του κόμβου όπου λειτουργεί αυτό το εργαλείο διαχείρισης διαμόρφωσης όπως φαίνεται και στο σχήμα 25.

**NOTE** Η αρχιτεκτονική επιτρέπει την κλιμάκωση και την εκτέλεση σεναρίων σε απομακρυσμένο υλικό, την αφαίρεση εκτελεσμένων σεναρίων και εγκατάσταση βιβλιοθηκών Python. Ομοίως, η διαμόρφωση μπορεί να αναπτυχθεί και να τελειώσει σε απομακρυσμένο υλικό μέσω REST API.



Εικόνα 25. Αρχιτεκτονική Ansible

## Ποή εργασίας του Ansible

Ενώ είδαμε και την αρχιτεκτονική του Ansible είναι αναγκαίο να δείξουμε και να αποτυπώσουμε πως λειτουργεί η ροή εργασίας<sup>4</sup> ενός σεναρίου Ansible [17]:

- **Installation and Set up [Εγκατάσταση και ρύθμιση]** : Πρακτικά, το Ansible γίνεται εγκατάσταση σε έναν κεντρικό κόμβο ελέγχου ή σε έναν αποκλειστικό διακομιστή. Το μηχάνημα αυτό, είναι ο υπεύθυνος για να ξεκινήσουν όλες οι εργασίες αυτοματισμού.
- **Creating Playbooks [Δημιουργία Playbooks]** : Τα Playbooks όπως είπαμε, δημιουργούνται σε **YAML** αρχεία και ορίζουν εργασίες που θα εκτελεστούν στους διαχειριζόμενους κόμβους. Παράλληλα, δημιουργούμε ρόλους και εργασίες ώστε να οργανώσουμε τις διεργασίες που πρέπει να γίνουν σε ρόλους, όπου κάθε ρόλος μπορεί να περιλαμβάνει εργασίες, πρότυπα, αρχεία και μεταβλητές. Και τέλος αρχικοποιούνται οι μεταβλητές και διάφορα templates ώστε να μπορέσουμε να αποθηκεύσουμε τιμές που ενδέχεται να αλλάξουν.
- **Executing Playbooks [Εκτέλεση Playbooks]** : Μέσω της εντολής **ansible-playbook** εκτελούνται τα playbooks που έχουμε δημιουργήσει. Και είναι σημαντικό να αναφερθεί πως η εντολή μπορεί να έχει επιλογές για τον καθορισμό της εκτέλεσης σε συγκεκριμένους υπολογιστές.

Ένα παράδειγμα ενός Playbook σε YAML αρχείο μπορεί να είναι το εξής:

Πίνακας 2. Παράδειγμα Playbook σε YAML αρχείο

| nginx.yml   |
|---|
| <pre>- name: Configure web servers   hosts: webservers   become: yes   tasks:     - name: Install Nginx       apt:         name: nginx         state: present      - name: Ensure Nginx <b>is</b> running       service:         name: nginx         state: started</pre> |

<sup>4</sup><https://www.ansible.com/how-ansible-works/>

#### 4.5.4 Pulumi



#### Εισαγωγή στο Pulumi

Συνεχίζοντας με τα Infrastructure as Code εργαλεία, και έχοντας αναλύσει το Ansible και το Terraform που είναι από τα πιο ευέλικτα εργαλεία για την διαμόρφωση της υποδομής, είναι σημαντικό να αναφερθούμε και να αναλύσουμε και ένα άλλο εργαλείο που ονομάζεται **Pulumi**<sup>5</sup>. Το Pulumi έκανε την εμφάνιση του το 2018 και είναι επίσης ένα σύγχρονο IaC εργαλείο που επιτρέπει στους προγραμματιστές να ορίζουν και να διαχειρίζονται την υποδομή cloud χρησιμοποιώντας διάφορες γλώσσες προγραμματισμού.

Σε αντίθεση με τα παραδοσιακά εργαλεία IaC που κάνουν χρήση γλώσσες για συγκεκριμένους τομείς DSL, YAML/JSON, το Pulumi επιτρέπει στους προγραμματιστές να γράφουν τον κώδικα υποδομής σε γλώσσες όπως η Python, Node.js, .NET. Με αυτή την δυνατότητα του εργαλείου, οι προγραμματιστές είναι σε θέση να αξιοποιήσουν τις υπάρχουσες προγραμματιστικές τους δεξιότητες για τη διαχείριση της υποδομής και την ανάπτυξη των συστημάτων λογισμικού, καθιστώντας την πιο αποτελεσματική και επεκτάσιμη. Παρακάτω θα αναλύσουμε περισσότερο τα βασικά χαρακτηριστικά του Pulumi, τα πλεονεκτήματα που μπορεί να έχει, ώστε να μπορέσουμε να έχουμε μία ευρεία εικόνα για αυτό το IaC εργαλείο.

#### Βασικά χαρακτηριστικά και πλεονεκτήματα του Pulumi

Σε αυτή την υποενότητα είναι σημαντικό να αποτυπωθούν τα κύρια χαρακτηριστικά που έχει το IaC εργαλείο Pulumi [18]:

- **Multi-Programming Language :** Ένα από τα βασικά χαρακτηριστικά που έχει το Pulumi είναι ότι δίνει την δυνατότητα σε ομάδες προγραμματιστών να επιλέξουν εκείνοι ποια γλώσσα προγραμματισμού θα χρησιμοποιήσουν. Το Pulumi υποστηρίζει: Python, Java, Javascript, Go, C-sharp, YAML.
- **Cloud and On-Premises Support :** Ένα σημαντικό χαρακτηριστικό που έχει είναι ότι μπορεί να υποστηρίξει διάφορα περιβάλλοντα cloud, όπως AWS, Azure, GCP. Πιο συγκεκριμένα, το Pulumi υποστηρίζει πάνω από 150 από τους κορυφαίους παρόχους cloud και σύγχρονες προσφορές cloud SaaS, όπως το Amazon Web Services, Microsoft Azure, Google Cloud, Kubernetes, Autho, CloudFlare, DataDog, Docker, κ.α. Επίσης, έχει την δυνατότητα υποστήριξης σε τεχνολογίες cloud, όπως το Kubernetes και υποστηρίζει προηγμένα σενάρια ανάπτυξης.

<sup>5</sup><https://www.pulumi.com/>

- **Third-Party CI/CD Tools Support [Υποστήριξη εργαλείων CI/CD τρίτων]** : Το Pulumi μπορεί εύκολα να ενσωματώνεται με υπάρχοντα εργαλεία και υπηρεσίες, όπως CI/CD πρακτικές και πλατφόρμες παρακολούθησης. Πιο συγκεκριμένα, μπορεί να ενσωματώνεται με το AWS Code Services, Azure Code Services, Azure DevOps, CircleCI, GitHub Actions, GitLab Pipelines, Google Cloud Build, Jenkins, Octopus Deploy, JetBrains TeamCity, Spinnaker και Travis.
- **State Management [Διαχειριστής κατάστασης]** : το Pulumi έχει και αυτό, - όπως έχουμε αναφέρει και για το Terraform-, δυνατότητα να παρέχει διαχείριση κατάστασης για να εξασφαλίζει την αξιοπιστία σε διαφορετικά cloud περιβάλλοντα. Σε αντίθεση με το Terraform, με το Pulumi μπορούμε να χρησιμοποιήσουμε γλώσσες γενικού σκοπού για να δοθεί η επιθυμητή κατάσταση [18]. Το Pulumi έχει την δυνατότητα και μέσα από το Pulumi Cloud να δημιουργήσεις την υποδομή και να θέσεις ένα ομαδικό περιβάλλον που θα μπορούν εκεί οι ομάδες DevOps να λειτουργήσουν με συνεργασία.
- **Scalability [Επεκτασιμότητα]** : Επίσης σημαντικό χαρακτηριστικό είναι πως έχει σχεδιαστεί για να χειρίζεται σύνθετες και μεγάλης κλίμακας διαμορφώσεις υποδομής. Αυτό, δημιουργεί την δυνατότητα στις ομάδες πληροφορικής να μπορούν να το χρησιμοποιούν καθώς παρέχει ευλεξία και επεκτασιμότητα, ανάλογα τα σενάρια.
- **Secret Management [Διαχείριση Μυστικών]** : το Pulumi είναι σε θέση να διαχειριστεί με αποτελεσματικότητα την ασφαλή επικοινωνία και χειρισμό των δεδομένων μέσω κρυπτογράφησης και ασφαλών μηχανισμών ελέγχου ταυτότητας. Έτσι, μπορεί να αποθηκεύει αρχεία κατάστασης με ασφάλεια, υποστηρίζει την κρυπτογράφηση εναίσθητων τιμών (π.χ κωδικούς πρόσβασης βάσης δεδομένων, διακριτικά SaaS, αρχεία διαπιστευτηρίων) ως μυστικά για επιπλέον προστασία [18].
- **Documentation and Community Support [Τεκμηρίωση και Υποστήριξη Κοινότητας]** : Δίνει την δυνατότητα στις ομάδες που θα το χρησιμοποιήσουν να υπάρχει σωστή τεκτηρίωση και μια κοινότητα η οποία είναι σε θέση να παρέχει άμεση βοήθεια στους χρήστες, ώστε να μπορέσουν να μάθουν και να αντιμετωπίζουν διάφορα προβλήματα.

## Προκλήσεις του Pulumi

Σε αυτή την υποενότητα πριν προχωρήσουμε με την αποτύπωση της αρχιτεκτονική του και το πώς δουλεύει, είναι σημαντικό να αναφέρουμε κάποιες προκλήσεις που μπορεί να έχει κατά την υιοθέτηση του:

- **Imperative vs Declarative Models** : Το Pulumi Κάνει χρήση επικακτικής γλώσσας, και αυτό ίσως δημιουργήσει περίπλοκες διαμορφώσεις, ειδικά σε έργα μεγάλης κλίμακας. Σε αντίθεση με το Terraform που χρησιμοποιεί το HCL , το μοντέλο του Pulumi μπορεί να οδηγήσεις σε κώδικα πιο επιρρεπή σε σφάλματα [19].
- **Inconsistent Abstraction Levels [Ασυνεπή επίπεδα αφαίρεσης]** : Επίσης μια πρόκληση του Pulumi είναι πως η διαχείριση των επιπέδων αφαίρεσης σε διαφορετικούς παρόχους cloud, μπορεί να καταστεί δύσκολη [19]. Αν αναλογιστούμε πως για να υπάρξει παροχή ενός Kubernetes Cluster χρειάζεται διαφορετική παραμετροποίηση σε κάθε cloud περιβάλλον, καταλαβαίνουμε πως μπορεί να οδηγήσει σε ζητήματα στο επίπεδο αφαίρεσης. Συνεπώς οι προγραμματιστές θα χρειαστεί να κατανοήσουν αρκετά τις αλλαγές που πρέπει να κάνουν στους διάφορους cloud providers για να μπορέσει η υποδομή να συνεργαστεί άρτια με το Pulumi.

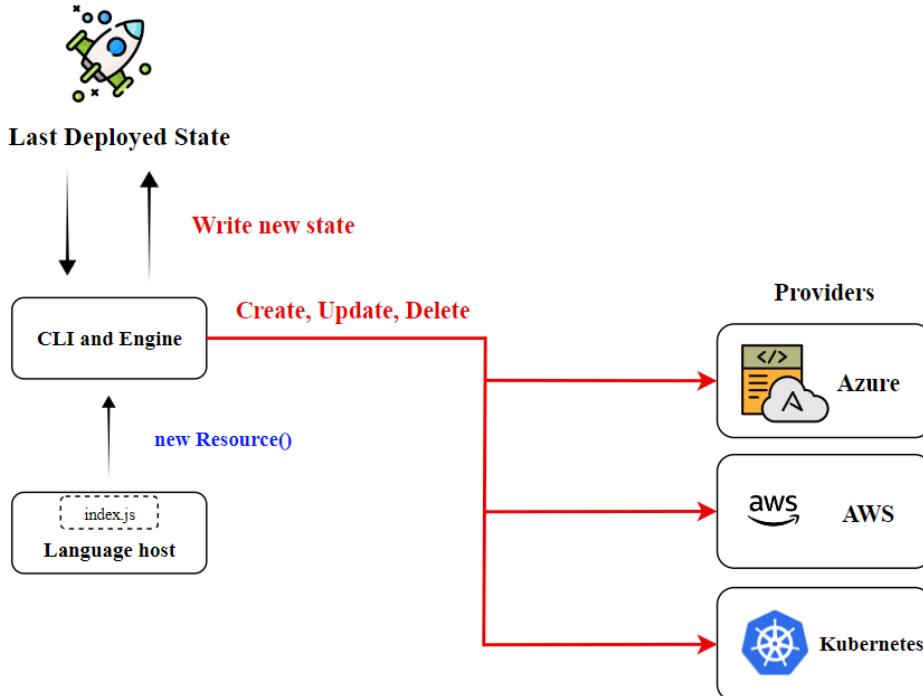
- **Deviation from Standard IaC Practice [Απόκλιση από την τυπική πρακτική IaC]** : Το Pulumi όπως είπαμε είναι ένα εργαλείο στο οποίο μπορεί ο καθένας να χρησιμοποιήσει την γλώσσα προγραμματισμού που θέλει. Αυτή η ευελιξία που παρέχει, σημαίνει πως αποκλίνει [19] από το δηλωτικό πρότυπο, και αυτό μπορεί να δημιουργήσει προκλήσεις για ομάδες που κάνουν αλλαγή για παράδειγμα από το Terraform στο Pulumi.

### Αρχιτεκτονική και ροή εργασίας του Pulumi

Σε αυτό το σημείο είναι πολύ σημαντικό να δώσουμε μια σωστή δομή στο κείμενο μας, και να αναφέρουμε τον τρόπο που λειτουργεί το Pulumi, δηλαδή την αρχιτεκτονική που έχει. Συνοπτικά, το εργαλείο συγκρίνει την επιθυμητή κατάσταση με την τρέχουσα κατάσταση της υποδομής [20], προσδιορίζοντας τους απαραίτητους πόρους και ενημερώνει την κατάσταση με πληροφορίες σχετικά με τους παρεχόμενους πόρους και τις εκρεμμείς λειτουργίες. Παρακάτω, θα περιγράψουμε καθένα από αυτά τα συστατικά και θα δούμε πώς ταιριάζουν όλα μαζί κατά τη διάρκεια μιας επίκλησης **pulumi up**, όπως φαίνεται στο σχήμα 26.

- **Deployment engine [Μηχανή Ανάπτυξης]** : Ο κινητήρας ανάπτυξης είναι ένα κρίσιμο εργαλείο στο Pulumi CLI που οδηγεί την τρέχουσα κατάσταση μιας υποδομής στην επιθυμητή κατάσταση. Ελέγχει εάν ένας πόρος έχει δημιουργηθεί προηγούμενως ή όχι, και εάν έχει δημιουργηθεί, συνεργάζεται με έναν πάροχο πόρων για να προσδιορίσει εάν έχουν συμβεί αλλαγές [20]. Εάν προκύψουν αλλαγές, ο κινητήρας αποφασίζει εάν θα ενημερώσει τον πόρο ή θα τον αντικαταστήσει με μια νέα έκδοση. Μόλις εκτελεστεί το πρόγραμμα, η μηχανή αναζητά υπάρχοντες πόρους που δεν είχαν καταχωρηθεί και προγραμματίζει τη διαγραφή τους. Ο κινητήρας ανάπτυξης είναι ενσωματωμένος στο πρόγραμμα.
- **Resource providers [Πάροχοι πόρων]** : Ένας πάροχος πόρων αποτελείται από δύο στοιχεία: ένα plugin πόρων, ένα δυαδικό που χρησιμοποιείται από τη μηχανή ανάπτυξης και ένα SDK που παρέχει δεσμεύσεις για κάθε τύπο πόρων. Αυτά τα πρόσθετα αποθηκεύονται στην κρυφή μνήμη των προσθηκών και η διαχείριση γίνεται χρησιμοποιώντας το σύνολο εντολών της προσθήκης **pulumi**.
- **Creation and Deletion Order [Εντολή δημιουργίας και διαγραφής]** : Όπως παρατηρούμε και στο σχήμα 26 το Pulumi εκτελεί λειτουργίες πόρων παράλληλα, αλλά αναγνωρίζει τις εξαρτήσεις μεταξύ των πόρων. Εάν μια έξοδος από ένα resource εισέρχεται σε έναν άλλο, ο κινητήρας καταγράφει την εξάρτηση μεταξύ τους στην κατάσταση, η οποία μπορεί να επαυξηθεί χρησιμοποιώντας την επιλογή πόρου. Εάν ένας πόρος χρειάζεται αντικατάσταση, το Pulumi δημιουργεί ένα νέο αντίγραφο πριν καταστρέψει το παλιό, επιτρέποντας ενημερώσεις υποδομής χωρίς διακοπές λειτουργίας. Η απενεργοποίηση της αυτόματης ονομασίας για έναν πόρο τον αντιμετωπίζει αυτόματα σαν να είχε επισημανθεί ως `deleteBeforeReplace` [20].

- **Declarative and Imperative Approach [Δηλωτική και επιτακτική προσέγγιση]**: Το Pulumi, όπως έχουμε αναφέρει είναι μια ισχυρή λύση κώδικα που επιτρέπει στους χρήστες να συντάξουν την υποδομή τους στη γλώσσα προγραμματισμού που προτιμούν. Ξεκινά τόσο τον κεντρικό υπολογιστή γλώσσας για την επιλεγμένη γλώσσα όσο και τους απαιτούμενους παρόχους μέσω της μηχανής Pulumi [20]. Οι πάροχοι, που συντονίζονται από τον Pulumi engine, εκτελούν την πραγματική δημιουργία, τροποποίηση ή διαγραφή της υποδομής. Η αρχιτεκτονική περιλαμβάνει έναν κεντρικό υπολογιστή επιτακτικής γλώσσας, έναν μηχανισμό δήλωσης και παρόχους.



Εικόνα 26. Αρχιτεκτονική Pulumi

#### 4.5.5 Chef



Σε αυτή την υποενότητα θα αναφέρουμε ένα άλλο IaC εργαλείο, το Chef<sup>6</sup>. Το Chef είναι ένα εργαλείο ανοιχτού κώδικα IaC που αναπτύχθηκε από την Opscode που αυτοματοποιεί την παροχή υποδομής, τη διαχείριση διαμόρφωσης και την ανάπτυξη εφαρμογών. Ακολουθεί μια δηλωτική προσέγγιση, επιτρέποντας στους χρήστες να ορίσουν την κατάσταση της υποδομής τους χρησιμοποιώντας κώδικα [17]. Το ισχυρό οικοσύστημα του Chef προσφέρει λειτουργικότητα σε περιβάλλοντα εσωτερικού χώρου, cloud και hybrid περιβάλλοντα, καθιστώντας το μια ευέλικτη επιλογή για οργανισμούς που επιδιώκουν να εξορθολογίσουν τις διαδικασίες διαχείρισης της υποδομής τους.

##### **Βασικά χαρακτηριστικά και πλεονεκτήματα του Chef**

Είναι σημαντικό να δώσουμε μια σαφή αποτύπωση και εδώ, για τα χαρακτηριστικά που έχει το εργαλείο Chef:

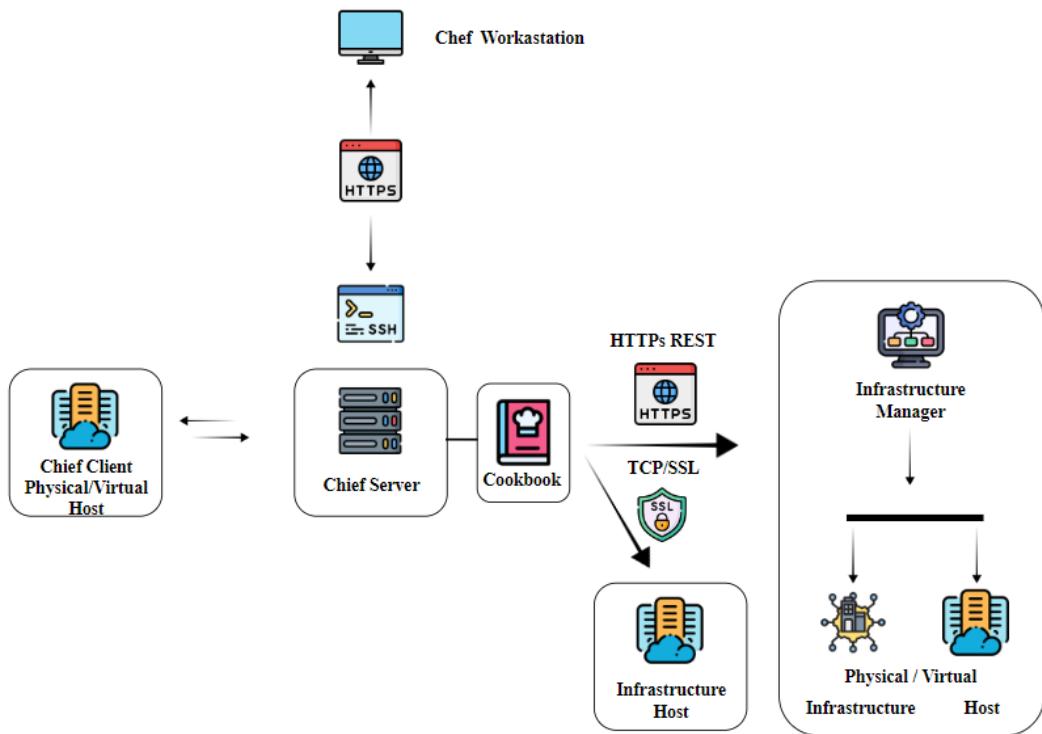
- **Declarative Approach [Δηλωτική προσέγγιση]** : Το εργαλείο Chef χρησιμοποιεί μια δηλωτική γλώσσα για να ορίσει την επιθυμητή κατάσταση μιας υποδομής. Έτσι με αυτόν τον τρόπο, απλοποιεί και αυτό τη διαχείριση της υποδομής σε πολύπλοκα περιβάλλοντα.
- **Extensible Ecosystem [Επεκτάσιμο Οικοσύστημα]** : Είναι επίσης σημαντικό να αναφέρουμε ότι, το εκτεταμένο οικοσύστημα των "cookbooks" και των πόρων του Chef επιτρέπει τη γρήγορη ανάπτυξη και διαμόρφωση των εφαρμογών [17]. Ο Chef πρακτικά λειτουργεί, αναλύοντας τη διαδικασία διαμόρφωσης σε μικρά επαναχρησιμοποιήσιμα εξαρτήματα που ονομάζονται "recipes". Με αυτόν τον τρόπο, οι χρήστες μπορούν να μοιράζονται τα "cookbooks" για να δημιουργήσουν τη δική τους υποδομή.
- **Enterprise-Grade Features** : Το εργαλείο Chef μπορεί να προσφέρει χαρακτηριστικά σε επίπεδο επιχείρησης, όπως έλεγχο πρόσβασης βάσει ρόλων [17], διαχείριση συμμόρφωσης και ενσωμάτωση με διάφορα εργαλεία παρακολούθησης και αναφοράς, καθιστώντας τον κατάλληλη επιλογή για πολύπλοκα περιβάλλοντα.
- **Scalability [Επεκτασιμότητα]** : Η κατανεμημένη αρχιτεκτονική του Chef και το μοντέλο ασύγχρονης εκτέλεσης συμβάλλουν στην επεκτασιμότητα και την απόδοσή του [17]. Έχει κατασκευαστεί ώστε να διαχειρίζεται μεγάλης κλίμακας υποδομές, καθώς μπορεί να διαχειριστεί πολλούς κόμβους αποτελεσματικά.
- **Multi-Platform Support** : Υποστηρίζει πολλαπλές πλατφόρμες, συμπεριλαμβανομένων των Linux, Windows και MacOS, καθιστώντας το κατάλληλο για τη διαχείριση ετερογενών περιβαλλόντων.

<sup>6</sup><https://www.chef.io/>

## Αρχιτεκτονική του Chef

Συνεχίζοντας την ανάλυση του Chef ως εργαλείο IaC, πάμε τώρα να δούμε πιο διεξοδικά την αρχιτεκτονική του εργαλείου, να αναλύσουμε σύντομα το πώς λειτουργεί κατά την ανάπτυξη υποδομής, όπως φαίνεται στο σχήμα 27.

- **Chef Workstation :** Ο σταθμός εργασίας Chef, είναι πρακτικά ο διαχειριστής ή ένας προγραμματιστής του συστήματος, όπου γράφει και δοκιμάζει τον κώδικα υποδομής, γνωστό και ως "cookbooks"[17]. Αυτά τα βιβλία μαγειρικής ορίζουν επιθυμητή κατάσταση και διαμόρφωσης της υποδομής. Σημαντικό δε, είναι πως η επικοινωνία από το chef workstation με τον "Chief Server" γίνεται μέσω των πρωτοκόλλων HTTPS ή SSH.
- **Chief Server :** Μέσα στην αρχιτεκτονική του Chef, υπάρχει ο Chef Server. Πρακτικά λειτουργεί ως κεντρικός κόμβος για την διαχείριση της κατάστασης της υποδομής [17]. Αυτό που κάνει, είναι να αποθηκεύει τα cookbooks και τα meta data σχετικά με τους clients μέσα στο περιβάλλον. Συνεπώς, όταν ένας Chief client χρειάζεται να διαμορφώσῃ την υποδομή του, κάνει ανάκτηση των κατάλληλων cookbooks από το Chief Server. Σημαντικό να αναφέρουμε είναι πως για την επικοινωνία χρησιμοποιεί HTTPS/REST και TCP/SSL πρωτόκολλα, για να μπορεί να πραγματοποιήσει την διανομή των cookbooks και των polices (πολιτικών) που έχει.
- **Chief Client :** Σε αυτό το σημείο είναι σημαντικό να αναφέρουμε πως το Cheif Client εκτελείται σε κάθε κόμβο, είτε εικονικό είτε φυσικό κεντρικό υπολογιστή, που θα χρειαστεί να διαχειριστεί. Η ενέργεια που κάνει, είναι να ελέγχει περιοδικά τον Chef Server, για πιθανές ενημερώσεις και μετέπειτα εφαρμόζει τις διαμορφώσεις που ορίζονται στα cookbooks [17]. Έτσι είναι σίγουρος πως η κατάσταση του κόμβου ταιριάζει απόλυτα με την επιθυμητή κατάσταση που ορίζεται στα cookbooks.
- **Cookbook :** Προηγουμένως, κάναμε αρκετές αναφορές στα "βιβλία μαγειρικής" που ως χαρακτηριστικό το Chef εργαλείο. Καταλαβαίνουμε ότι είναι μια θεμελιώδης μονάδα διαμόρφωσης και διανομής πολιτικής στο Chef. Πρακτικά περιέχει "συνταγές", όπου καθορίζουν τον τρόπο διαμόρφωσης των πόρων [17]. Τα cookbooks αποθηκεύονται στον Chef Server και λαμβάνονται από τους Chef Client κατά παραγγελία για να διασφαλιστεί ότι ακολουθούν την καθορισμένη διαμόρφωση.
- **Infrastructure / Host :** Στην ουσία, έχει να κάνει με τους πραγματικούς φυσικούς ή εικονικούς διακομιστές που διαχειρίζονται. Εκεί πρακτικά εκτελούνται όλες οι ενέργειες ενός Chef Client, ο οποίος εφαρμόζει τις διαμορφώσεις.
- **Infrastructure Manager :** Όπως παρατηρούμε και στο σχήμα 27 ο διαχειριστής συνήθως είναι ένας πάροχος cloud ή ένα virtual machine (εικονική μηχανή), και παρέχει τους πόρους που χρειάζεται να διαχειριστεί ο Chef.



Εικόνα 27. Αρχιτεκτονική Chef

#### 4.5.6 Puppet



Σε αυτό το σημείο θα δώσουμε μια σαφή εικόνα και του εργαλείου Puppet<sup>7</sup>, όπου και αυτό είναι ένα δημοφιλές Infrastructure as Code εργαλείο, που αυτοματοποιεί τη διαχείριση και τη διαμόρφωση μεγάλης κλίμακας υποδομής. Αναφορικά, έχει αναπτυχθεί από την Puppet, Inc., και επιτρέπει τόσο στους διαχειριστές συστημάτων, όσο και στις ομάδες DevOps να ορίσουν ποια θα είναι η επιθυμητή κατάσταση της υποδομής τους, χρησιμοποιώντας δηλωτική γλώσσα. Παρακάτω θα γίνει μια ανάλυση ως προς τα βασικά χαρακτηριστικά και πλεονεκτήματα που έχει, αλλά και ποια είναι η αρχιτεκτονική του και τα στοιχεία εκείνα που συνθέτουν αυτό το εργαλείο.

#### Βασικά χαρακτηριστικά και πλεονεκτήματα του Puppet

Συνοπτικά για το Puppet εργαλείο, μπορούμε να αναφέρουμε τα εξής:

- **Idempotency [Ανικανότητα]** : Όπως έχουμε δει και στα πλεονεκτήματα του Ansible 4.5.3, το βασικό του χαρακτηριστικό είναι η "ανικανότητα", η οποία επιτρέπει την επαναλαμβανόμενη εφαρμογή κώδικα για να εγγυηθεί την επιθυμητή κατάσταση σε ένα σύστημα [21], διασφαλίζοντας το ίδιο αποτέλεσμα κάθε φορά. Αυτή η δυνατότητα επιτρέπει στο Puppet να λειτουργεί συνεχώς, διασφαλίζοντας ότι η κατάσταση της υποδομής ταιριάζει με την επιθυμητή κατάσταση. Εάν αλλάξει μια κατάσταση συστήματος, τότε το Puppet μπορεί να ενημερώσει αυτόματα ολόκληρη την υποδομή.
- **Reporting [Αναφορά]** : Το Puppet προσφέρει ισχυρές δυνατότητες αναφοράς που μπορεί να παρακολουθεί την κατάσταση των κόμβων και την εφαρμογή των διαμορφώσεων. Έτσι με αυτόν τον τρόπο βοηθάει στην διασφάλιση της συμμόρφωσης με τις καθορισμένες πολιτικές και τα πρότυπα, παρέχοντας με αυτόν τον τρόπο λεπτομερείς αναφορές και ελέγχους.

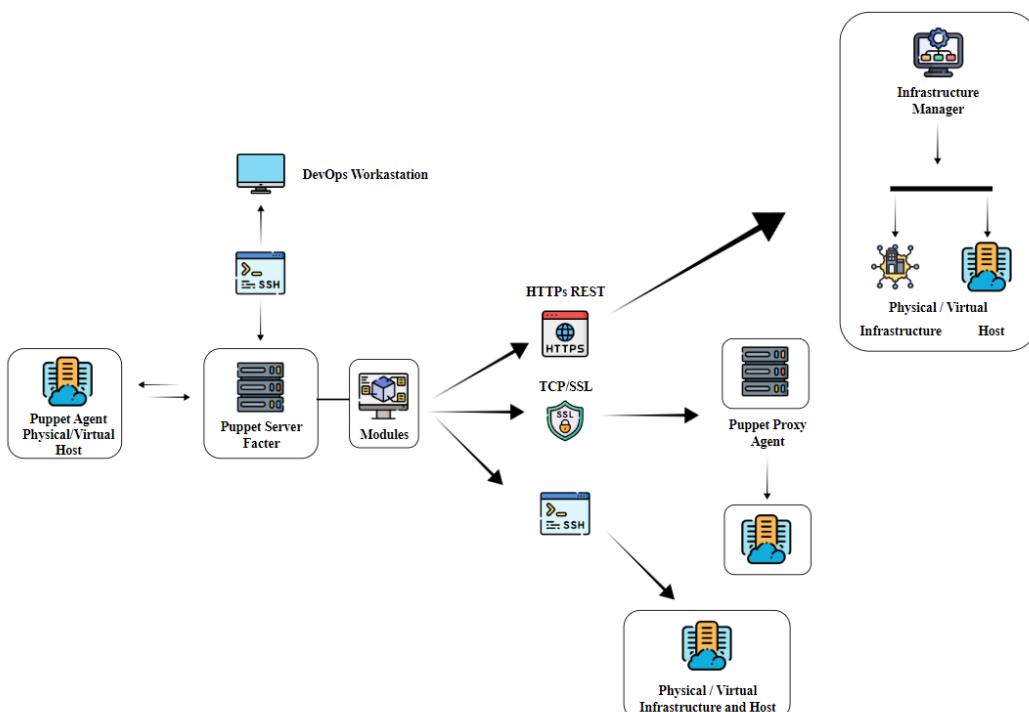
#### Αρχιτεκτονική του Puppet

Σε αυτή την υποενότητα θα δούμε πώς είναι η αρχιτεκτονική του Puppet και ποια είναι τα στοιχεία που την συνθέτουν :

- **Puppet Server [Puppet Master]** : Αυτό περιλαμβάνει την υπηρεσία συλλογής πληροφοριών που ονομάζεται Facter. Παράλληλα περιέχει και την PuppetDB για να μπορεί να αποθηκεύει συμβάντα, καταλόγους κόμβων, διαμορφώσεων και ιστορικών δεδομένων [17].

<sup>7</sup><https://www.puppet.com/>

- **Client-Server Architecture [Αρχιτεκτονική πελάτη-διακομιστή]** : To Client (Agent) έχει εγκατασταθεί και ρυθμιστεί με ασφάλεια σε μηχανές-στόχους [17]. Τόσο ο client όσο και ο server ελέγχουν ο ένας τον άλλον χρησιμοποιώντας πιστοποιητικά αυτο-υπογεγραμμένα.
- **Functionality [Λειτουργικότητα]** : Επίσης σημαντικό είναι να δούμε την λειτουργικότητα του. Πρακτικά ο πράκτορας [agent], συλλέγει συμβάντα υπό τον έλεγχο της υπηρεσίας Facter. Εφαρμόζει τις αλλαγές διαμόρφωσης σύμφωνα με τις οδηγίες του Puppet Server. Και οι λειτουργικές μονάδες επιτρέπουν την συνδεσμότητα για Cloud API και υλικό που δεν μπορεί να εκτελέσει ένας agent.
- **User Interactions [Αλληλεπιδράσεις χρηστών]** : Χρησιμοποιεί το SSH και την γραμμή εντολών, ώστε να γίνεται η αλληλεπίδραση των χρηστών.
- **Agent Installation - Configuration Agent [Εγκατάσταση - Διαμόρφωση πράκτορα]** : Το εργαλείο Puppet Agent γίνεται εγκατάσταση στους clients μετά την λειτουργία του διακομιστή. Επίσης οι πληροφορίες της διαμόρφωσης του server αποθηκεύονται στο αρχείο Puppet.conf στον υπολογιστή-πελάτη. Έπειτα ο διακομιστής συλλέγει αυτές τις πληροφορίες από τον πελάτη και ενημερώνει τη κατάσταση του. [17], με αλλαγές διαμόρφωσης. Σε αρχεία δήλωσης με επέκτηση .pp υπάρχουν οι ειδοποιήσεις διαμορφωμένες, γραμμένα σε δηλωτική γλώσσα που μοιάζει με Ruby.



Εικόνα 28. Αρχιτεκτονική Puppet

## 4.6 Infrastructure as Code εργαλεία από Cloud Providers

Έχοντας δώσει μια σαφή εικόνα και ανάλυση για τα πιο δημοφιλή Infrastructure as Code εργαλεία στις προηγούμενες ενότητες, είναι εξίσου σημαντικό να δούμε και κάποια IaC εργαλεία που μπορούν να προσφέρουν οι πάροχοι cloud, καθώς είναι σε θέση να προσφέρουν δικές τους λύσεις IaC. Αυτά τα εργαλεία Iac, επιτρέπουν επίσης σε ομάδες και οργανισμούς να διαχειρίζονται την υποδομή τους σε αυτά τα περιβάλλοντα cloud με αυτοματοποιημένο τρόπο.

### 4.6.1 AWS CloudFormation



Ένα από τα Iac εργαλεία ονομάζεται AWS CloudFormation και παρέχεται από την **Amazon Web Services (AWS)** που επιτρέπει την αυτόματη δημιουργία και διαχείριση πόρων υποδομής. Επιτρέπει στους χρήστες να ορίσουν τις δικές τους υποδομές AWS, με αρχεία σε μορφή JSON ή YAML.

Αυτά τα αρχεία προτύπων, βοηθούν αρκετά τους χρήστες να προσδιορίσουν τους πόρους, τις σχέσεις και τις διαμορφώσεις τους για το σενάριο που θέλουν να πετύχουν. Το CloudFormation επιτρέπει στους χρήστες να αναπτύξουν την υποδομή τους γρήγορα και με επαναλαμβανόμενο τρόπο [17]. Έτσι αυτά τα πρότυπα μπορούν να περιλαμβάνουν πόρους όπως εικονικά ιδιωτικά σύννεφα (VPC), διακομιστές, βάσεις δεδομένων, και άλλες υπηρεσίες AWS.

#### Αρχιτεκτονική του AWS CloudFormation

Σημαντικό είναι να δούμε την αρχιτεκτονική του AWS CloudFormation που φαίνεται στο σχήμα 29, ώστε να έχουμε μια σαφή εικόνα για το πως λειτουργεί ως εργαλείο IaC.

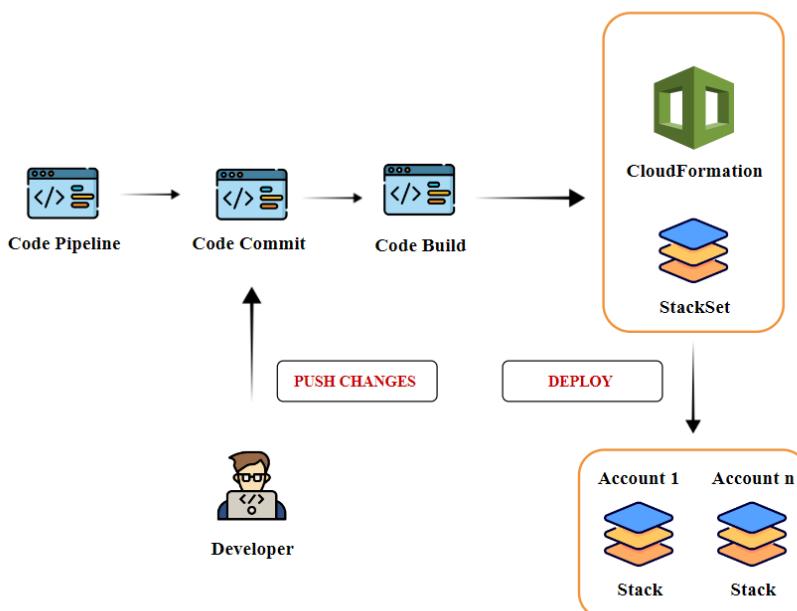
#### Βασικά χαρακτηριστικά της αρχιτεκτονικής του AWS CloudFormation

- **Developer :** Ξεκινάει όλες τις διαδικασίες για τις αλλαγές που πρέπει να γίνουν ή να προστεθούν.
- **Code Pipeline :** Με το code pipeline, δίνεται η δυνατότητα να αυτοματοποιήσει τις φάσεις κατασκευής, δοκιμής, και ανάπτυξης των συστημάτων της υποδομής, και παράλληλα συντονίζει με τον βέλτιστο τρόπο όλα τα στάδια του κύκλου ζωής της ανάπτυξης [17].
- **Code Commit :** Οι προγραμματιστές είναι σε θέση με τη υπηρεσία αυτή να προωθούν τις αλλαγές στον κώδικα τους και ενεργοποιείται ο αγωγός [pipeline] [17] για να ξεκινήσει η διαδικασία κατασκευής και ανάπτυξης.

- **Code Build** : Σε αυτή την φάση η υπηρεσία είναι σε θέση να μεταγλωττίσει το πηγαίο κώδικα, να εκτελέσει τις απαραίτητες δοκιμές και να παράγει πακέτα λογισμικού έτοιμα για την ανάπτυξη [17].
- **CloudFormation Stackset** : Ένα άλλο χαρακτηριστικό που έχει είναι το σύνολο των στοιβών CloudFormation που μπορούν να αναπτυχθούν σε πολλούς λογαριασμούς και περιοχές στο AWS, καθώς μπορεί να διαχειρίζεται την ανάπτυξη και την ενημέρωση των στοιβών σε διάφορα περιβάλλοντα.

### Διαδικασία ανάπτυξης στο AWS CloudFormation

- **Pushing Changes** : Στην διαδικασία της ανάπτυξης ο προγραμματιστής είναι σε θέση να προωθήσει τις αλλαγές στην υπηρεσία Code Commit.
- **Pipeline Execution** : Η υπηρεσία Code Pipeline ανιχνεύει τις αλλαγές και ενεργοποιεί την διαδικασία κατασκευής. Παράλληλα η υπηρεσία Code Build μεταγλωττίζει τον κώδικα και τον προετοιμάζει για την ανάπτυξη.
- **Deployment with AWS CloudFormation** : Σε αυτό το στάδιο, μόλις η κατασκευή είναι επιτυχής, το CloudFormation StackSet ξεκινάει την ανάπτυξη της υποδομής. Η υποδομή ορίζεται σε ένα πρότυπο CloudFormation, το οποίο εφαρμόζεται για την δημιουργία ή την ενημέρωση των στοιβών.
- **Multi-Account Deployment** : Τέλος το CloudFormation StackSet αναπτύσσει τις καθορισμένες στοιβες υποδομής σε πολλούς λογαριασμούς AWS, και εξασφαλίζεται πως η διαμόρφωση της υποδομής μπορεί να γίνει άρτια σε διαφορετικά περιβάλλοντα.



Εικόνα 29. Αρχιτεκτονική AWS CloudFormation

#### 4.6.2 Azure Resource Manager



Ένα άλλο εργαλείο Infrastructure as Code, είναι αυτό που προσφέρει η Microsoft και έχει την ονομασία Azure Resource Manager<sup>8</sup>. Το ARM είναι ικανό να διαχειρίζεται πόρους Azure. Τα πρότυπα (templates) είναι γραμμένα σε JSON [22], μπορούν και ορίζουν επιθυμητή κατάσταση της υποδομής Azure, καθώς επιτρέπονται και εδώ οι επαναλαμβάνομενες αναπτύξης.

#### Πλεονεκτήματα του Azure Resource Manager

Τα οφέλη του Resource Manager με τη Διαχείριση πόρων, είναι τα εξής [22]:

- Μπορούμε να διαχειριστούμε την υποδομή μας μέσω δηλωτικών προτύπων και όχι μέσω σεναρίων.
- Παράλληλα μπορούμε να αναπτύξουμε, να διαχειριστούμε και να παροκολουθήσουμε όλους τους πόρους, αντί να χειρίζομαστε αυτούς τους πόρους μεμονωμένα.
- Επίσης σημαντικό όφελος είναι ότι μπορούμε να ανακατανείμουμε τη λύση σας σε όλη τη διάρκεια του κύκλου ζωής της ανάπτυξης και έτσι με αυτό τον τρόπο, εμπιστοσύνη στους πόρους μας ότι αναπτύσσονται σε συνεπή κατάσταση.
- Παράλληλα μπορούμε να Καθορίσουμε τις εξαρτήσεις μεταξύ των πόρων, ώστε να αναπτύσσονται με τη σωστή σειρά.
- Μπορούμε να εφαρμόζουμε έλεγχο πρόσβασης σε όλες τις υπηρεσίες επειδή ο έλεγχος πρόσβασης βάσει ρόλων Azure (Azure RBAC) είναι εγγενώς ενσωματωμένος στην πλατφόρμα διαχείρισης.
- Χρησιμοποιούμε ετικέτες σε πόρους της υποδομής μας για να οργανώσουμε λογικά όλους τους πόρους στη συνδρομή μας.
- Είναι εύκολα μέσα από την πλατφόρμα του Azure να διευκρινίσουμε τη χρέωση της εταιρίας, προβάλλοντας το κόστος για μια ομάδα πόρων που μοιράζονται την ίδια ετικέτα.

<sup>8</sup><https://learn.microsoft.com/en-us/azure/azure-resource-manager/>

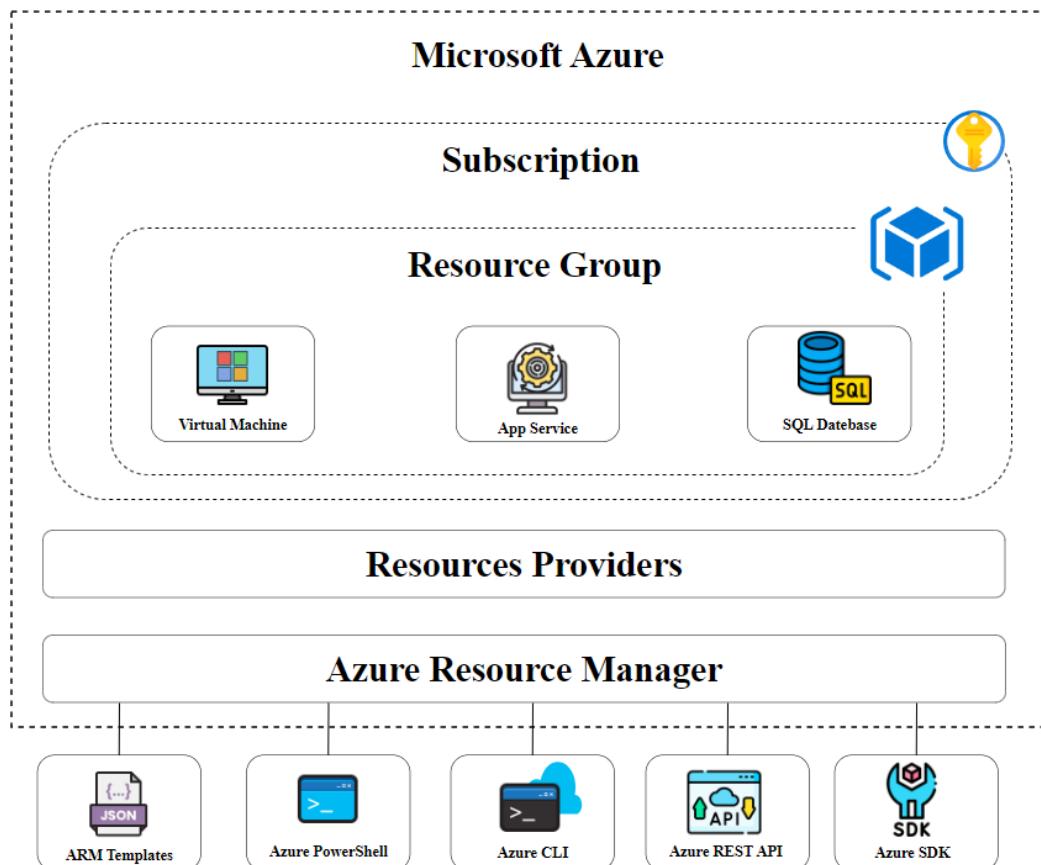
## Αρχιτεκτονική του Azure Resource Manager

Σε αυτή την υποενότητα θα προσπαθήσουμε να δώσουμε μια εικόνα για τα βασικά χαρακτηριστικά που έχει η αρχιτεκτονική του Azure Resource Manager και τις δυνατότητές που έχει κατά την υιοθέτηση του.

### Βασικά χαρακτηριστικά της αρχιτεκτονικής του ARM

- **Resource Group :** Το ARM είναι σε θέση να παρέχει ένα container, μια ομάδα από πόρους που λειτουργούν ως container και περιέχουν σχετικούς πόρους Azure. Επιτρέπουν τη διαχείριση και την ομαδοποίηση των πόρων, όπως είναι τα virtual machines, οι υπηρεσίες εφαρμογών και οι βάσεις δεδομένων, όπου μοιράζονται τον ίδιο κύκλο ζωής.
- **Resource Providers :** Όπως βλέπουμε και στο σχήμα 30 της αρχιτεκτονικής του Azure Resource Manager, πρόκειται για υπηρεσίες που παρέχουν τις διαθέσιμες λειτουργίες για διάφορους πόρους [22].
- **Azure Resource Manager :** Εδώ βρίσκεται το κεντρικό πεδίο διαχείρισης των πόρων. Το ARM είναι η βασική υπηρεσία που παρέχει όλες τις δυνατότητες διαχείρισης και ανάπτυξης για τους πόρους του Azure. Συνεπώς, είναι σε θέση να διαχειρίζεται την ανάπτυξη και την παροκολούθηση αυτών των πόρων μέσω διάφορων μεθόδων, διασφαλίζοντας τον συστηματικό έλεγχο.
- **Tools [Εργαλεία]**
  - ! ARM Templates: είναι συγκεκριμένα πρότυπα που βασίζονται σε JSON και ορίζουν την υποδομή και τη διαμόρφωση των πόρων του Azure.
  - ! Azure PowerShell: είναι η διεπαφή γραμμής εντολών, που χρησιμεύει στην διαχείριση των πόρων, χρησιμοποιώντας σενάρια και εντολές.
  - ! Azure CLI: επίσης ένα εργαλείο γραμμής εντολών
  - ! Azure REST API: βοηθάει στον προγραμματισμό και την διαχείριση των πόρων.
  - ! Azure SDK: είναι το βασικό kit ανάπτυξης των συστημάτων λογισμικού για διάφορες γλώσσες προγραμματισμού μέσω των υπηρεσιών του Azure.

Παρατηρώντας λοιπόν το σχήμα της αρχιτεκτονικής του Azure Resource Manager 30, απεικονίζει την ιεραρχική δομή του ARM, δείχνοντας πώς ομαδοποιούνται οι πόροι μέσα από τις συνδρομές που έχει ενας χρήστης, και πώς αλληλεπιδρά το ARM με αυτούς τους πόρους μέσω εργαλείων και API.



Εικόνα 30. Αρχιτεκτονική Azure Resource Manager

### 4.6.3 Google Cloud Deployment Manager



Το Google Cloud Deployment Manager είναι ένα εργαλείο υποδομής ως κώδικα (IaC) που επιτρέπει στους προγραμματιστές και τις ομάδες DevOps να αυτοματοποιούν τη δημιουργία και τη διαχείριση των πόρων του Google Cloud. Χρησιμοποιεί πρότυπα για τον καθορισμό των πόρων, καθιστώντας την ανάπτυξη επαναλαμβανόμενη και συνεπή. Παρακάτω θα ακολουθήσει μια δομημένη ανάλυση της αρχιτεκτονικής και των βασικών χαρακτηριστικών του.

#### Βασικά χαρακτηριστικά και πλεονεκτήματα του GCDM

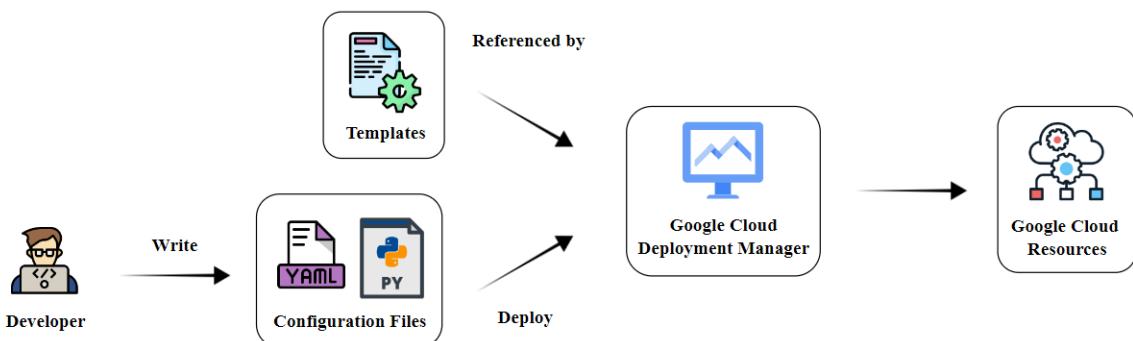
- **Automated Resource Management :** Το Google Cloud Deployment Manager, επιτρέπει την ακριβή και συνεπή δημιουργία, διαχείριση και διαγραφή των πόρων.
- **YAML-Based Configuration :** Η διαμόρφωση για τους πόρους γίνεται κυρίως μέσω αρχείων YAML, καθώς προωθεί και αυτό το εργαλείο IaC την επαναχρησιμοποίηση πόρων για την υποδομή.
- **Integrated With Google Cloud Services :** Είναι δομημένο και σχεδιασμένο με τέτοιο τρόπο, ώστε να μπορεί να λειτουργεί απρόσκοπτα με την ευρεία γκάμα υπηρεσιών του GCP [23].

#### Αρχιτεκτονική του Google Cloud Deployment Manager

Συνεχίζοντας, θα δώσουμε συνοπτικά τον τρόπο που η αρχιτεκτονική του GCDM λειτουργεί με τα χαρακτηριστικά που έχει.

- Παρακτικά οι ομάδες DevOps, ορίζουν την υποδομή τους, κάνοντας χρήση templates γραμμένα σε YAML ή Python, όπως φαίνεται και στο απλό σχήμα του GCDM 31.
- Τα αρχεία διαμόρφωσης που δημιουργούνται μέσα στα templates, παρέχουν συγκεκριμένες τιμές για τις μεταβλητές που ορίζονται. Αυτά τα αρχεία, είναι εκείνα τα οποία ουσιαστικά περιγράφουν λεπτομερώς την επιθυμητή κατάσταση της υποδομής.
- Συνεχίζοντας, για να γίνει η ανάπτυξη του σεναρίου που είναι αποτυπωμένο μέσα στα πρότυπα αρχεία, ξεκινάει η αποστολή τους στο GCDM, όπου αναλαμβάνει να κατανοήσει τις απαιτήσεις που υπάρχουν για την δημιουργία της υποδομής.
- Μετέπειτα, το Deployment Manager αλληλεπιδρά με API του Google Cloud για γίνει η απαραίτητη δημιουργία και διαμόρφωση των καθορισμένων πόρων. Αυτό πιθανόν να περιλαμβάνει τη δημιουργία εικονικών μηχανών, δικτύων, βάσεων δεδομένων και άλλων πόρων cloud. Είναι λοιπόν σε θέση να χειρίζεται τις όποιες εξαρτήσεις πόρων, διασφαλίζοντας πως οι πόροι δημιουργούνται με τη σωστή σειρά.

- Παράλληλα, αν χρειαστεί να γίνουν αλλαγές στις υποδομές το GCDM είναι υπεύθυνο να κάνει αυτές τις αλλαγές στους πόρους που χρειάζεται, χωρίς να γίνει οποιαδήποτε άλλη αλλαγή μέσα στην υποδομή.
- Σημαντικό επίσης είναι πως αν για παράδειγμα μια ενημέρωση για κάποιο λόγο αποτύχει, τότε ο Deployment Manager μπορεί να επαναφέρει τις αλλαγές και να διατηρήσει την επιθυμητή κατάσταση στην υποδομή. Συμπερασματικά λοιπόν, είναι σημαντικό καθώς ελαχιστοποιεί το χρόνο διακοπής λειτουργίας και εξασφαλίζει σταθερότητα κατά την διάρκεια των ενημερώσεων.
- Είναι επίσης σημαντικό να αναφέρουμε πως το Deployment Manager έχει στην διάθεση του κατεγραμμένα λεπτομερή αρχεία και πληροφορίες παρακολούθησης σχετικά με τη διαδικασία ανάπτυξης. Οι ομάδες DevOps είναι σε θέση να παρακολουθούν την κατάσταση και το ιστορικό.
- Τέλος, είναι αναγκαίο να παρέχεται σωστή πρόσβαση μέσα στην υποδομή. Υπάρχει λοιπόν η δυνατότητα, η πρόσβαση στους πόρους που διαχειρίζεται το GCDM να ελέγχεται μέσα από τους Google Cloud Identity και Access Management.



Εικόνα 31. Αρχιτεκτονική Google Cloud Deployment Manager

## 4.7 Σύγκριση εργαλείων IaC

Σε αυτή την ενότητα και αφού έχουμε δώσει με σαφή και δομημένο τρόπο για το τι είναι το Infrastructure as Code, αλλά και ποια εργαλεία έχει αυτή η καινοτόμα προσέγγιση για την ανάπτυξη του κύκλου ζωής των συστημάτων λογισμικού, είναι επιτακτική ανάγκη να μπορέσουμε να αποτυπώσουμε μια ολοκληρωμένη σύγκριση των IaC εργαλείων που έχουν αναλυθεί σε προηγούμενες ενότητες. Έτσι λοιπόν στόχος είναι να υπάρξει μια σύντομη σύγκριση των εργαλείων, εξετάζοντας στοιχεία όπως η ευκολία χρήσης, η επεκτασιμότητα, οι δυνατότητες ολοκλήρωσης και η υποστήριξη που παρέχει η κοινότητα σε κάθε ένα από αυτά.

Συνεπώς, στοχεύουμε στο να εντοπίσουμε το καταλληλότερο εργαλείο για την πρακτική εφαρμογή στο πλαίσιο αυτής της εργασίας, διασφαλίζοντας φυσικά μια τεκμηριωμένη επιλογή που χρειάζεται για τους στόχους του έργου, που δεν είναι άλλοι από το να μπορέσουμε να δώσουμε σενάρια CI/CD pipelines σε ένα GitLab project. Παρακάτω, θα δώσουμε ένα συνοπτικό πίνακα με τα IaC εργαλεία, αλλά και τα κριτήρια που είχαμε σε όλη την διάρκεια των αναλύσεων πάνω στα εργαλεία και το τι μπορεί να προσφέρουν:

Πίνακας 3. Σύγκριση Infrastructure as Code εργαλεία

| Comparison of IaC Tools |   |   |   |  |   |  |  |   |
|-------------------------|---|---|---|--|---|--|--|---|
| KRITΗΡΙΑ                | Terraform   | Ansible                                 | Pulumi  | Chef   | Puppet  | AWS CloudFormation                                 | Azure Resource Manager [ARM]                 | Google Cloud Deployment Manager [GCDM]              |
| Infrastructure          | Mutable / Immutable   | Mutable                                 | Immutable   | Mutable  | Mutable   | Mutable / Immutable                                | Immutable                                    | Immutable   |
| Architecture            | Agentless Client-only   | Agentless                               | Agentless   | Client/Server  | Client/Server   | Agentless Client-Server                            | Agentless Client-Server                      | Agentless Client-Server                             |
| Code Approach           | Declarative   | Procedural                              | Procedural  | Procedural   | Declarative   | Declarative  | Declarative                                  | Declarative   |
| Data Format             | HCL   | YAML                                    | Various   | DSL / JSON   | Embedded DSL  | JSON, YAML   | JSON   | YAML  |
| Configuration Tool      | Modules, Resources  | Playbook                                | Stacks  | Cookbook   | Recipes / Cookbook  | Template   | Template                                     | Deployment configuration                            |
| Language                | Go  | Python                                  | Go  | Ruby   | Ruby  | AWS Lambda   | JSON   | YAML/PYTHON   |
| Ease to Use             | Hard  | Easy                                    | Easy  | Medium / Hard  | Medium / Hard   | Medium / Hard                                      | Medium / Hard                                | Easy  |
| Community Support       | Strong  | Large                                   | Growing   | Established  | Strong  | Strong   | Strong                                       | Growing   |
| Cloud Integration       | Multi Cloud   | Multi Cloud                             | Multi Cloud   | Multi Cloud  | Multi Cloud   | AWS Cloud  | Microsoft Azure Cloud                        | Google Cloud  |
| Model                   | Push / Pull   | Push                                    | Push  | Pull   | Pull  | Push / Pull  | Push   | Push  |
| Cost                    | Low (open-source)   | Low (open-source)                       | Low (open-source)   | Low (open-source)  | Low (open-source)   | Free (resource costs)                              | Free (resource costs)                        | Free (resource costs)                               |
| Management Purpose      | Orchestration, Provisioning, Configuration Management, Deployment | Orchestration, Deployment, Provisioning | Orchestration, Provisioning, Configuration Management, Deployment | Developer Based Infrastructure Automation, Automated workload deployment | System Management, Code Management, Configuration Automation, Reporting | Provisioning, Configuration Management, Deployment | Orchestration, Deployment of Azure resources | Orchestration, Deployment of Google Cloud resources |

Συνοψίζοντας λοιπόν, μπορούμε να πούμε τα εξής για τα Infrastructure as Code εργαλεία:

- **Terraform** : Χρησιμοποιεί κυρίως αμετάβλητη (Immutable) υποδομή, με αρχιτεκτονική χωρίς πράκτορα (Agentless), και με δηλωτική προσέγγιση [Declarative] και τα δεδομένα παράγονται μέσα από την HCL. Θεωρείται ένα από τα πιο ευέλικτα και επεκτάσιμα εργαλεία IaC, που μπορεί να χρησιμοποιήσει πολλαπλά περιβάλλοντα υπολογιστικού νέφους. Έχοντας ως μεγάλο πλεονέκτημα την μεγάλη κοινότητα υποστήριξης, μπορεί να ανταπεξέλθει επάξια σε μεγάλης κλίμακας έργα.
- **Ansible** : Χρησιμοποιεί μεταβλητή (Mutable) υποδομή, με αρχιτεκτονική χωρίς πράκτορα, με επιτακτική προσέγγιση (Imperative) και οι μορφές των δεδομένων είναι σε YAML αρχεία. Σημαντικό να αναφερθεί πως είναι ένα από τα πιο απλά εργαλεία, εύκολο στην χρήση, με μια κοινότητα που βοηθάει, και μπορεί να χρησιμοποιήσει πολλαπλά cloud περιβάλλοντα. Τέλος, είναι ιδανικό για εργασίες αυτοματισμού και ενορχήστρωσης.
- **Pulumi** : Χρησιμοποιεί αμετάβλητη (Immutable) υποδομή, με αρχιτεκτονική χωρίς πράκτορα (Agentless). Βασικό του χαρακτηριστικό είναι ότι μπορεί να υποστηρίξει πολλές προγραμματιστικές γλώσσες, και είναι εύκολο στην χρήση. Επειδή είναι νέο εργαλείο το οποίο έκανες την εμφάνισει του το 2018, η κοινότητα ακόμα μεγαλώνει για να μπορέσει να υποστηρίξει όλο και περισσότερα σενάρια. Χρησιμοποιεί την λογική του multi-cloud, και είναι αρκετά ελκυστικό από οιμάδες προγραμματιστές που είναι εξοικιωμένοι με γλώσσες προγραμματισμού.
- **Chef** : Χρησιμοποιεί μεταβλητή (Mutable) υποδομή, με αρχιτεκτονική client-server και με επιτακτική προσέγγιση χρησιμοποιώντας Ruby DSL. Έχει υψηλή ευελιξία και απόδοση και λόγω ότι είναι πιο παλιό εργαλείο, έχει μια καλά οργανωμένη κοινότητα.
- **Puppet** : Και εδώ, το Puppet χρησιμοποιεί μεταβλητή (Mutable) υποδομή, με αρχιτεκτονική client-server, και η προσέγγιση του είναι δηλωτική (Declarative). Παράλληλα έχει ασφαλή state διαχείριση για την υποδομή, και αξιόπιστη απόδοση.
- **AWS CloudFormation** : Χρησιμοποιεί αμετάβλητη (Immutable) υποδομή, η προσέγγιση του είναι δηλωτική (Declarative), με αρχιτεκτονική που βασίζεται στο cloud και τα data είναι σε μορφή JSON / YAML. Είναι άρτια συνδεδεμένο με όλες τις υπηρεσίες AWS.
- **Azure Resource Manager** : Το ARM χρησιμοποιεί αμετάβλητη (Immutable) υποδομή, βασισμένο σε cloud και με μια δηλωτική προσέγγιση με πρότυπα JSON.
- **Google Cloud Deployment Manager** : Χρησιμοποιεί αμετάβλητη (Immutable) υποδομή, με αρχιτεκτονική που βασίζεται και εδώ, στο cloud, έχει δηλωτική προσέγγιση και υποστηρίζει YAML / JSON / PYTHON.

Κλείνοντας, μετά την ανάλυση και την σύγκριση των Iac εργαλείων, θεωρούμε πως το **Terraform** είναι ένα ιδανικό εργαλείο για να χρησιμοποιηθεί στο πρακτικό κομμάτι της διπλωματικής εργασίας, λόγω της ισχυρής υποστήριξης που έχει η κοινότητα, επειδή είναι αρκετά επεκτάσιμο εργαλείο, και μπορεί άνετα να χρησιμοποιηθεί σε πολλούς cloud providers<sup>9</sup>.

<sup>9</sup><https://registry.terraform.io/>

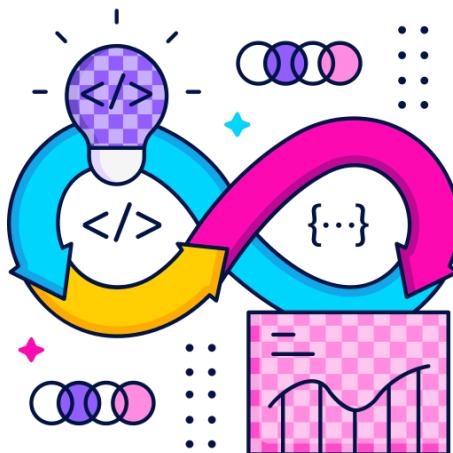
## 4.8 IaC: DevOps και CI/CD pipelines

Στις μέρες μας, σε ένα εξελισσόμενο τεχνολογικό τοπίο όσον αφορά την ανάπτυξη συστημάτων λογισμικού, οι καινοτόμες προσεγγίσεις όπως το Infrastructure as Code έχουν αρχίσει πλέον να υιοθετούνται από πολλούς τεχνολογικούς οργανισμούς, καθώς έχουν καταστεί ουσιαστικές για την αποτελεσματική διαχείριση πολύπλοκων συστημάτων, σε συνάρτηση με τις ολοένα και αυξανόμενες απαιτήσεις.

Έτσι λοιπόν, και με βάση την ανάλυση και αποτύπωση που έχουμε δώσει ως τώρα, από την αρχή της διπλωματικής εργασίας, ως προς τα πλεονεκτήματα που προσφέρει το Cloud Computing, την εξέλιξη του Software Development Lifecycle, αλλά και τις πρακτικές IaC και των εργαλείων του, αυτή η ενότητα είναι σε θέση να μπορέσει να εισάγει τις έννοιες των DevOps που έχουν αναφερθεί και κατά την διάρκεια της ανάλυσης στα προηγούμενα κεφάλαια, αλλά και την έννοια των CI/CD pipelines που είναι εν τέλη και το ζητούμενο της διπλωματικής μέσα από την έρευνα που έχει γίνει ως τώρα.

Συνεπώς, αυτές οι πρακτικές είναι σημαντικές καθώς προωθεί την ενοποίηση της ανάπτυξης των συστημάτων λογισμικού και των λειτουργιών του, την αυτοματοποίηση της διαδικασίας ανάπτυξης, και τις συνεχούς συνεργασίες μεταξύ των ομάδων, ενισχύοντας έτσι τη συνολικότερη αποτελεσματικότητα και αξιοπιστία του SDLC.

### 4.8.1 DevOps



Αρχικά είναι σημαντικό να αναφέρουμε πως το DevOps, εισήχθη για πρώτη φορά από τον Patrick Debois, και στοχεύει στην αντιμετώπιση των αναποτελεσματικών σταδίων ανάπτυξης λογισμικού, εστιάζοντας στη συνεχή ανάπτυξη και παράδοση. Παράλληλα, στοχεύει στην ελαχιστοποίηση του κόστους μέσω της υλοποίησης και της υιοθέτησης, επιτρέποντας τη συνεργασία μεταξύ των ομάδων ανάπτυξης και λειτουργίας εντός ενός οργανισμού [24]. Παρακάτω θα δοθεί ένας σαφής ορισμός του DevOps καθώς και ποιά είναι τα πλεονεκτήματα του.

#### Τι είναι το DevOps;

**! Ορισμός :** Το DevOps συνδυάζει ανάπτυξη (Dev) και λειτουργίες (Ops) για να αυξήσει την αποτελεσματικότητα, την ταχύτητα και την ασφάλεια της ανάπτυξης και παράδοσης λογισμικού σε σύγκριση με τις παραδοσιακές διαδικασίες. Ένας πιο ευκίνητος γρήγορος ζωής ανάπτυξης λογισμικού οδηγεί σε ανταγωνιστικό πλεονέκτημα για τις επιχειρήσεις και τους πελάτες τους [25].

## Φάσεις του Κύκλου ζωής του DevOps

Ο κύκλος ζωής του DevOps είναι μια συνεχής διαδικασία ανάπτυξης λογισμικού που χρησιμοποιεί τις βέλτιστες πρακτικές του DevOps για να σχεδιάσει, να δημιουργήσει, να ενσωματώσει, να αναπτύξει, να παρακολουθήσει, να λειτουργήσει και να προσφέρει συνεχή ανατροφοδότηση σε όλο τον κύκλο ζωής του λογισμικού [26].

- **Plan :** Σε αυτή την φάση, του σχεδιασμού στο DevOps περιλαμβάνετε ο σχεδιασμός του κύκλου ζωής του λογισμικού, όπου κάθε στάδιο να επαναλαμβάνεται ανάλογα με τις απαιτήσεις. Αυτό το στάδιο λαμβάνει υπόψη μελλοντικές επαναλήψεις, αλλά και προηγούμενες εκδόσεις, διασφαλίζοντας ότι οι προηγούμενες πληροφορίες ενημερώνουν την επόμενη επανάληψη [26]. Συνεπώς, σε αυτό το στάδιο όλες οι ομάδες λειτουργούν μαζί, για να διασφαλιστεί και να αποτυπωθεί ο στόχος του έργου που είναι προς υλοποίηση.
- **Code :** Σε αυτό το στάδιο, οι DevOps ξεκινούν να γράφουν τον κώδικα και θα τον προετοιμάσουν για την επόμενη φάση κατά το στάδιο της κωδικοποίησης [26]. Προφανώς και ο κώδικας που θα γραφτεί θα είναι στις προδιαγραφές και τις απαιτήσεις του έργου έτσι ώστε να επιτευχθεί η ανάπτυξη όλων των λειτουργιών που έχουν καταγραφεί στο στάδιο της σχεδίασης.
- **Build :** Στην συνέχεια, ο κώδικας ενσωματώνεται στη φάση της κατασκευής και όπου χρειάζεται θα γίνουν οι απαραίτητες αλλαγές. Συνεπώς σε αυτό το στάδιο χρησιμοποιούνται συστήματα version control όπως είναι το GitLab<sup>10</sup>. Με την εισαγωγή του κώδικα, ο προγραμματιστής θα περιμένει να γίνει ο απαραίτητος έλεγχος και εφόσον εγκριθεί, σημαίνει ότι είναι έτοιμος για να προστεθεί στο έργο.
- **Test :** Συνεχίζοντας με την φάση των δοκιμών και των ελέγχων, όλες οι ομάδες του έργου, θα χρειαστεί να κάνουν έλεγχο ώστε να διασφαλίσουν με αυτόν τον τρόπο ότι το λογισμικό που έχει αναπτυχθεί, λειτουργεί κανονικά.
- **Release :** Έχοντας κάνει και τους απαραίτητους ελέχους εισερχόμαστε στην φάση του release όπου πραγματοποιείται όταν έχει γίνει η απαραίτητη επαλήθευση ότι ο κώδικας είναι έτοιμος για να γίνει deploy.
- **Deploy :** Στη φάση ανάπτυξης τώρα, το λογισμικό και όλα τα συστατικά μέρη που το απαρτίζουν, προετοιμάζοται για το περιβάλλον παραγωγής και λειτουργεί όπως έχει προγραμματιστεί σε αυτό το περιβάλλον. Σημαντικό να αναφερθεί πως στο DevOps, είναι κοινή ευθύνη το να γίνει το deploy και γι'αυτό τον λόγο προάγεται η συνεργασία ώστε να εγγυηθούν όλες οι ομάδες μαζί πως η ανάπτυξη είναι επιτυχημένη.
- **Operate :** Συνεχίζοντας στη φάση λειτουργίας, οι ομάδες δοκιμάζουν το έργο σε περιβάλλον παραγωγής και οι τελικοί χρήστες χρησιμοποιούν το προϊόν. Αυτό το κρίσιμο στάδιο δεν είναι σε καμία περίπτωση το τελικό βήμα. Αντίθετα, ενημερώνει τους μελλοντικούς κύκλους ανάπτυξης και διαχειρίζεται τη διαμόρφωση του περιβάλλοντος παραγωγής και την υλοποίηση τυχόν απαιτήσεων χρόνου εκτέλεσης [26].

<sup>10</sup><https://about.gitlab.com/platform/>

- **Monitor :** Τέλος και σημαντικότερο, είναι η φάση της παρακολούθησης όλου του προϊόντος, καθώς αν παρουσιαστούν ζητήματα ή χρειαστούν να δημιουργηθούν οι δυνατότητες βελτίωσης του λογισμικού, τότε αναγνωρίζονται και τεκμηριώνονται [26]. Μετά, αυτές οι πληροφορίες που έχουν συλλεχθεί μεταφέρονται στην επόμενη επανάληψη για να βοηθήσουν την διαδικασία ανάπτυξης.

Παρακάτω μπορούμε να δούμε σε σχηματική απεικόνηση τον κύκλο ζωής του DevOps:



Εικόνα 32. Κύκλος ζωής του DevOps  
[27]

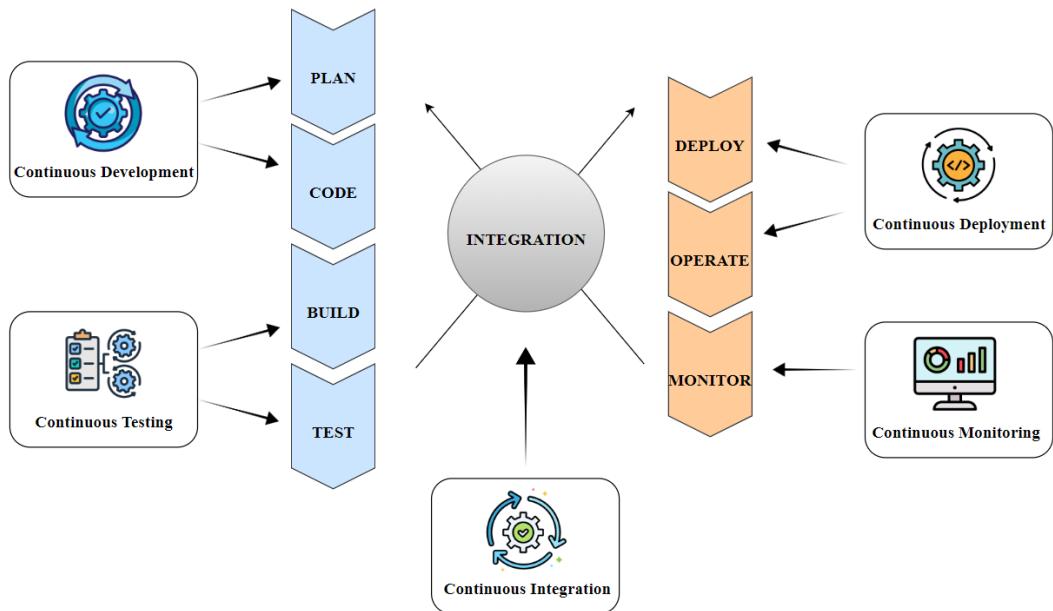
### Τα 7 Cs του DevOps

Έχοντας δώσει μια ανάλυση για της φάσεις που έχει η έννοια του DevOps, μπορούμε να τις αναλυσουμε σε 7 C για περισσότερη πρακτική σαφήνεια. Συνοπτικά λοιπόν έχουμε:

- **Continuous Development :** Η συνεχής ανάπτυξη αποτελείται από το συνδυασμό δύο φάσεων, που περιλαμβάνουν τον προγραμματισμό και την κωδικοποίηση. Σε αυτή τη φάση, ο στόχος του έργου καθορίζεται και οι προγραμματιστές αρχίζουν να δημιουργούν κώδικα για το λογισμικό. [24]. Αυτή είναι η πιο σημαντική φάση του DevOps, συνήθως χρησιμοποιούνται κάποια εργαλεία, όπως το Git.
- **Continuous Integration :** Η συνεχής ολοκλήρωση είναι μια διαδικασία που συνδύαζει διάφορα βήματα, όπως η συλλογή κώδικα, η επικύρωση του, η αποδοχή των δοκιμών, και δημιουργία πακέτων ανάπτυξης. Η συνεχής ενοποίηση ξεκινά εντοπίζοντας σφάλματα για τους προγραμματιστές [24]. Εδώ είναι σημαντικό κάθε ενσωμάτωση κώδικα να δοκιμάζεται, ώστε το σύστημα να παραμένει λειτουργικό μετά την εισαγωγή νέων αλλαγών από τους προγραμματιστές. Το κλειδί σε αυτή την φάση είναι η επαναληψιμότητα και η δοκιμή του κώδικα, ώστε να μπορέσει με σαφήνεια να αποτυπώνεται το σενάριο που είναι προς υλοποίηση.

- **Continuous Testing :** Η συνεχής δοκιμή είναι το στάδιο όπου η αναπτυγμένη εφαρμογή περνά από συνεχείς δοκιμές για τον εντοπισμό σφαλμάτων. Συνεπώς η συνεχής δοκιμή έχει δύο χαρακτηριστικά:
    - ! δοκιμές για ολόκληρο τον κύκλο ζωής, δηλαδή συνεχείς δοκιμές από την αρχή μέχρι το τέλος
    - ! στρατηγικός αυτοματισμός δοκιμών, που σημαίνει ότι οι δοκιμές προσαρμόζονται και εκτελούνται συνεχώς με βάση τις απαιτήσεις [24].
  - **Continuous Deployment :** Η συνεχής ανάπτυξη είναι μια διαδικασία που βοηθά στην γρήγορη και αυτόματη ανάπτυξη των συστημάτων λογισμικού, διατηρώντας ένα πρότυπο ποιότητας. Κατά τη διάρκεια της διαδικασίας συνεχούς ανάπτυξης, δεν υπάρχει ανάγκη για χειροκίνητες φάσεις ή λήψη αποφάσεων του προγραμματιστή. Συνεπώς, πραγματοποιούνται αυτοματοποιημένες δοκιμές [24]. Η συνεχής ανάπτυξη είναι ένας καλός τρόπος συλλογής σχολίων από τους χρήστες, γεγονός που μειώνει επίσης το κόστος παραγωγής.
  - **Continuous Feedback :** Σε αυτό το στάδιο, η συνεχής ανατροφοδότηση για την αξιολόγηση και τη βελτίωση του πηγαίου κώδικα του λογισμικού είναι το κύριο ζητούμενο. Κατά τη διάρκεια αυτής της φάσης, η συμπεριφορά του client εξετάζεται τακτικά για κάθε έκδοση σε μια προσπάθεια να βελτιωθούν οι μελλοντικές εκδόσεις και αναπτύξεις. Οι εταιρείες μπορούν να συλλέγουν σχόλια χρησιμοποιώντας:
    - ! **δομημένη στρατηγική**, όπου τα στοιχεία συλλέγονται με τη χρήση ερωτηματολογίων και ερευνών.
    - ! **μη δομημένη στρατηγική**, όπου τα σχόλια είναι μέσω των πλατφορμών κοινωνικής δικτύωσης.
- Συνεπώς, καταλαβαίνουμε πως αυτή η φάση είναι κρίσιμη για να καταστεί δυνατή η συνεχής παράδοση προκειμένου να κυκλοφορήσει μια καλύτερη έκδοση του προγράμματος [24].
- **Continuous Monitoring :** Η παρακολούθηση είναι ένα άλλο πάρα πολύ σημαντικό στάδιο στον κύκλο ζωής του DevOps. Αυτό συμβαίνει γιατί ένα σύστημα monitoring είναι αναγκαίο για τον προσδιορισμό της ορθότητας της συμπεριφοράς του συστήματος [24]. Αυτή είναι μια φάση κατά την οποία οι DevOps Engineers παρακολουθούν συνεχώς την απόδοση των συστημάτων λογισμικού.
  - **Continuous Operations :** Τέλος, κατά τη διάρκεια αυτής της φάσης, η λειτουργία και οι δυνατότητες της εφαρμογής παρακολουθούνται τακτικά για τον εντοπισμό σφαλμάτων του συστήματος, όπως η χαμηλή μνήμη ή σφάλμα κατά την πρόσβαση σε κάποιον server της υποδομής [24]. Με αυτόν τον τρόπο, επιτρέπει στις ομάδες πληροφορικής να εντοπίζουν γρήγορα ζητήματα απόδοσης της εφαρμογής και τις υποκείμενες αιτίες τους. Κάθε φορά που οι ομάδες IT ανακαλύπτουν ένα σοβαρό πρόβλημα, η εφαρμογή περνά ξανά από τον πλήρη κύκλο DevOps για να προσδιορίσει μια λύση [24].

Παρακάτω στο σχήμα έχουμε μια απεικόνηση των όσων αναλύθηκαν παραπάνω ανάλογα την κάθε φάση:



Εικόνα 33. Τα 7 Cs του DevOps

#### 4.8.2 CI/CD pipelines

Σε αυτή την υποενότητα και αφού έχουμε καλύψει πλήρως τον κύκλο ζωής του DevOps και ποιες είναι οι βασικές φάσεις του, είναι επικατακή ανάγκη να αναλύσουμε με συνοπτικό τρόπο και την έννοια των CI/CD pipelines.

Είναι γνωστό ότι για διακετίες, οι προγραμματιστές επιδίωκαν να αυτοματοποιήσουν τα επαναλαμβανόμενα στοιχεία κωδικοποίησης, ώστε να μπορούν να επικεντρωθούν την καινοτομία και να μειώσουν αρκετά τον χρόνο στο SDLC [28]. Το CI/CD λοιπόν ή αλλιώς Continuous Integration / Continuous Deployment/Delivery, είναι μια αποτελεσματική λύση, καθώς έχει στόχο να επιλύσει προβλήματα που σχετίζονται με την ενσωμάτωση νέου κώδικα και τη μη αυτόματη δοκιμή.

Η συνεχής ενοποίηση, η συνεχής παράδοση και η συνεχής ανάπτυξη εξορθολογίζουν τη διαδικασία συνδυασμού της εργασίας από χωριστές ομάδες σε ένα αποτελεσματικό λογισμικό. Το CI/CD είναι σε θέση να παρέχει ένα ενιαίο χώρο αποθήκευσης για την αποθήκευση της εργασίας και αυτοματοποιεί με συνέπεια την ενοποίηση και τη συνεχή δοκιμή [28]. Παρακάτω, θα δώσουμε έναν ορισμό αυτής της έννοιες, καθώς και ποια είναι τα στάδια του και τα εργαλεία που χρησιμοποιεί.

**! Ορισμός :** Οι CI/CD pipelines είναι οι αυτοματοποιημένες διαδικασίες που αντικατροπτίζονται σε μια σειρά βημάτων, όπου συγκεντρώνουν όλα τα στάδια ή κάποια από αυτά, του κύκλου ζωής των συστημάτων λογισμικού και δημιουργούν μια ροή εργασιών αυτοματισμού. Αυτοί οι αγωγοί πρακτικά στοχεύουν στο να εφαρμόσουν με βέλτιστο τρόπο όλες τις φάσεις του DevOps, επιτρέποντας στις εταιρίες να παρέχουν το λογισμικό ταχύτερα και πιο αξιόπιστα.

Από τον ορισμό λοιπόν, συμπεραίνουμε πως οι αυτοματοποιημένοι αγωγοί μπορούν να βοηθήσουν στην αποτροπή σφαλμάτων, να επιτρέπουν γρήγορες επαναλήψεις στην ανάπτυξη του λογισμικού, και να παρέχουν ένα συνεχές feedback κατά την διαδικασία της ανάπτυξης. Αξίζει να σημειωθεί, πως κάθε CI/CD pipeline είναι ένα υποσύνολο ομαδοποιημένων εργασιών.

## Ροή εργασίας των CI/CD pipelines

Εφόσον έχουμε δει όλα τα στοιχεία του DevOps και πως ακριβώς ορίζεται ένας CI/CD αγωγός, είναι ένα σημαντικό σημείο, ώστε να μπορέσουμε να αποτυπώσουμε το πως λειουργεί ένας τέτοιος αγωγός, ποια είναι δηλαδή τα στάδια που ακολουθεί, αυτοματοποιημένα.

- **Source :** Ένα pipeline συνήθως ενεργοποιείται από ένα version control πηγαίου κώδικα. Οι αλλαγές στον κώδικα ενεργοποιούν μια ειδοποίηση στο εργαλείο αγωγών CI/CD, το οποίο λειτουργεί τον αντίστοιχο αγωγό [28]. Τέλος, οι ροές εργασίας που ξεκινούν ή προγραμματίζονται αυτόματα από τον χρήστη ή τα αποτελέσματα άλλων αγωγών ενδέχεται επίσης να ενεργοποιήσουν ένα pipeline.
- **Build :** Κατά τη διάρκεια της φάσης κατασκευής, οι μηχανικοί DevOps μοιράζονται τον κώδικα που έχουν αναπτύξει μέσω του εργαλείου version control για να δημιουργήσουν μια εκτελέσιμη επανάληψη για το λογισμικό. Σε πολλές περιπτώσεις οι DevOps μηχανικοί χρησιμοποιούν εργαλεία όπως το Docker<sup>11</sup> για να γίνει η ανάπτυξη του λογισμικού στο cloud, και αυτό το στάδιο του pipeline θα δημιουργήσεις τα απαραίτητα Docker containers [28].

**!** Είναι σημαντικό αυτό το στάδιο να είναι επιτυχές σε ένα CI/CD pipeline σενάριο, αλλιώς υποδηλώνει ότι κάτι δεν πάει καλά στην διαμόρφωση.

- **Test :** Κατά τη διάρκεια της δοκιμής, γίνεται επικύρωση του κώδικα και έχουμε την ευκαιρία να παρατηρήσουμε πως λειτουργεί και συμπεριφέρεται το λογισμικό. Είναι ένα πολύ σημαντικό στάδιο, καθώς παρέχει την απαραίτητη ασφάλεια και εμποδίζει με αυτόν τον τρόπο να εμφανιστούν τυχόν σφάλματα στους τελικούς χρήστες. Βασικό προαπαιτούμενο για να επιτευχθεί η κυκλοφορία του λογισμικού όσο πιο σύντομα γίνεται στην αγορά [28], είναι να δημιουργήσουν οι DevOps μηχανικοί, τις αυτοματοποιημένες δοκιμές. Στόχος λοιπόν στο στάδιο αυτό είναι να δοθεί άμεσα οποιαδήποτε ανατροφοδότηση στους προγραμματιστές, για πιθανά σφάλματα.
- **Deploy :** Μόλις δημιουργηθεί και δοκιμαστεί μια παρουσία όλου του κώδικα με δυνατότητα εκτέλεσης, είναι έτοιμη για ανάπτυξη [28]. Παράλληλα είναι εξαιρετικά οφέλιμο το γεγονός ότι σε ένα CI/CD pipeline σενάριο, μπορούμε να δημιουργήσουμε κατάλληλο κώδικα που να αναπτύσσεται με συγκεκριμένο χρονοδιάγραμμα, και να γίνεται επιλογή σε ποια περιβάλλοντα πελατών θα χρειασθεί αυτή η αλλαγή ή να επαναφέρει ένα version σε περίπτωση κάποιου προβλήματος.

<sup>11</sup><https://docs.docker.com/>

## CI/CD εργαλεία

Σήμερα, υπάρχουν πολλά εργαλεία που μπορούν να χρησιμοποιηθούν για την αυτοματοποιημένη διαδικασία CI/CD για οποιοδήποτε λογισμικό. Μερικά από τα ευρέως χρησιμοποιούμενα εργαλεία CI/CD μπορούμε να πούμε πως είναι τα εξής:

- **Jenkins** : Πρακτικά είναι ένας open-source server, που χρησιμοποιείται σε CI/CD pipelines. Προσφέρει ένα ευρύ φάσμα από plugins, καθιστώντας το αρκετά ευέλικτο.
- **Travis CI** : Είναι ένα εργαλείο που βασίζεται σε cloud, μπορεί να ενσωματωθεί στο Github και υποστηρίζει πολλές γλώσσες προγραμματισμού.
- **CircleCI** : Είναι ουσιαστικά μια πλατφόρμα CI/CD, βασίζεται σε cloud, και προσφέρει ένα αρκετά φιλικό προς τον χρήστη UI (User Interface), και υποστηρίζει ένα ευρύ φάσμα από γλώσσες προγραμματισμού.
- **GitLab CI/CD** : Αυτό είναι ένα ενσωματωμένο εργαλείο CI/CD του GitLab, καθώς είναι αρκετά χρήσιμο για εκείνες τις ομάδες που χρησιμοποιούν ήδη την πλατφόρμα του GitLab για version control.
- **Azure Pipelines** : To Azure Pipelines είναι μια υπηρεσία CI/CD και προσφέρεται από την Microsoft, μέσα από την συνίτια Azure DevOps <sup>12</sup>.
- **Github Actions** : To GitHub Actions είναι ένα εργαλείο CI/CD που είναι ενσωματωμένο με την πλατφόρμα GitHub, και προσφέρει ένα καλό UI στους χρήστες, ώστε να μπορούν να ενσωματώσουν τα σενάρια ανάπτυξης λογισμικού.

## Επιλογή CI/CD εργαλείου

Όπως παρατηρούμε είναι χρήσιμο μέσα από μια σύντομη αναφορά στα CI/CD εργαλεία που υπάρχουν, να δούμε με ποια κριτήρια πρέπει να επιλέξουμε ένα εργαλείο για την διπλωματική εργασία.

Η επιλογή ενός εργαλείου CI/CD θα πρέπει να ενσωματώνεται με άλλα εργαλεία και πλατφόρμες, να είναι φιλική προς το χρήστη, να κλιμακώνεται και να προσαρμόζεται. Θα πρέπει να χειρίζεται την πολυπλοκότητα των ροών εργασίας των DevOps ομάδων, να προσφέρει γρήγορους χρόνους κατασκευής και να είναι προσαρμόσιμος στις συγκεκριμένες ανάγκες τους.

Συμπερασματικά λοιπόν μπορούμε να πούμε πως το **GitLab** είναι μια ιδανική επιλογή για αυτή την διπλωματική εργασία, καθώς μπορεί να ενσωματώσει αρκετά εύκολα αρχεία Terraform και CI/CD pipelines. Σημαντικό επίσης είναι πως είναι ανοιχτού κώδικα, και αυτό επιτρέπει την εγκατάσταση του σε έναν private server, κάτι που το καθιστά ιδανικό σε σενάρια μεγάλης κλίμακας, και όπου η ασφάλεια παίζει σημαντικό ρόλο [28].

<sup>12</sup><https://azure.microsoft.com/en-us/products/devops/pipelines/>



## Κεφάλαιο 5

# Μεθοδολογία – Υλοποίηση – Εφαρμογή

Στο Κεφάλαιο 5, θα εφαρμόσουμε αυτές τις έννοιες πρακτικά, που έχουν αναφερθεί στα προηγούμενα κεφάλαια, χρησιμοποιώντας το **Terraform** για να επιδείξουμε την αυτοματοποίηση της παροχής υποδομής σε έναν αγωγό CI/CD μέσω του GitLab καθώς και το deploy της εφαρμογής στην υποδομή. Συνεπώς, αυτή η πρακτική εφαρμογή θα παρουσιάσει την ενσωμάτωση του Terraform σε μια ροή εργασίας DevOps, δίνοντας έμφαση στα οφέλη του Infrastructure as Code στις σύγχρονες πρακτικές ανάπτυξης λογισμικού. Κλείνοντας αυτό το Κεφάλαιο, θα δώσουμε συνοπτικά το χρονοδιάγραμμα της διπλωματικής εργασίας, καθώς και τον προυπολογισμό που είχαμε κατά την εκπόνηση όλου του έργου.

### 5.1 Μεθοδολογία

Αρχικά, είναι σημαντικό να αναφερθεί η μεθοδολογία που χρησιμοποιήθηκε σε αυτή την μελέτη, καθώς έχει σχεδιαστεί για να παρέχει μια ολοκληρωμένη διερεύνηση της πρακτικής εφαρμογής των καινοτόμων προσεγγίσεων στο Software Development LifeCycle (SDLC). Πρακτικά, δίνεται ιδιαίτερη έμφαση στην χρήση της Υποδομής με το εργαλείο Terraform, που είναι ένα Infrastructure as Code εργαλείο καθώς και των βέλτιστων πρακτικών του, με την χρήση του Linode ως cloud provider. Συνεπώς, στις παρακάτω υποενότητες, θα δούμε με σαφήνεια τη μεθοδολογία που υιοθετήθηκε και των τεχνικών που εφαρμόζονται στο σενάριο της διπλωματικής εργασίας.

#### 5.1.1 Ερευνητική προσέγγιση

Σε αυτό το σημείο, και αφού έχουμε δώσει μια πρώτη εικόνα για την μεθοδολογία, είναι σημαντικό να δώσουμε έμφαση και στην ερευνητική προσέγγιση για αυτή την διπλωματική εργασία. Η προσέγγιση που επιλέχθηκε περιέχει τόσο ποιοτικά, όσο και ποσοτικά στοιχεία. Έτσι λοιπόν, με γνώμονα την παροχή μιας ολοκληρωμένης κατανόησης των καινοτόμων προσεγγίσεων SDLC και των εργαλείων IaC, σε όλη την διάρκεια της εργασίας κλιθήκαμε να παρουσιάσουμε μια ποιοτική πτυχή, που περιλαμβάνει την βιβλιογραφική ανασκόπηση και ανάλυση των πρακτικών που προαναφέρθηκαν, και μία ποσοτική πτυχή, που περιλαμβάνει την πρακτική εφαρμογή και αξιολόγηση της απόδοσης αυτών των πρακτικών, χρησιμοποιώντας προφανώς συγκεκριμένα εργαλεία.

Η ποιοτική έρευνα περιελάμβανε μια εκτενή ανασκόπηση επιστημονικών άρθρων και εκθέσεων πάνω στο SDLC και το IaC, ώστε να είμαστε σε θέση να εντοπίσουμε τις καινοτόμες τάσεις. Από την άλλη πλευρά, η ποσοτική έρευνα περιελάμβανε πρακτικά πειράματα και δοκιμές, εστιάζοντας στο πως μπορούμε να χρησιμοποιήσουμε το εργαλείο IaC, Terraform, τις Docker containerized τεχνικές, καθώς και την αυτοματοποίηση όλης της υποδομής μέχρι να γίνει deploy η εφαρμογή, με τεχνικές CI/CD αγωγούς (pipelines) μέσα από το GitLab.

### 5.1.2 Μέθοδοι

Συνεχίζοντας λοιπόν, είναι σημαντικό να θέσουμε τις βάσεις για το πρακτικό κομμάτι αυτής της διπλωματικής εργασίας, καθώς πραγματοποιήθηκε από μια σειρά δομημένων βήμάτων, που στόχο έχουν να δημιουργηθεί μια εικονική μηχανή (VM) μέσα στο Linode, να γίνει η εγκατάσταση του Docker ώστε μετέπειτα να γίνει deploy μια απλή εφαρμογή. Πιο συγκεκριμένα έχουμε:

#### Εργαλεία και Τεχνικές

- Infrastructure as Code tool : Όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο μέσα στην διπλωματική εργασία για το πρακτικό σκέλος της, το Terraform είναι το εργαλείο που χρησιμοποιήσαμε για την παροχή της υποδομής.
- Τεχνική Docker: Μέσω του Docker δημιουργήσαμε το κατάλληλο repository στο docker hub ώστε να κάνουμε deploy την εφαρμογή, καθώς αυτή η πλατφόρμα ανοιχτού κώδικα μας επιτρέπει να δημιουργήσουμε ένα package της εφαρμογής μας, με όλες τις εξαρτήσεις που μπορεί να έχει, σε μια τυποποιημένη μονάδα που ονομάζεται **container**.
- GitLab CI/CD pipeline: Χρησιμοποιήσαμε την πλατφόρμα του GitLab ώστε να δημιουργήσουμε τον κατάλληλο αγωγό στο αρχείο **.gitlab-ci.yml**, και με αυτόν τον τρόπο να αυτοματοποιήσουμε πλήρως ολόκληρη την διαδικασία.

#### Ρύθμιση περιβάλλοντος

- Η υποδομή που δημιουργήθηκε ήταν μια εικονική μηχανή (VM) στον cloud provider Linode, χρησιμοποιώντας το Terraform. Αυτό φυσικά, περιελάμβανε την σύνταξη κατάλληλων αρχείων .tf και των σεναρίων Terraform για τον καθορισμό της διαμόρφωσης VM και την χρήση του παρόχου για γίνει connection με το πρωτόκολλο ssh και να γίνει η εγκατάσταση του Docker.

#### Αυτοματοποίηση με GitLab CI/CD αγωγούς

- Ο στόχος όλου του πρακτικού μέρους του σεναρίου είναι η δημιουργία ενός αρχείου .gitlab-ci.yml για την αυτοματοποίηση της διαδικασίας ανάπτυξης όλων των στοιχείων της υποδομής και της εφαρμογής. Συνεπώς αυτό το αρχείο που είναι ένα YAML αρχείο, περιέχει τους ορισμούς του pipeline για τα διάφορα στάδια της ανάπτυξης του σεναρίου, συμπεριλαμβανομένης της κατασκευής, της δοκιμής και της ανάπτυξης της εφαρμογής.

## Ανάπτυξη εφαρμογής

- Σε αυτό το σημείο, μέσα στο GitLab project, πέρα από την ρύθμιση των αρχείων διαμόρφωσης Terraform για την υποδομή, έχουμε και τα αρχεία που αναπτύχθηκαν για την εφαρφογή, η οποία είναι μια απλή python εφαρμογή. Έτσι περιλαμβάνεται η απαραίτητη διαμόρφωση των υπηρεσιών και η διασφάλιση της ομαλής εφαρμογής εντός ενός **docker container**.

### Δοκιμή και επικύρωση

- Τέλος, μέσα στον CI/CD αγωγό, ενσωματώνονται όλες οι δοκιμές ώστε να διασφαλιστεί ότι η εφαρμογή λειτουργεί όπως αναμένεται. Αυτές οι δοκιμές έχουν σχεδιαστεί ώστε να επικυρώνουν τόσο τη διαδικασία ανάπτυξης, όσο και την απόδοση της εφαρμογής. Σκοπός είναι μετά το πέρας του αγωγού να μπορούμε να πάρουμε κάποια report.

### 5.1.3 Τεχνικές Ανάλυσης

Η ανάλυση της πρακτικής εφαρμογής επικεντρώθηκε σε διάφορες βασικές μετρήσεις:

- Deployment Time :** Σε αυτό το σημείο πρακτικά είδαμε πως μέσα από έναν CI/CD αγωγό, με αυτοματοποίηση όλης της διαδικασίας της παροχής υποδομής και του τεστ της εφαρμογής, μέχρι την ώρα που θα γίνει deploy, μπορεί η συνολική ανάπτυξη μέσα από τις καινοτόμες προσεγγίσεις να γίνει σε σύντομο χρονικό διάστημα και με την προσέγγιση του Code Everything.
- Χρήση πόρων:** Με αυτή την μέτρηση πρακτικά μας δόθηκε η δυνατότητα μέσα από την πλατφόρμα του cloud παρόχου, και στην δική μας περίπτωση το Linode, να παρακολουθήσουμε την χρήση της CPU, της μνήμης και της αποθήκευσης του VM, ώστε να διασφαλιστεί η αποτελεσματική χρήση των πόρων. Είναι σημαντικό να αναφέρουμε πως χρησιμοποιήθηκε το minimum για τους πόρους, καθώς με περισσότερα resources, υπήρχαν και οι ανάλογες χρεώσεις.
- Επεκτασιμότητα :** Σε αυτό το σημείο έγινε μια αξιολόγηση του πόσο καλά κλιμακώνεται η διαδικασία ανάπτυξης του σεναρίου μέσα από την αυτοματοποίηση του CI/CD pipeline, καθώς έχει τεράστιες δυνατότητες σε μια μεγάλης κλίμακας έργο.
- Αποτελέσματα δοκιμής :** Τέλος σε αυτό το στάδιο έγινε η απαραίτητη ανάλυση των αποτελεσμάτων των αυτοματοποιημένων δοκιμών για τον εντοπισμό τυχόν ζητημάτων, καθώς και να έχουμε μια εικόνα για το πως λειτουργεί ένας τέτοιος αγωγός, με τι προτεραιότητες και πόσο ευέλικτο μπορεί να γίνει με την χρήση του Terraform ως Infrastructure as Code εργαλείο.

## 5.2 Περιγραφή Υλοποίησης

Συνεχίζοντας, σε αυτή την ενότητα είναι πολύ σημαντικό να παρέχουμε μια περιεκτική, αλλά σαφή περιγραφή των διαδικασιών του σεναρίου που χρησιμοποιήθηκαν για την μελέτη περίπτωσης αυτής της διπλωματικής εργασίας. Πιο συγκεκριμένα, καλούμαστε να αναλύσουμε με λεπτομέρεια τις διαδικασίες, τα τεστ που έγιναν κατά την διάρκεια της υλοποίησης, καθώς και τυχόν δυσκολίες που προέκυψαν μέσα από την υιοθέτηση των καινοτόμων τεχνικών για το SDLC.

### 5.2.1 Λεπτομερής Περιγραφή των Διαδικασιών

#### Ρύθμιση υποδομής με Terraform

- **Configuration files :** Έχουν αναπτυχθεί αρχεία διαμόρφωσης Terraform .tf αρχεία, για τον καθορισμό της υποδομής, με όλες τις προδιαγραφές για το Linode VM.
- **Provisioning :** Αφού δημιουργήθηκαν οι προδιαγραφές της υποδομής, χρησιμοποιήθηκαν Terraform provisioner για να μπορέσει να κάνει εγκατάσταση το Docker όπου το χρησιμοποιήσαμε για την εφαρμογή, μετά την δημιουργία της υποδομής.
- **Execution :** Τέλος, έγινε εφαρμογή των διαμορφώσεων Terraform για την δημιουργία του VM, διασφαλίζοντας ότι πληροί τις απαιτούμενες προδιαγραφές και για τα επόμενα βήματα ανάπτυξης. Η εφαρμογή έγινε μέσα από ένα Terraform Template, που ακολουθεί την αρχιτεκτονική του Terraform: write, plan, apply και ενσωματώθηκε με κατάλληλες προτεραιότητες στο YAML αρχείο του αγωγού **.gitlab-ci.yml**.

#### Αυτοματισμός με GitLab CI/CD pipeline

- **Pipeline Configuration :** Όπως έχουμε πει προηγουμένως, δημιουργήσαμε ένα αρχείο **.gitlab-ci.yml** για τον καθορισμό όλων των σταδίων του αγωγού CI/CD, όπως **validate**, **test-app**, **build** [**build-linode**, **build-image**], **deploy** [**deploy-linode**, **deploy-image**].
- **Validate stage :** Σε αυτό το σημείο, στην αρχή του αγωγού πρώτα γίνεται ο κατάλληλος έλεγχος για τα αρχεία διαμόρφωσης Terraform ότι πληρούν όλες τις προδιαγραφές που πρέπει να έχει ένα .tf αρχείο με την γλώσσα HCL του Terraform.
- **Test stage :** Σε αυτό το στάδιο, έγινε η κατάλληλη διαμόρφωση για να γίνουν κάποια τεστ για την εφαρμογή, αυτοματοποιημένα, και κατά την διάρκεια του "kick off" του pipeline να επικυρωθεί στην ουσία η σωστή λειτουργία της εφαρμογής.
- **Build stage :** Έγινε η κατάλληλη διαμόρφωση στο pipeline για την δημιουργία του virtual machine στο cloud, και στην δημιουργία του docker container για την εφαρμογή.
- **Deploy stage :** Σε αυτό το στάδιο του αγωγού πρακτικά καθορίσαμε όλα τα βήματα που χρειάζεται να γίνουν ώστε να μπορέσει αυτόματα να δημιουργηθεί το VM με τις προδιαγραφές που έχουμε εισάγει, και στην συνέχεια να γίνει η αυτόματη ανάπτυξη της εφαρμογής μέσα στο συγκεκριμένο VM.

#### Ανάπτυξη και διαμόρφωσης εφαρμογής

- **Docker setup :** Στο σημείο αυτό, μέσα από την εφαρμογή που χρησιμοποιήσαμε και αντλήσαμε από το διαδίκτυο, έγινε η χρήση του Docker σαν τεχνική, και ρυθμίστηκε κατάλληλα στο Linode VM για να μπορέσει να χρησιμοποιηθεί το docker container της εφαρμογής.

### 5.2.2 Δυσκολίες που αντιμετωπίστηκαν

- **Διαθεσιμότητα πόρων :** Χρησιμοποιήσαμε ένα Linode plan για την παροχή των πόρων της εικονικής μηχανής **Linode 2 GB** που έχει τα εξής χαρακτηριστικά: RAM: 2 GB, CPUs: 1, Storage: 50GB, Transfer: 2TB, Network In/Out: 40Gbps/2Gbps. Σε αυτό το σενάριο που δημιουργήσαμε ώστε να κάνουμε test την εφαρμογή και να δούμε πως λειτουργεί η αυτοματοποιημένη διαδικασία μέσα από την προσέγγιση του CI/CD pipeline, επαρκούσε. Από την άλλη πλευρά, προφανώς η χρήση περισσότερων resources είναι διαθέσιμα, ανάλογα και την κλιμάκωση που έχεις να δημιουργήσεις για το σενάριο σου. Και αυτό συνεπάγεται με περισσότερο κόστος, χρησιμοποιώντας περισσότερους πόρους στην πλατφόρμα του Linode. Στην διπλωματική αυτή, χρησιμοποιήθηκε το minimum ώστε να γίνει σωστό balance στο κόστος.
- **Δυσκολίες στην ανάπτυξη :** Σε αυτό το σημείο είναι σημαντικό να καταγραφούν δυσκολίες που παρουσιάστηκαν κατά την διάρκεια του σεναρίου. Αρχικά, αντιμετωπίσαμε προκλήσεις κατά τη σύνταξη και τον εντοπισμό σφαλμάτων στα Terraform Configuration files, ειδικότερα στον χειρισμό του Terraform provisioner και του να μπορέσεις να κάνεις retrieve και να έχεις κατάλληλο artifact<sup>1</sup> της Linode IP διένθυνσης ώστε να μπορέσεις να το χρησιμοποιήσεις μετέπειτα για την αυτοματοποίηση του login στην εικονική μηχανή μέσα SSH σύνδεσης και να μπορέσεις να κάνεις deploy την εφαρμογή με το docker.
- **GitLab platform - trial :** Επίσης, καθώς επιλέχθηκε η πλατφόρμα του GitLab ως version control για το repository του πρκατικού μέρους της διπλωματικής εργασίας, είναι σημαντικό να αναφερθεί πως υπήρχε ο περιορισμός του trial που ήταν για κάποιες ημέρες από την ενεργοποίηση του, καθώς και ο περιορισμός για το πόσες φορές μπορείς να κάνεις "kick off" έναν αγωγό. Υπήρχε η δυνατότητα με αγορά 10 δολαριών να μπορέσεις να έχεις περισσότερες προσπάθειες στο κομμάτι του compute, πράγμα που επιλέχθηκε μια φορά κατά την διάρκεια του σεναρίου.
- **Cloud Providers :** Σημαντικό είναι το γεγονός πως υπήρχαν περιορισμοί ως προς το κόστος και στην επιλογή του cloud παρόχου. Έγινε έλεγχος και χρήση και άλλων παρόχων όπως το Microsoft Azure, AWS και DigitalOcean, που είχαν στην ουσία την ίδια δομή για το κόστος με τα resources που θα χρειαζόσουν. Επιλέχθηκε το Linode ως μια cloud πλατφόρμα πιο εύχρηστη με καλύτερο User Interface (UI).

<sup>1</sup>Tα pipeline artifacts είναι αρχεία που παράγονται από ένα βήμα μέσα σε ένα pipeline.

## 5.3 Αρχιτεκτονική Συστήματος

Αυτή η υποενότητα, και αφού έχουμε δώσει μια συνοπτική εικόνα για το πρακτικό μέρος της διπλωματικής εργασίας, στοχεύει στο να εμβαθύνουμε περισσότερο στον αρχιτεκτονικό σχεδιασμό του συστήματος που εφαρμόζεται στην μελέτη περίπτωσης. Πιο συγκεκριμένα, σε αυτό το μέρος θα περιγράψουμε τα βασικά στοιχεία, τις αλληλεπιδράσεις τους και την ενοποίηση τους στον αγωγό CI/CD. Για να γίνει πιο κατανοητό το μέρος της αρχιτεκτονικής του συστήματος, θα παρουσιάσουμε σχηματικό διάγραμμα για να παρέχουμε την οπτικοποίηση της αρχιτεκτονικής.

### 5.3.1 Infrastructure : Terraform Configuration files

Αρχικά έχουμε δημιουργήσει τα αρχεία .tf για την διαμόρφωση της υποδομής μας. Πρακτικά πρόκειται για ένα Linode VM, το οποίο λειτουργεί ως κύριος υπολογιστής για την εφαρμογή. Για τα αρχεία διαμόρφωσης, μπορούμε εύκολα να ανατρέξουμε στο Παράρτημα A. Τα αρχεία του Terraform που έχουν δημιουργηθεί είναι τα εξής:

- [69] **drosos.tf** : Σε αυτό το αρχείο γίνεται η αρχικοποίηση του cloud provider και το token που έχει δημιουργηθεί στην πλατφόρμα του Linode ώστε να μπορέσει το API να επικοινωνήσει και να πιστοποιήσει την αυθεντικότητα.
- [70] **linode-vm01.tf** : Το αρχείο αυτό έχει δημιουργηθεί να δώσει με δηλωτικό τρόπο όλες τις προδιαγραφές που χρειάζεται η εικονική μηχανή ως υποδομή, χρησιμοποιώντας μεταβλητές και έναν terraform provisioner ώστε να αυτοματοποιηθεί να μπορέσει να κάνει εγκατάσταση το docker. Σημαντικό να αναφέρουμε πως ο **provisioner "remote-exec"**<sup>2</sup> που χρησιμοποιούμε, καλεί ένα σενάριο σε έναν απομακρυσμένο πόρο μετά τη δημιουργία του. Αυτό μπορεί να χρησιμοποιηθεί για την εκτέλεση ενός εργαλείου διαχείρισης διαμόρφωσης, την εκκίνηση σε ένα cluster κ.λπ. Εμείς το χρησιμοποιούμε για να μπορέσουμε αυτοματοποιημένα μέσα στο virtual machine να κάνουμε εγκατάσταση το docker.
- [71] **output.tf** : Γενικότερα στο Terraform, τα output.tf αρχεία καθιστούν διαθέσιμες πληροφορίες σχετικά με την υποδομή μας στη γραμμή εντολών και μπορούν να παρουσιάσουν τις πληροφορίες για άλλες διαμορφώσεις Terraform προς χρήση. Σημαντικό να αναφερθεί πως οι τιμές εξόδου είναι παρόμοιες με τις επιστρεφόμενες τιμές στις γλώσσες προγραμματισμού. Το αρχείο διαμόρφωσης εδώ, θα ενεργήσει κατάλληλα ώστε να μπορέσει να κάνει extract την ip address του Linode VM που θα δημιουργηθεί, ώστε μετέπειτα να χρησιμοποιηθεί στην διάρκεια του CI/CD pipeline ως artifact για το deploy της εφαρμογής.
- [72] **variables.tf** : Ένα αρχείο variables.tf χρησιμοποιείται για να ορίσει τον τύπο μεταβλητών και προαιρετικά να ορίσει μια προεπιλεγμένη τιμή. Το αρχείο αυτό ουσιαστικά, αρχικοποιεί τις μεταβλητές που χρησιμοποιούμε στα Terraform Config αρχεία και δίνεται προαιρετικά μια περιγραφή και υποχρεωτικά το type της μεταβλητής (π.χ string).

<sup>2</sup><https://developer.hashicorp.com/terraform/language/resources/provisioners/remote-exec>

- [73] **drososdevenv.tfvars** : Πρακτικά στο Terraform τα αρχεία .tfvars χρησιμοποιούνται για την αποθήκευση ορισμών μεταβλητών. Έτσι με αυτό τον τρόπο μας επιτρέπει να διευκολύνει τη διαχείριση τους, ειδικά εάν έχουμε μεγάλο αριθμό μεταβλητών ή χρειάζεται να κάνουμε χρήση τις ίδιες μεταβλητές σε πολλά περιβάλλοντα. Συνεπώς, σε αυτό το αρχείο περιέχονται οι τιμές που έχει κάθε μεταβλητή που θα χρησιμοποιηθεί στα .tf αρχεία.

### 5.3.2 CI/CD pipeline με GitLab

Σε αυτή την υποενότητα είναι κρίσιμο να δώσουμε μια σαφή εικόνα για το κυριάρχο μέρος του πρακτικού κομματιού της διπλωματικής εργασίας που είναι το CI/CD pipeline. Ο αγωγός GitLab CI/CD που είναι ένα YAML<sup>3</sup> αρχείο (**.gitlab-ci.yml**) αυτοματοποιεί τον κύκλο ζωής ανάπτυξης λογισμικού, διασφαλίζοντας με αυτόν τον τρόπο την κατασκευή, τη δοκιμή και την ανάπτυξη αποτελεσματικών αλλαγών στον κώδικα.

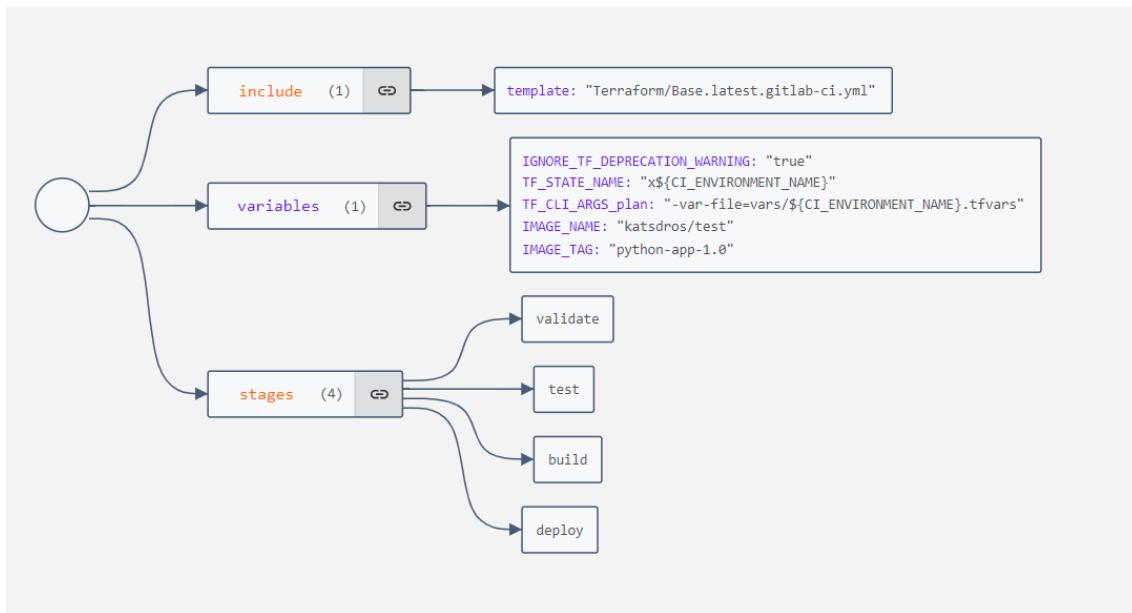
Πρακτικά, αυτό που καλείται να κάνει είναι να χρησιμοποιήσει τις ισχυρές δυνατότητες της έννοιας **Continuous Integration/Continuous Deployment - Development** ώστε να πετύχει τη μείωση των σφαλμάτων και την βελτίωση της παραγωγικότητας της ανάπτυξης των συστημάτων του λογισμικού. Κάθε στάδιο (**stages**) είναι σε θέση να διαχειριστεί εργασίες (**jobs**), όπως Docker images, αυτοματοποιημένες δοκιμές και την ανάπτυξη των εφαρμογών, έχοντας νιοθετήσεις τις σύγχρονες και βέλτιστες πρακτικές προσεγγίσεις του SDLC. Συνολικότερα όλα τα στάδια του CI/CD pipeline και γενικότερα όλο το .gitlab-ci.yml αρχείο μπορούμε να τα δούμε στο **Παράρτημα Β.**

Σε αυτό το σημείο θα αναλύσουμε βήμα-βήμα σε σχεδιαστικό διάγραμμα πως είναι το pipeline ώστε να παρέχει μια εις βάθος ανάλυση της διαμόρφωσης για να τονίσει τη δομή και τη λειτουργικότητά της, με όλα τα στάδια και τις εργασίες που αποτελούν το συνολικότερο pipeline. Αυτό το αρχείο καταλήγει να επιτρέπει την απρόσκοπτη αυτοματοποίηση από την δέσμευση κώδικα έως την ανάπτυξη όλων των συστημάτων του λογισμικού και τις υποδομής, εξασφαλίζοντας την συνεχή ενοποίηση και παράδοση.

---

<sup>3</sup>Η YAML είναι μια αναγνώσιμη από τον άνθρωπο γλώσσα σειριοποίησης δεδομένων που χρησιμοποιείται για τη σύνταξη αρχείων διαμόρφωσης

### .gitlab-ci.yml : Template, variables, stages



Εικόνα 34. Διάγραμμα .gitlab-ci.yml : Template, variables, stages

#### Εισαγωγή Template :

- **include** : Αρχικά είναι σημαντικό να αναφέρουμε πως η λέξη 'include' χρησιμοποιείται για να συμπεριλάβει το εξωτερικό πρότυπο .gitlab-ci.yml. Αυτό βοηθά στη διατήρηση της κύριας διαμόρφωσης CI/CD DRY (Don't Repeat Yourself) επαναχρησιμοποιώντας τις κοινόχρηστες διαμορφώσεις.
- **template** : Terraform/Base.latest.gitlab-ci.yml: Αυτή η γραμμή περιλαμβάνει ένα προκαθορισμένο πρότυπο CI/CD GitLab για το Terraform, το οποίο περιέχει τις κοινές εντολές και διαμορφώσεις Terraform (init, plan, apply). Είναι σημαντικό να διασφαλίσουμε την σωστή αναφορά αυτού του πρωτύπου, ώστε το GitLab να μπορεί να βρει και να εφαρμόσει το πρότυπο.

#### Ορισμός μεταβλητών :

- **variables** :

**IGNORE\_TF\_DEPRECATED\_WARNING** : Αυτή η μεταβλητή έχει οριστεί σε 'true' για να καταργήσει τις προειδοποιήσεις κατάργησης Terraform, διασφαλίζοντας ότι δεν προκαλούν αποτυχία του pipeline.

**TF\_CLI\_ARGS\_plan** : Αυτή η μεταβλητή καθορίζει ορίσματα για την εντολή terraform plan. Χρησιμοποιεί το αρχείο vars/\${CI\_ENVIRONMENT\_NAME}.tfvars για να παρέχει τις μεταβλητές που έχουμε αρχικοποιήσει, επιτρέποντας διαφορετικές διαμορφώσεις ανάλογα με τα στάδια.

**IMAGE\_NAME** : Αυτή η μεταβλητή καθορίζει το όνομα του Docker image που πρόκειται να κατασκευαστεί και να χρησιμοποιηθεί. Στην δικιά μας περιπτωση έχει ρυθμιστεί στο katsdros/test που είναι και το path του repository στο docker hub.

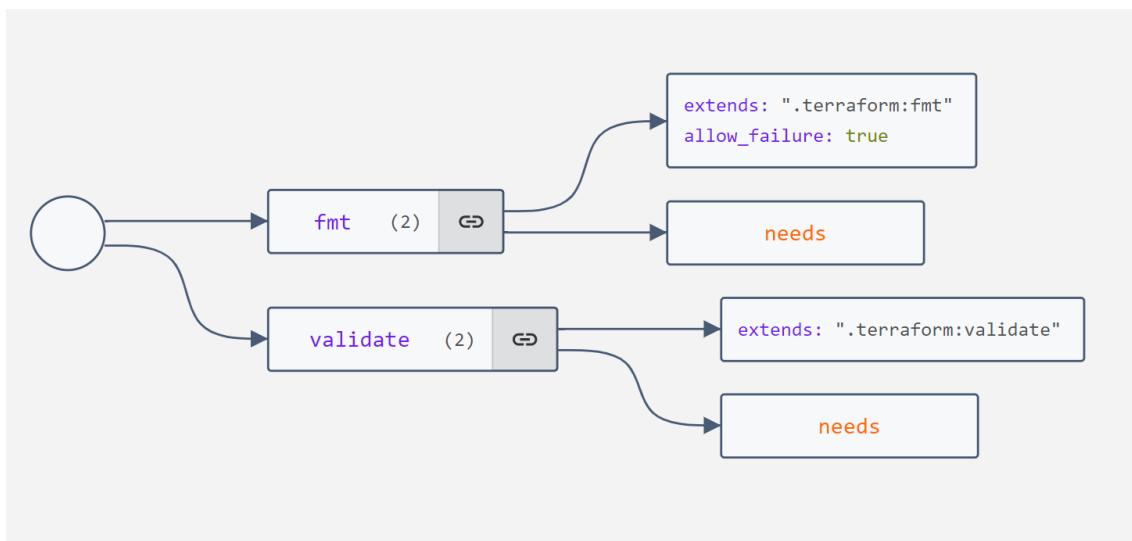
**IMAGE\_TAG** : Συνεχίζοντας έχουμε ορίσει το tag για το Docker image. Το έχουμε ορίσει σε python-app-1.0, το οποίο αντιστοιχεί στην έκδοση της εφαρμογής.

### Στάδια :

- validate : Αυτό το στάδιο είναι για την επικύρωση διαμορφώσεων Terraform και σύνταξης συνολικά του YAML αρχείου.
- test : Σε αυτό το στάδιο θα εκτελεστούν αυτοματοποιημένες δοκιμές για να επαληθεύσει το pipeline ότι ο κώδικας λειτουργεί όπως αναμένεται και έχει σχεδιαστεί.
- build: Εδώ, δημιουργείται το Docker image για την εφαρμογή. Οι μεταβλητές που έχουμε IMAGE\_NAME, IMAGE\_TAG και ορίστηκαν νωρίτερα χρησιμοποιούνται εδώ. Θα δούμε παρακάτω, πιο αναλυτικά πως χρησιμοποιούνται.
- deploy : Τέλος σε αυτό το στάδιο ο αγωγός χειρίζεται την ανάπτυξη της εφαρμογής στην πλατφόρμα του Linode. Επίσης χρησιμοποιεί το Docker image και εκτελεί τις απαραίτητες εντολές για την ανάπτυξη της εφαρμογής.

Συνοπτικά, το αρχικό απόσπασμα του configuration για το pipeline, ξεκινάει να δημιουργήσει το Template που θα χρησιμοποιήσουμε, τις αρχικοποιήσεις των μεταβλητών, καθώς και τα στάδια που θα έχει αυτός ο αγωγός κατά την εκτέλεση του. Τώρα πάμε να δούμε σε σχηματικό διάγραμμα πάλι τα στάδια με την σειρά και πως πρακτικά λειτουργεί το CI/CD pipeline.

### .gitlab-ci.yml : Validate stage



Εικόνα 35. Διάγραμμα .gitlab-ci.yml : Validate stage

Στη διαμόρφωση CI/CD του GitLab, το extends χρησιμοποιείται για να κληρονομήσει τη διαμόρφωση από προκαθορισμένες εργασίες. Αυτές οι προκαθορισμένες εργασίες μπορούν να οριστούν στο παρεχόμενο πρότυπο (Terraform/Base.latest.gitlab-ci.yml) Παρακάτω βλέπουμε πως διαμορφώνονται τα jobs: **fmt** και **validate** :

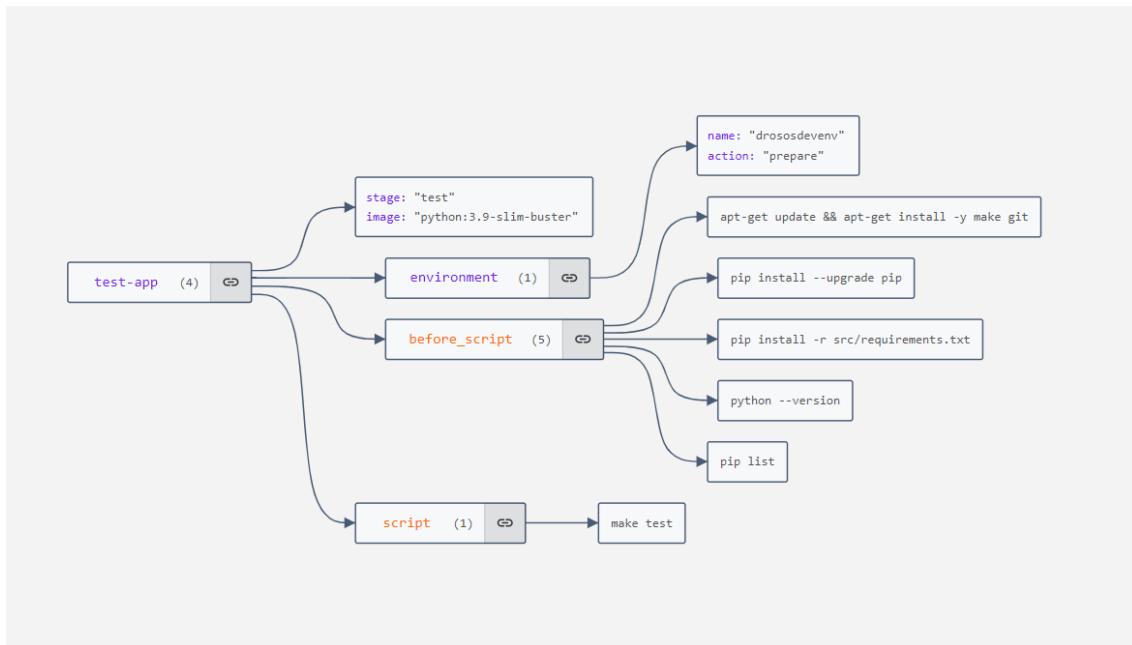
### extends: .terraform

Έδω, υποδεικνύεται ότι το job: validate κληρονομεί τη διαμόρφωση από μια προκαθορισμένη εργασία που ονομάζεται .terraform:validate. Η προκαθορισμένη εργασία .terraform:validate περιλαμβάνει όλα τα απαραίτητα βήματα και σενάρια για το validate της διαμόρφωσης Terraform. Συνεπώς, περιλαμβάνει ελέγχους σύνταξης για σφάλματα, ζητήματα διαμόρφωσης στο configuration και άλλα βήματα χωρίς φυσικά να αλληλεπιδρά με άλλες απομακρυσμένες υπηρεσίες, για να διασφαλιστεί ότι τα αρχεία Terraform είναι σωστά και έτοιμα για εφαρμογή. Αυτό το στάδιο βοηθάει αρκετά ώστε να αντιμετωπιστεί κατά τον κύκλο του αγωγού έγκαιρα τα ζητήματα που τυχόν να προκύψουν, εμποδίζοντας με αυτόν τον τρόπο configuration που δεν είναι σωστό.

### needs: []

Σε αυτό το σημείο, έχουμε την λέξη-κλειδί needs, όπου ορίζει τις εξαρτήσεις μεταξύ των εργασιών. Στο validate stage έχει οριστεί ένας κενός πίνακας, που σημαίνει ότι το job δεν εξαρτάται από κανένα άλλο job για να πραγματοποιηθεί η εκτέλεση. Συνεπώς αυτό διασφαλίζει ότι το validate stage εκτελείται ανεξάρτητα και δεν περιμένει να ολοκληρωθούν άλλα stages ή jobs πριν ξεκινήσει.

### .gitlab-ci.yml : test-app



Εικόνα 36. Διάγραμμα .gitlab-ci.yml : test stage

- **stage** : Με αυτό τον τρόπο γίνεται ανάθεση στο στάδιο δοκιμής για το pipeline.
- **image** : Χρησιμοποιούμε το docker image "python:3.9-slim-buster" , όπου είναι μια mini έκδοση της Python 3.9 και θα χρησιμοποιηθεί για το job:test-app που είναι στο test stage του pipeline.
- **environment** : Ονομάζουμε το περιβάλλον ως drososdevenv και με το **action** το αναθέτουμε ως prepare ώστε να ξεκινήσει να προετοιμαστεί το περιβάλλον για την εργασία.
- **services** : Χρησιμοποιούμε την υπηρεσία Docker-in-Docker (dind) - docker:20.10.16-dind, για την εκτέλεση των εντολών Docker μέσα στην εργασία.
- **before\_script** :

#### **Ενημέρωση και εγκατάσταση εξαρτήσεων:**

`apt-get update && apt-get install -y make git`: Ο στόχος είναι μέσα από αυτή την εντολή να ενημερώσει όλες τις λίστες πακέτων και να κάνει εγκατάσταση το 'make' και το 'git'

#### **Ενημέρωση του 'pip':**

`pip install --upgrade pip` : Ο σκοπός εδώ είναι να διασφαλίσει πως το 'pip' θα ενημερωθεί στην πιο πρόσφατη έκδοση του.

#### **Εγκατάσταση των python εξαρτήσεων:**

`pip install -r src/requirements.txt`: με αυτή την εντολή τώρα, είναι σε θέση να κάνει εγκατάσταση όλα τα απαιτούμενα packages της Python που έχουν καταγραφεί στο requirements.txt.

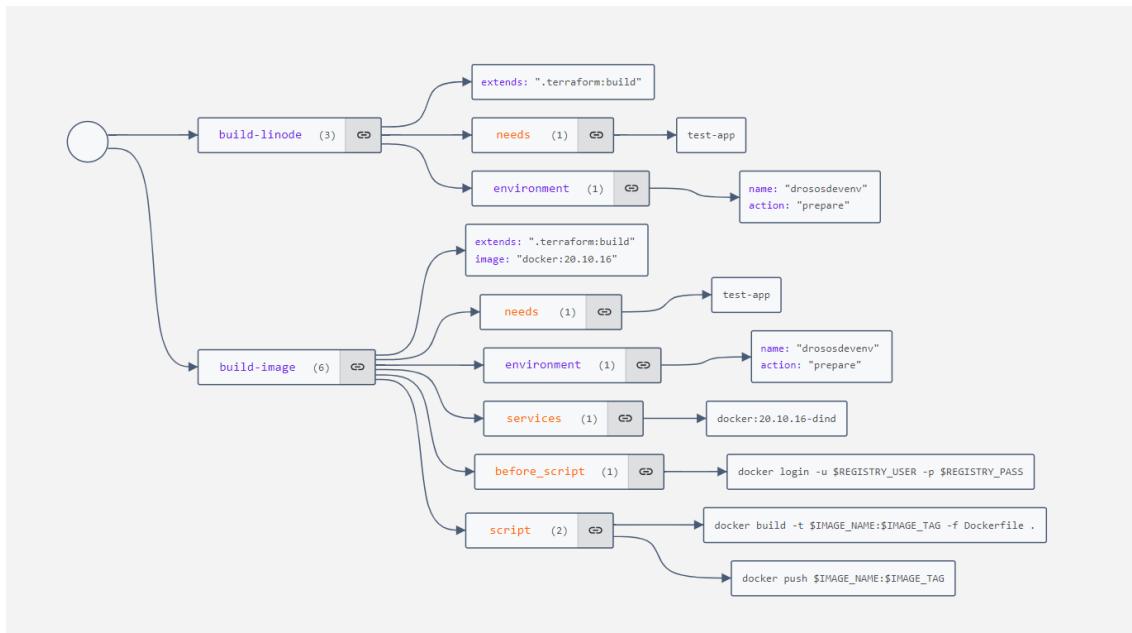
#### **'Ελεγχος έκδοσης της Python:**

`python --version`: δίνει κατά την διάρκεια της εκτέλεσης αυτού του job μέσα στο CI/CD pipeline την έκδοση Python για να γίνει επαλήθευση ότι έχει σωστή ρύθμιση.

- **script** :
- `make test`: σε αυτό το σημείο εκτελεί την εντολή 'make test' για να γίνουν unit tests όπως έχουν οριστεί στην εφαρμογή που έχουμε υιοθετήσει μέσα στο αρχείο Makefile.

Είναι σημαντικό να αναφέρουμε πως το job:test-app στο CI/CD pipeline είναι ένα κρίσιμο στάδιο που κύριο στόχο έχει να διασφαλίσει την ποιότητα και την λειτουργικότητα του κώδικα μέσω αυτοματοποιημένων δοκιμών. Όπως είδαμε αυτό το job χρησιμοποιεί ένα Docker image που έχει το Python 3.9-slim-bluster για να μπορέσει να δημιουργήσει ένα συνεπές και απομονωμένο περιβάλλον για την εκτέλεση όλων των δοκιμών. Σημαντικό πως, πριν την εκτέλεση τους στο σημείο του 'before\_script' εκτελούνται αρκετά βήματα όπως είδαμε προηγουμένως, και παραθέτη μια λίστα με τα εγκατεστημένα πακέτα για να διασφαλιστεί ότι το περιβάλλον έχει ρυθμιστεί σωστά. Τέλος, η στιγμή που εκτελείται το make test με όλα τα τεστ που έχουν ανατεθεί μέσα στο Makefile, μας επιτρέπει την αυτοματοποιημένη δοκιμή που είναι ζωτικής σημασίας για τη διατήρηση της ακεραιότητας του κώδικα και τον εντοπισμό τυχόν προβλημάτων νωρίς στην διαδικασία ανάπτυξης.

### .gitlab-ci.yml : build stage



Εικόνα 37. Διάγραμμα .gitlab-ci.yml : build stage

Όπως παρατηρούμε από το σχεδιάγραμμα, αρκετές διαδικασίες παραμένουν ίδιες στο build stage μέσα στο pipeline, απλώς έχει γίνει διαχωρισμός σε δύο jobs, ένα για να γίνει build το Linode VM χρησιμοποιώντας τα Terraform configuration files μέσα από το template που χρησιμοποιούμε, και ένα job για να γίνει build η εφαρμογή. Πιο συγκεκριμένα έχουμε:

#### build-linode

- Όπως αναλύσαμε συνοπτικά στο 5.3.2, τώρα το extends είναι το **.terraform:build**.
- Προσθέτουμε την επιλογή του needs όπως έχουμε πει και στο 5.3.2, αλλά αυτή την φορά δίνουμε μια ιεραρχία στο pipeline καθώς χρειάζεται να έχει τελειώσει επιτυχώς το test stage.

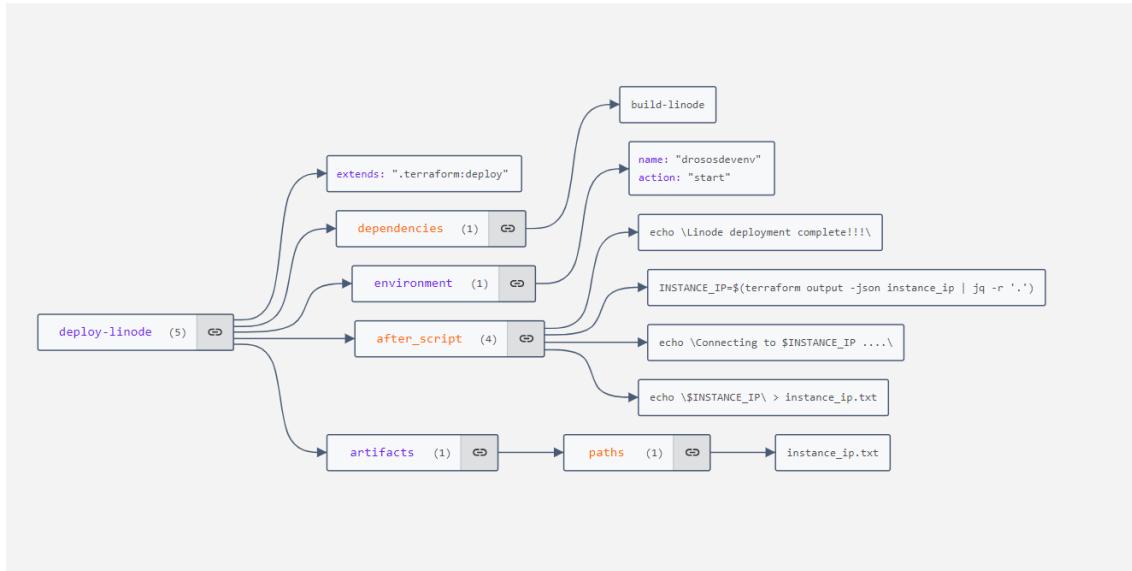
#### build-image

- Πρακτικά στο job του build-image, παραμένουν όλα ίδια όσον αφορά την χρήση του template, το περιβάλλον drososdevenv που έχουμε φτιάξει, καθώς και στα needs, όπως είπαμε και πιο πάνω, χρησιμοποιούμε το service πάλι docker:20.10.16 dind και πριν το script κάνουμε το login με την βοήθεια των μεταβλητών που έχουμε αρχικοποιήσει. Αυτό που προσθέτουμε, όταν φτάνει το job στο να εκτελέσει το script είναι μια εντολή :

`docker push $IMAGE_NAME:$IMAGE_TAG` : Κάνει push το image στο Docker registry.

Συνεχίζοντας στο deploy stage, βλέπουμε πάλι πως αρκετές διαδικασίες έχουν παραμένει ίδιες, έχει χωριστεί το συγκεκριμένο στάδιο του pipeline σε δύο jobs, ένα είναι το **deploy-linode** που καλείται να δημιουργήσει στον cloud provider την εικονική μηχανή με όλες τις προδιαγραφές, και ένα δεύτερο job, που είναι το **deploy-image** που καλείται αυτοματοποιημένα, να κάνει login στο vm που δημιουργήσαμε στην πλατφόρμα του Linode και να πραγματοποιήσει το deploy της εφαρμογής μας.

### .gitlab-ci.yml : deploy-linode



Εικόνα 38. Διάγραμμα .gitlab-ci.yml : deploy-linode stage

- Όπως αναλύσαμε συνοπτικά στο 5.3.2, τώρα το extends είναι το **.terraform:deploy**.
- Στο αρχείο του pipeline, θα προσθέσουμε ένα dependencies. Ετοι σε αυτό το job του deploy stage έχουμε βάλει ως **dependencies** : build-linode, που σημαίνει πως ιεραρχικά πρέπει πρώτα να έχει τελειώσει το job: build-linode του build stage, ώστε να μπορέσει μετά το CI/CD pipeline να προχωρήσει στο deploy του Linode VM.
- Αφού τελειώσει το deploy του VM, μετά σειρά έχει το **after\_script**. Σε αυτό το βήμα, πρακτικά καλείται το pipeline μέσα από εντολές να κάνει extract την ip address του virtual machine:

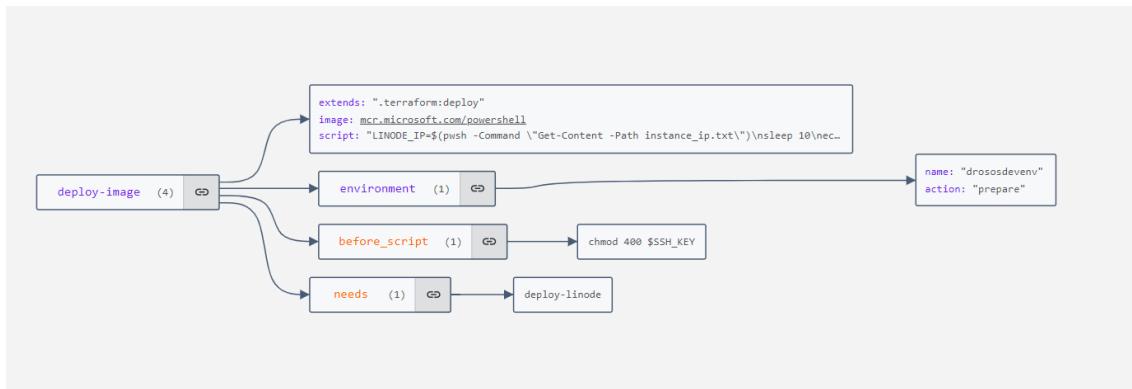
`INSTANCE_IP=$(terraform output -json instance_ip | jq -r '.')`<sup>4</sup>  
κάνει εξαγωγή της instance ip διεύθυνσης από το Terraform output και μεταφέρει αυτή την τιμή σε μια μεταβλητή με όνομα INSTANCE\_IP.

`echo "$INSTANCE_IP" > instance_ip.txt`: Αποθηκεύει την IP διεύθυνση σε ένα αρχείο κειμένου με το όνομα instance\_ip.txt .

- artifacts** : αρχικοποιώντας ως paths: instance\_ip.txt, πρακτικά καθορίζουμε πως θα πρέπει να συλλέγεται ως artifact, ώστε να μπορεί να είναι διαθέσιμο για λήψη μετά την ολοκλήρωση του job.

<sup>4</sup><https://developer.hashicorp.com/terraform/cli/commands/output>

### .gitlab-ci.yml : deploy-image



Εικόνα 39. Διάγραμμα .gitlab-ci.yml : deploy-image stage

- Και εδώ όπως βλέπουμε στο διάγραμμα από πάνω, το extends και εδώ είναι το **.terraform:deploy**.
- Και εδώ θα προσθέσουμε την επιλογή του **needs** όπως έχουμε πει και στο 5.3.2, αλλά αυτή την φορά δίνουμε μια ιεραρχία στο pipeline καθώς χρειάζεται να έχει τελειώσει επιτυχώς το deploy-linode.
- **image**: mcr.microsoft.com/powershell, χρησιμοποιούμε ένα docker image για το powershell ώστε να μπορέσουμε με powershell commands να κάνουμε extract την IP διεύθυνση του vm που έχουμε δημιουργήσει ως υποδομή.
- **before\_script**: chmod 400 \$SSH\_KEY  
Πριν τρέξει το script για το job: deploy-image, αλλάζουμε τα δικαιώματα του SSH key, ώστε να είναι μόνο για ανάγνωση από τον κάτοχο, διασφαλίζοντας ότι είναι ασφαλές.
- **script**:  
LINODE\_IP=\$(pwsh -Command "Get-Content -Path instance\_ip.txt")  
με αυτή την Powershell εντολή, καθώς μπορούμε να χρησιμοποιήσουμε powershell εντολές λόγω του ότι σε αυτό το job χρησιμοποιούμε ένα docker image, πρακτικά ανακτά τη διεύθυνση IP του Linode VM από το αρχείο instance\_ip.txt και το τοποθετεί στην μεταβλητή LINODE\_IP.  
ssh -o StrictHostKeyChecking=no -i \$SSH\_KEY root@\${LINODE\_IP}  
και χρησιμοποιώντας το ssh πρωτόκολλο και την μεταβλητή που περιέχει την IP, συνδεόμαστε αυτόματα στο Linode VM και με τις παρακάτω εντολές κάνουμε deploy την εφαρμογή:  
docker login -u \$REGISTRY\_USER -p \$REGISTRY\_PASS  
docker run -d -p 5009:5009 \$IMAGE\_NAME:\$IMAGE\_TAG

Σε αυτήν την υποενότητα, μπορούμε να αναφέρουμε πως αναλύσαμε σχολαστικά το CI/CD pipeline, καθώς περιγράφει την υλοποίηση του για την ανάπτυξη εφαρμογής και της υποδομής με χρήση ένα Linode VM, χρησιμοποιώντας Terraform, Docker και GitLab CI/CD. Κλείνοντας, ο αγωγός παρουσιάζει σύγχρονες και βέλτιστες πρακτικές ανάπτυξης λογισμικού, διασφαλίζοντας αποτελεσματικότητα, επαναληψιμότητα και επεκτασιμότητα, που ήταν και ο στόχος της διπλωματικής εργασίας.

### Πλαίσιο εφαρμογής

Σε αυτό το σημείο και θέλοντας να υπάρχει σαφήνεια της διπλωματικής εργασίας, ο στόχος ήταν να υπάρξει πλήρης ανάλυση στο θεωριτικό κομμάτι για την έννοια του Software Development LifeCycle, του Infrastructure as Code αλλά και των εργαλείων που υπάρχουν, καθώς και της τεκμιρωμένης αποτύπωσης για την έννοια του DevOps αλλά και των τεχνικών CI/CD pipelines. Στο πρακτικό μέρος λοιπόν, εφαρμόσαμε όλα όσα είδαμε στην έρευνα που αποτυπώσαμε σε προηγούμενα κεφάλαια, με κύριο στόχο μέσα από το Terraform ως Iac εργαλείο να κάνουμε provision την υποδομή που χρειαζόμαστε, και μέσα από ένα καλά δομημένο CI/CD pipeline να ερευνήσουμε το πως λειτουργεί όλο το workflow για να επιτύχουμε την αυτοματοποιημένη διαδικασία σε όλα τα στάδια.

Γι' αυτό τον λόγο η εφαρμογή που χρησιμοποιήσαμε για την επίδειξη όλων των παραπάνω προέρχεται από το διαδίκτυο και χρησιμεύει για την απεικόνιση της λειτουργικότητας του CI/CD pipeline που έχουμε δημιουργήσει. Συνεπώς, πρωταρχικός στόχος της διπλωματικής εργασίας είναι η υλοποίηση και η αυτοματοποίηση των διαδικασιών CI/CD, παρά η ίδια η εφαρμογή. Το Dockerfile και ο κώδικας της εφαρμογής περιλαμβάνεται στην βιβλιογραφία [29].

## 5.4 Use case σενάριο

Σε αυτήν την ενότητα, θα παρουσιάσουμε το πρακτικό σενάριο όπου εφαρμόζεται στο CI/CD pipeline που έχουμε δημιουργήσει και πως λειτουργούν σε κάθε στάδιο όλα τα jobs μέχρι να φτάσουμε στο deploy της εφαρμογής. Είναι σημαντικό να αναφέρουμε πως αυτή η περίπτωση χρήσης υπογραμμίζει την συνολική αποτελεσματικότητα του αγωγού και την αυτοματοποίηση που επιτυγχάνεται μέσω της υλοποίησης.

### 5.4.1 Use case σενάριο: Automated deploy of web app

Αρχικά χρειάζεται να καταγράψουμε συνοπτικά τα βήματα που γίνονται για την ανάπτυξη της εφαρμογής σε ένα Linode VM με ελάχιστη χειροκίνητη παρέμβαση και χρησιμοποιώντας το CI/CD pipeline μέσα από την πλατφόρμα του GitLab.

- **Code commit** : Ο προγραμματιστής πρακτικά, αυτό που κάνει είναι να δεσμεύσει τις αλλαγές που είναι να κάνει στο Git repository.
- **CI/CD pipeline trigger** : Η δέσμευση που κάνει, αυτόματα ενεργοποιεί τον CI/CD pipeline ξεκινώντας με την σειρά τα στάδια validate και test που έχουν δομηθεί στο .gitlab-ci.yml αρχείο.
- **Build stage** : Μετέπειτα και εφόσον οι δοκιμές είναι επιτυχής, σειρά πέρνει το build stage χωρίζεται σε δύο jobs: ένα που δημιουργεί το Linode VM μέσα από τα Terraform Configuration files και ένα που δημιουργεί ένα Docker image της εφαρμογής

- **Deploy stage** : Μετά, ενώ έχει ολοκληρωθεί το build stage, συνεχίζει με το deploy stage που χωρίζεται και εκείνο σε δύο jobs: ένα που παρέχει την εικονική μηχανή στο Linode χρησιμοποιώντας το Terraform και ένα που αναπτύσσει το Docker image μέσα στο Linode VM.
- **Validation** : Αφού ολοκληρωθεί όλο το CI/CD pipeline, είναι η στιγμή που γίνεται επαλήθευση, ότι δηλαδή εκτελείται σωστά στο Linode VM η εφαρμογή.

### 5.4.2 CI/CD pipeline - workflow

Για να μπορέσουμε να κατανοήσουμε και να δώσουμε μια λεπτομερή εικόνα για το πως λειτουργεί το CI/CD pipeline, θα προχωρήσουμε στην πραγματική εκτέλεση του αγωγού, παρουσιάζοντας σε κάθε stage και job ποια είναι η έξοδο και τα αποτελέσματα του.

#### Validate stage

```

validate
Passed Started 6 hours ago by Drosos Katsimpris
Search job log
1 Running with gitlab-runner 17.0.0-pre.88.g761ae5dd (761ae5dd)
2 on green-4.saas-linux-small-amd64.runners-manager.gitlab.com/default ntHFEtyX, system ID: a_8990de21c550
3 Preparing the "docker-machine" executor
4 Using Docker executor with image registry.gitlab.com/gitlab-org/terraform-images/stable:latest ...
5 Authenticating with credentials from job payload (GitLab Registry)
6 Pulling docker image registry.gitlab.com/gitlab-org/terraform-images/stable:latest ...
7 Using docker image sha256:abb83ff04190f9822ffd23e776138e00e954643fbef3d1073a0c765109019c2c for registry.gitlab.com/gitlab-org/terraform-images/stable:latest with digest registry.gitlab.com/gitlab-org/terraform-images/stable@sha256:d5b421ec092dd6ec67d2192b914af33e15db1fe5b07c9d543c11510800b4132e0c ...
8 Preparing environment
9 Running on runner-nthfetyx-project-59600889-concurrent-0 via runner-nthfetyx-s-1-s-amd64-1721469880-ba92c05e...
10 Getting source from Bit repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/drososkats/gitlab-ci-cd-course/.git/
13 Created fresh repository.
14 Checking out a4efc47e as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Restoring cache
18 Checking cache for /builds/drososkats/gitlab-ci-cd-course-protected...
19 Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59600889/builds/drososkats/gitlab-ci-cd-course-protected
20 Successfully extracted cache
21 Executing "step_script" stage of the job script
22 Using docker image sha256:abb83ff04190f9822ffd23e776138e00e954643fbef3d1073a0c765109019c2c for registry.gitlab.com/gitlab-org/terraform-images/stable:latest with digest registry.gitlab.com/gitlab-org/terraform-images/stable@sha256:d5b421ec092dd6ec67d2192b914af33e15db1fe5b07c9d543c11510800b4132e0c ...
23 $ terraform --version
24 Terraform v1.5.7
25 on linux_amd64
26 Your version of Terraform is out of date! The latest version
27 is 1.9.2. You can update by downloading from https://www.terraform.io/downloads.html
28 Saving cache for successful job
29 Creating cache /builds/drososkats/gitlab-ci-cd-course-protected...
30 /builds/drososkats/gitlab-ci-cd-course/.terraform/: found 11 matching artifact files and directories
31 Archive is up to date!
32 Created cache
33 Cleaning up project directory and file based variables
34 Job succeeded

```

Εικόνα 40. Validate stage - report

Παρατηρούμε πως μετά την πραγματοποίηση των αλλαγών ή προσθηκών που έγιναν στον κώδικα, ενεργοποιείται το CI/CD pipeline. Στο validate stage, γίνεται ο έλεγχος της σύνταξης και της δομής των αρχείων Terraform. Και παρατηρούμε πως το αποτέλεσμα είναι να ολοκληρωθεί με επιτυχία το job, διασφαλίζοντας ότι τα αρχεία διαμόρφωσης είναι έγκυρα.

## Test stage

The screenshot shows a GitLab CI job log for a project named 'test-app'. The job has passed and started 6 hours ago by user 'Drosos Katsimpris'. The log details the steps taken during the build:

```

1 Running with gitlab-runner 17.0.8-pre.88.g76iae5dd (76iae5dd)
2 on green-1.saa5-linux-small-amd64.runners-manager.gitlab.com/default JLguopmM, system ID: s_deaa2ca89de7
3 Preparing the "docker-machine" executor
4 Using Docker executor with image python:3.9-slim-buster ...
5 Pulling docker image python:3.9-slim-buster ...
6 Using docker image sha256:c84d0fe3b3b8deeb39e17d121220107f8354a9883b468a320a77788cd128f11c87 for python:3.9-slim-buster with digest python@sha256:320a7a4250aba4249f458872adecf92ea88dc6cab2d76dc5c6ff61cac953998 ...
7 Preparing environment
8 Running on runner-jlguopmM-project-59608889-concurrent-0 via runner-jlguopmM-s-l-s-amd64-1721469881-24997c1b...
9 Getting source from Git repository
10 Fetching changes with git depth set to 20...
11 Initialized empty Git repository in /builds/drososkats/gitlab-ci-cd-course/.git/
12 Created fresh repository.
13 Checking out a4efef07e as detached HEAD (ref is main)...
14 Skipping Git submodules setup
15 $ git remote set-url origin "${CI_REPOSITORY_URL}"
16 Restoring cache
17 Checking cache for /builds/drososkats/gitlab-ci-cd-course-protected...
18 Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/59608889/builds/drososkats/gitlab-ci-cd-course-protected
19 Successfully extracted cache
20 Executing "step_script" stage of the job script
21 Using docker image sha256:c84d0fe3b3b8deeb39e17d121220107f8354a9883b468a320a77788cd128f11c87 for python:3.9-slim-buster with digest python@sha256:320a7a4250aba4249f458872adecf92ea88dc6cab2d76dc5c6ff61cac953998 ...
22 $ apt-get update && apt-get install -y make git
23 Get:1 http://deb.debian.org/debian buster InRelease [122 kB]
24 Get:2 http://deb.debian.org/debian-security buster/updates InRelease [34.8 kB]
25 Get:3 http://deb.debian.org/debian buster-updates InRelease [56.6 kB]
26 Get:4 http://deb.debian.org/debian/buster/main amd64 Packages [7989 kB]
27 Get:5 http://deb.debian.org/debian-security buster/updates/main amd64 Packages [610 kB]
28 Get:6 http://deb.debian.org/debian buster-updates/main amd64 Packages [8788 B]

```

Eικόνα 41. Test stage - report [1]

The screenshot shows a GitLab CI job log for a project named 'Gitlab CiCd Course'. The job has passed and started 6 hours ago by user 'Drosos Katsimpris'. The log details the steps taken during the build:

```

259 Setting up xauth (1:1.8.10-1) ...
260 Processing triggers for libc-bin (2.28-10+deb10u2) ...
261 $ pip install --upgrade pip
262 Requirement already satisfied: pip in /usr/local/lib/python3.9/site-packages (23.0.1)
263 Collecting pip
264   Downloading pip-24.1.2-py3-none-any.whl (1.8 MB)
265     ┌─────────────────────────────────────────────────────────────────────────┐
265     │          1.8/1.8 MB 20.7 MB/s eta 0:00:00
265     └─────────────────────────────────────────────────────────────────────────┘
266 Installing collected packages: pip
267   Attempting uninstall: pip
268     Found existing installation: pip 23.0.1
269     Uninstalling pip-23.0.1:
270       Successfully uninstalled pip-23.0.1
271 Successfully installed pip-24.1.2
272 WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
273 $ pip install -r src/requirements.txt
274 Collecting Flask==1.1.2 (from -r src/requirements.txt (line 1))
275   Downloading Flask-1.1.2-py3-none-any.whl.metadata (4.6 kB)
276 Collecting Werkzeug==1.0.1 (from -r src/requirements.txt (line 2))

```

Eικόνα 42. Test stage - report [2]

```
392 $ python --version
393 Python 3.9.17
394 $ pip list
395 Package           Version
396 -----
397 appdirs            1.4.4
398 attrs              23.2.0
399 black              20.8b1
400 click              8.1.7
401 flake8             3.9.0
402 Flask              1.1.2
403 gunicorn            20.1.0
404 configparser        2.0.0
405 itsdangerous        1.1.0
406 Jinja2              2.11.3
407 MarkupSafe          1.1.1
408 mccabe              0.6.1
409 mypy-extensions      1.0.0
410 packaging            24.1
411 pathspec            0.12.1
412 pip                  24.1.2
413 pluggy              0.13.1
414 psutil              5.8.0
415 py                   1.11.0
416 py-cpuinfo           7.0.0
417 pycodestyle          2.7.0
418 pyflakes             2.3.1
419 pytest               6.2.2
420 regex                 2024.5.15
421 setuptools            58.1.0
422 toml                 0.18.2
423 typed-ast             1.5.5
424 typing_extensions     4.12.2
425 Werkzeug             1.0.1
426 wheel                 0.40.0
427 $ make test
428 python3 -m venv src/.venv
429 . src/.venv/bin/activate; pip install -Ur src/requirements.txt
```

Εικόνα 43. Test stage - report [3]

```
506 && pytest -v
507 ===== test session starts =====
508 platform linux -- Python 3.9.17, pytest-6.2.2, py-1.11.0, pluggy-0.13.1 -- /builds/drososkats/gitlab-cicd-course/src/.venv/bin/python3
509 cachedir: .pytest_cache
510 rootdir: /builds/drososkats/gitlab-cicd-course
511 collecting ... collected 5 items
512 src/app/tests/test_api.py::test_api_process PASSED [ 20%]
513 src/app/tests/test_api.py::test_api_monitor PASSED [ 40%]
514 src/app/tests/test_views.py::test_home PASSED [ 60%]
515 src/app/tests/test_views.py::test_page_content PASSED [ 80%]
516 src/app/tests/test_views.py::test_info PASSED [100%]
517 ===== 5 passed in 2.19s =====
518 Saving cache for successful job
519 Creating cache /builds/drososkats/gitlab-cicd-course-protected...
520 /builds/drososkats/gitlab-cicd-course/.terraform/: found 11 matching artifact files and directories
521 Archive is up to date!
522 Created cache
523 Cleaning up project directory and file based variables
524 Job succeeded
```

#### Εικόνα 44. Test stage - report [4]

Σε αυτό σημείο, από τα 4 screenshots παρατηρούμε πως στο Test stage του CI/CD pipeline εκτελείται το job: test-app, κάνοντας αυτοματοποιημένες δοκιμές για να διασφαλίσει την ποιότητα του κώδικα. Στο τελευταίο screenshot βλέπουμε πως το job έχει κάνει όλο το installation στα requirements που χρειάζεται η εφαρμογή, έχει εκτελέσει τα απαραίτητα tests και φαίνεται ότι είναι **'job succeeded'**.

## Build stage

**build-linode**

Passed Started 7 hours ago by Drosos Katsimpris

Search job log

```
1 Running with gitlab-runner 17.0.0-pre.88.g761ae5dd (761ae5dd)
2 on green-3.saas-linux-small-amd64.runners-manager.gitlab.com/default Jhc_jxvh, system ID: s_8e6850b2bcce
3 Preparing the "docker-machine" executor
4 Using Docker executor with image registry.gitlab.com/gitlab-org/terraform-images/stable:latest ...
5 Authenticating with credentials from job payload (GitLab Registry)
6 Pulling docker image registry.gitlab.com/gitlab-org/terraform-images/stable:latest ...
7 Using docker image sha256:abb03ff0a4190f9822ff23e776138e00e054643fbef3d1073a0c765109819c2c for registry.gitlab.com/gitlab-org/terraform-images/stable:latest with digest registry.gitlab.com/gitlab-org/terraform-images/stable@sha256:d5b621ec092ddec67d2192b914af3e15db1fe5b7c9d543c1151080b4132e0c ...
8 Preparing environment
9 Running on runner-jhcjxvh-project-5960889-concurrent-0 via runner-jhcjxvh-a-l-s-amd64-1721469817-3dc464ec...
10 Getting source from Git repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/drososkats/gitlab-cicd-course/.git/
13 Created fresh repository.
14 Checking out a6effe74e as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Restoring cache
18 Checking cache for /builds/drososkats/gitlab-cicd-course-protected...
19 Downloading cache from https://storage.googleapis.com/gitlab-com-runners-cache/project/5960889/builds/drososkats/gitlab-cicd-course-protected
20 Successfully extracted cache
21 Executing "step_script" stage of the job script
22 Using docker image sha256:abb03ff0a4190f9822ff23e776138e00e054643fbef3d1073a0c765109819c2c for registry.gitlab.com/gitlab-org/terraform-images/stable:latest with digest registry.gitlab.com/gitlab-org/terraform-images/stable@sha256:d5b621ec092ddec67d2192b914af3e15db1fe5b7c9d543c1151080b4132e0c ...
23 $ gitlab-terraform plan
24 Initializing the backend...
25 Initializing provider plugins...
26 - Finding linode/linode versions matching "1.30.0"...
27 - Installing linode/linode v1.30.0...
28 - Installed linode/linode v1.30.0 (signed by a HashiCorp partner, key ID F4E6BBD9EA4FE463)
```

Eukóva 45. Build stage: Linode VM - report [1]

```

35 you can `terraform init` in the future.
36 Terraform has been successfully initialized!
37 Terraform used the selected providers to generate the following execution
38 plan. Resource actions are indicated with the following symbols:
39 + create
40 Terraform will perform the following actions:
41 # linode_instance.drosos-tfvm01 will be created
42 + resource "linode_instance" "drosos-tfvm01" {
43     + authorized_keys = [
44         + "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCLLf89o6NyPh0LiEm7IWYLP9BlosdeI98HTpPPfuXJe+KRYXPDNVAT
K4C592wrMqd4sItt/L/9lNUgc2p/FSLNuajrZ0LgEKNmYpz2Kkd6VA/CcHYDI1t73MvR0kgvBnJQj8C7q9rVf+By+w/gq/d/5Va8ioJvu
Fquyq40beNG82D12su00mcA3H954V/5Sm0LYjDDYstMVHfyVBQSLjcnG0fpqIL6GA1fEpgFog42D8LnnN41gjfUP3E47oqQPvahYZu0/
VoB86Pc0GArz+VNz4n+H8n/ZIj0Ll1YsgrdGhjlhgBJfe19zjXzqP9a0hnFnsw2v2yz8c8K+/M4HqPr3HaKL6o18T8tFW+jixYwN7Chd
yBw6W2Krz02e5TWo0EfUJB0LggqeQw== cs131110@uniwa.gr",
45     ]
46     + backups      = (known after apply)
47     + backups_enabled = (known after apply)
48     + boot_config_label = (known after apply)
49     + booted       = (known after apply)
50     + host_uuid    = (known after apply)
51     + id           = (known after apply)
52     + image         = "linode/ubuntu22.04"
53     + ip_address   = (known after apply)
54     + ipv4          = (known after apply)
55     + ipv6          = (known after apply)
56     + label         = "drosos-develenv01"
57     + private_ip_address = (known after apply)
58     + region        = "us-central"
59     + resize_disk   = false
60     + root_pass     = (sensitive value)
61     + shared_ipv4   = (known after apply)
62     + specs          = (known after apply)
63     + status         = (known after apply)
64     + swap_size     = (known after apply)
65     + type          = "g6-standard-1"
66     + watchdog_enabled = true
67   }
68 Plan: 1 to add, 0 to change, 0 to destroy.
69 Changes to Outputs:
70   + instance_ip = (known after apply)

```

Εικόνα 46. Build stage: Linode VM - report [2]

```

69 Changes to Outputs:
70   + instance_ip = (known after apply)
71 $ gitlab-terrefrom plan-json
72 Saving cache for successful job
73 Creating cache /builds/drososkats/gitlab-cicd-course-protected...
74 /builds/drososkats/gitlab-cicd-course/.terraform/: found 11 matching artifact files and directories
75 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59600089/builds/drososkats
76 Created cache
77 Uploading artifacts for successful job
78 Cleaning up project directory and file based variables
89 Job succeeded

```

Εικόνα 47. Build stage: Linode VM - report [3]

Παρατηρούμε πως στο πρώτο job: build-linode του build stage, δημιουργείται αυτόματα το Linode VM μέσω του Terraform με όλες τις προδιαγραφές που έχουμε δώσει στα Terraform Configuration αρχεία, και μετέπειτα θα εκτελεστεί το δεύτερο job: build-image, όπως βλέπουμε παρακάτω:

**build-image**

Passed Started 11 hours ago by Drosos Katsimpras

```

1 Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
2   on green-6.saas-linux-small-amd64.runners-manager.gitlab.com/default YKxHNyexq, system ID: s_a201ab37b78a
> 3 Preparing the "docker+machine" executor
> 11 Preparing environment
> 13 Getting source from Git repository
> 20 Restoring cache
> 24 Executing "step_script" stage of the job script
25 Using docker image sha256:2a153cb5c7c52610ceb46876231f1c7ab8d2a8926aaeb5283994ef3d6f78def9 for docker:20.10.16
e33b4274fb68a1ade2d6c9da4 ...
26 $ docker login -u $REGISTRY_USER -p $REGISTRY_PASS
27 WARNING! Using --password via the CLI is insecure. Use --password-stdin.
28 WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
29 Configure a credential helper to remove this warning. See
30 https://docs.docker.com/engine/reference/commandline/login/#credentials-store
31 Login Succeeded

```

Εικόνα 48. Build stage: Docker image - report [1]

```

32 $ docker build -t $IMAGE_NAME:$IMAGE_TAG -f build/Dockerfile .
33 Step 1/11 : FROM python:3.9-slim-buster
34 3.9-slim-buster: Pulling from library/python
35 8b91b88d5577: Pulling fs layer
36 824416e23423: Pulling fs layer
37 8d53da260408: Pulling fs layer
38 84c8c79126f6: Pulling fs layer
39 2e1c130fa3ec: Pulling fs layer
40 84c8c79126f6: Waiting
41 2e1c130fa3ec: Waiting
42 824416e23423: Verifying Checksum
43 824416e23423: Download complete
44 8d53da260408: Verifying Checksum
45 8d53da260408: Download complete
46 8b91b88d5577: Verifying Checksum
47 8b91b88d5577: Download complete
48 84c8c79126f6: Verifying Checksum
49 84c8c79126f6: Download complete
50 2e1c130fa3ec: Verifying Checksum
51 2e1c130fa3ec: Download complete
52 8b91b88d5577: Pull complete
53 824416e23423: Pull complete
54 8d53da260408: Pull complete
55 84c8c79126f6: Pull complete
56 2e1c130fa3ec: Pull complete
57 Digest: sha256:320a7a4250aba4249f458872adecf92eea88dc6abd2d76dc5c0f01cac9b53990
58 Status: Downloaded newer image for python:3.9-slim-buster
59 ---> c84dbfe3b8de
60 Step 2/11 : LABEL Name="Python Flask Demo App" Version=1.4.2
61 ---> Running in 24690b165306

```

Εικόνα 49. Build stage: Docker image - report [2]

```

185 Step 11/11 : CMD ["gunicorn", "-b", "0.0.0.0:5009", "run:app"]
186 ---> Running in c9bb9120b1ba
187 Removing intermediate container c9bb9120b1ba
188 ---> 338742eac327
189 Successfully built 338742eac327
190 Successfully tagged katsdros/test:python-app-1.0
191 $ docker push $IMAGE_NAME:$IMAGE_TAG
192 The push refers to repository [docker.io/katsdros/test]
193 98afc4a79659: Preparing
194 dce5ff5a154d: Preparing
195 5a4da979d00d: Preparing
196 0ddede07efca: Preparing
197 1f6934520950: Preparing
198 067ea27560c1: Preparing
199 7fb1037e08b3: Preparing
200 14cbeede8d6e: Preparing
201 ae2d55769c5e: Preparing
202 e2ef8a51359d: Preparing
203 067ea27560c1: Waiting
204 7fb1037e08b3: Waiting
205 14cbeede8d6e: Waiting
206 ae2d55769c5e: Waiting
207 e2ef8a51359d: Waiting
208 dce5ff5a154d: Pushed
209 98afc4a79659: Pushed
210 1f6934520950: Pushed
211 0ddede07efca: Pushed
212 067ea27560c1: Layer already exists
213 7fb1037e08b3: Layer already exists
214 14cbeede8d6e: Layer already exists
215 ae2d55769c5e: Layer already exists
216 e2ef8a51359d: Layer already exists
217 5a4da979d00d: Pushed
218 python-app-1.0: digest: sha256:0b84ba0685adf7a7ddd4a8d336b6b66d68477ca535a63fe67a1b9c90f2d3fbb4 size: 2412
219 Saving cache for successful job
220 Creating cache /builds/drososkats/gitlab-ci-cd-course-protected...
221 /builds/drososkats/gitlab-ci-cd-course/.terraform/: found 11 matching artifact files and directories
222 Archive is up to date!
223 Created cache

```

Εικόνα 50. Build stage: Docker image - report [3]

```

217 5a4da979d00d: Pushed
218 python-app-1.0: digest: sha256:0b84ba0685adf7a7ddd4a8d336b6b66d68477ca535a63fe67a1b9c90f2d3fbb4 size: 2412
< 219 Saving cache for successful job
220 Creating cache /builds/drososkats/gitlab-ci-cd-course-protected...
221 /builds/drososkats/gitlab-ci-cd-course/.terraform/: found 11 matching artifact files and directories
222 Archive is up to date!
223 Created cache
> 224 Uploading artifacts for successful job
< 231 Cleaning up project directory and file based variables
232 Job succeeded

```

Εικόνα 51. Build stage: Docker image - report [4]

Σε αυτό το job του CI/CD pipeline, βλέπουμε πως η δημιουργία του Docker image για την εφαρμογή ολοκληρώθηκε με επιτυχία και έχει γίνει pushed στο docker repository. Τώρα το CI/CD pipeline είναι σε θέση να προχωρήσει στο τελευταίο στάδιο που είναι το deploy του Linode VM και το deploy του application.

## Deploy stage

deploy-linode

Passed Started 11 hours ago by Drosos Katsimpras

This job is deployed to drososdevenv.

```
1 Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
2   on green-4.saas-linux-small-amd64.runners-manager.gitlab.com/default ntHFEtyX, system ID: s_8990de21c550
> 3 Preparing the "docker+machine" executor
> 4 Preparing environment
> 5 Running on runner-nthfetyx-project-59600089-concurrent-0 via runner-nthfetyx-s-l-s-amd64-1721469276-33021bbd...
> 6 Getting source from Git repository
> 7 Restoring cache
> 8 Downloading artifacts
> 9 Executing "step_script" stage of the job script
10 Using docker image sha256:abb83ff04190f9822ffd23e776138e00e954643fbef3d1073a0c765109019c2c for registry.gitlab.com...
11 -images/stable@sha256:d5b621ec092dd6ec67d2192b914af33e15db1fe5b7c9d543c1151008b4132e0c ...
12 $ gitlab-terraform apply
13 Initializing the backend...
14 Initializing provider plugins...
15 - Finding linode/Linode versions matching "1.30.0"...
16 - Installing linode/Linode v1.30.0...
17 - Installed linode/Linode v1.30.0 (signed by a HashiCorp partner, key ID F4E68BD0EA4FE463)
18 Partner and community providers are signed by their developers.
19 If you'd like to know more about provider signing, you can read about it here:
20 https://www.terraform.io/docs/cli/plugins/signing.html
21 Terraform has created a lock file .terraform.lock.hcl to record the provider
22 selections it made above. Include this file in your version control repository
23 so that Terraform can guarantee to make the same selections by default when
24 you run "terraform init" in the future.
25 Terraform has been successfully initialized!
26 linode_instance.drosos-tfvm01: Creating...
27 linode_instance.drosos-tfvm01: Still creating... [10s elapsed]
28 linode_instance.drosos-tfvm01: Still creating... [20s elapsed]
29 linode_instance.drosos-tfvm01: Still creating... [30s elapsed]
30 linode_instance.drosos-tfvm01: Still creating... [40s elapsed]
31 linode_instance.drosos-tfvm01: Provisioning with 'remote-exec'...
32 linode_instance.drosos-tfvm01 (remote-exec): Connecting to remote host via SSH...
33 linode_instance.drosos-tfvm01 (remote-exec): Host: 45.33.9.167
34 linode_instance.drosos-tfvm01 (remote-exec): User: root
35 linode_instance.drosos-tfvm01 (remote-exec): Password: false
36 linode_instance.drosos-tfvm01 (remote-exec): Private key: true
37 linode_instance.drosos-tfvm01 (remote-exec): Certificate: false
38 linode_instance.drosos-tfvm01 (remote-exec): SSH Agent: false
```

Εικόνα 52. Deploy stage: Linode VM - report [1]

```

57 linode_instance.drosos-tfvm01 (remote-exec): Connecting to remote host via SSH...
58 linode_instance.drosos-tfvm01 (remote-exec):   Host: 45.33.9.167
59 linode_instance.drosos-tfvm01 (remote-exec):   User: root
60 linode_instance.drosos-tfvm01 (remote-exec):   Password: false
61 linode_instance.drosos-tfvm01 (remote-exec):   Private key: true
62 linode_instance.drosos-tfvm01 (remote-exec):   Certificate: false
63 linode_instance.drosos-tfvm01 (remote-exec):   SSH Agent: false
64 linode_instance.drosos-tfvm01 (remote-exec):   Checking Host Key: false
65 linode_instance.drosos-tfvm01 (remote-exec):   Target Platform: unix
66 linode_instance.drosos-tfvm01 (remote-exec): Connected!
67 linode_instance.drosos-tfvm01 (remote-exec): # Executing docker install script, commit: 6d9743e9656cc56f69
68 linode_instance.drosos-tfvm01 (remote-exec): + sh -c apt-get update -qq >/dev/null
69 linode_instance.drosos-tfvm01: Still creating... [1m10s elapsed]
70 linode_instance.drosos-tfvm01: Still creating... [1m20s elapsed]
71 linode_instance.drosos-tfvm01 (remote-exec): + sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq
72 linode_instance.drosos-tfvm01 (remote-exec): + sh -c install -m 0755 -d /etc/apt/keyrings
73 linode_instance.drosos-tfvm01 (remote-exec): + sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/c
74 linode_instance.drosos-tfvm01 (remote-exec): + sh -c chmod a+r /etc/apt/keyrings/docker.asc
75 linode_instance.drosos-tfvm01 (remote-exec): + sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] c/apt/sources.list.d/docker.list
76 linode_instance.drosos-tfvm01 (remote-exec): + sh -c apt-get update -qq >/dev/null
77 linode_instance.drosos-tfvm01 (remote-exec): + sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq
    -rootless-extras docker-buildx-plugin >/dev/null
78 linode_instance.drosos-tfvm01: Still creating... [1m30s elapsed]
79 linode_instance.drosos-tfvm01: Still creating... [1m40s elapsed]
80 linode_instance.drosos-tfvm01 (remote-exec): + sh -c docker version
81 linode_instance.drosos-tfvm01 (remote-exec): Client: Docker Engine - Community
82 linode_instance.drosos-tfvm01 (remote-exec): Version:           27.0.3
83 linode_instance.drosos-tfvm01 (remote-exec): API version:        1.46
84 linode_instance.drosos-tfvm01 (remote-exec): Go version:         go1.21.11
85 linode_instance.drosos-tfvm01 (remote-exec): Git commit:        7d4bcd8
86 linode_instance.drosos-tfvm01 (remote-exec): Built:              Sat Jun 29 00:02:33 2024
87 linode_instance.drosos-tfvm01 (remote-exec): OS/Arch:            linux/amd64

```

Εικόνα 53. Deploy stage: Linode VM - report [2]

```

139 linode_instance.drosos-tfvm01: Creation complete after 1m45s [id=61671173]
140 Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
141 Outputs:
142 instance_ip = "45.33.9.167"
143 Running after_script
144 Running after_script+
145 $ echo "Linode deployment complete!!!"
146 Linode deployment complete!!!
147 $ INSTANCE_IP=$(terraform output -json instance_ip | jq -r '.')
148 $ echo "Connecting to $INSTANCE_IP ...."
149 Connecting to 45.33.9.167 ....
150 $ echo "$INSTANCE_IP" > instance_ip.txt
151 Saving cache for successful job
152 Creating cache /builds/drososkats/gitlab-ci-cd-course-protected...
153 /builds/drososkats/gitlab-ci-cd-course/.terraform/: found 11 matching artifact files and directories
154 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/59600089/builds/drososkats/
155 Created cache
156 Uploading artifacts for successful job
157 Uploading artifacts...
158 instance_ip.txt: found 1 matching artifact files and directories
159 WARNING: Upload request redirected                                     location=https://gitlab.com/api/v4/jobs/7388467035/artifacts?all=1&
160 WARNING: Retrying...                                                 context=artifacts-uploader error=request redirected
161 Uploading artifacts as "archive" to coordinator... 201 Created id=7388467035 responseStatus=201 Created token=global
162 Cleaning up project directory and file based variables
163 Job succeeded

```

Εικόνα 54. Deploy stage: Linode VM - report [3]

Όπως και στο Build stage, έτσι και εδώ χωρίζεται σε 2 jobs, ένα για να γίνει το deploy του infrastructure, να γίνει εγκατάσταση αυτόματα το docker και να αποθηκευτεί ως artifacts η IP διεύθυνση που θα μας χρησιμεύσει στο επόμενο job, όπου θα γίνει αυτοματοποιημένα το deploy της εφαρμογής. Πιο συγκεκριμένα παρακάτω έχουμε το job:deploy-image :

```

deploy-image
Passed Started 11 hours ago by Drosos Katsimpras

1 Running with gitlab-runner 17.0.0~pre.88.g761ae5dd (761ae5dd)
2   on green-3.saas-linux-small-amd64.runners-manager.gitlab.com/default Jhc_Jxvh, system ID: s_0e6850b2bce1
3 Preparing the "docker+machine" executor
4 Using Docker executor with image mcr.microsoft.com/powershell ...
5 Pulling docker image sha256:0feac0f6bbe6480c2f7fa5c93d8b9e9d1d3219ed83d98cd66bf7242735c90b0 for mcr.microsoft.com/powershell with di
766f1739d329b2948df1fd7d5b0 ...
6 Using docker image sha256:0feac0f6bbe6480c2f7fa5c93d8b9e9d1d3219ed83d98cd66bf7242735c90b0 for mcr.microsoft.com/powershell with di
766f1739d329b2948df1fd7d5b0 ...
7 Preparing environment
8 Getting source from Git repository
9 Restoring cache
10 Downloading artifacts
11 Downloading artifacts for deploy-linode (7388467835)...
12 Downloading artifacts from coordinator... ok host=storage.googleapis.com id=7388467835 responseStatus=200 OK token=glcbt-66
13 Executing "step_script" stage of the job script
14 Using docker image sha256:0feac0f6bbe6480c2f7fa5c93d8b9e9d1d3219ed83d98cd66bf7242735c90b0 for mcr.microsoft.com/powershell with di
766f1739d329b2948df1fd7d5b0 ...
15 $ chmod 400 $SSH_KEY
16 $ LINODE_IP=$(pwsh -Command "Get-Content -Path instance_ip.txt") # collapsed multi-line command
17 Connecting to 45.33.9.167 ...
18 Warning: Permanently added '45.33.9.167' (ED25519) to the list of known hosts.
19 WARNING! Using --password via the CLI is insecure. Use --password-stdin.
20 WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
21 Configure a credential helper to remove this warning. See
22 https://docs.docker.com/engine/reference/commandline/login/#credential-stores
23 Login Succeeded
24 Unable to find image 'katsdros/test:python-app-1.0' locally
25 python-app-1.0: Pulling from katsdros/test
26 8b91b88d5577: Pulling fs layer
27 824416e23423: Pulling fs layer

```

Εικόνα 55. Deploy stage: Docker image - report [1]

```

72 1349d03ccfa2: Download complete
73 8b91b88d5577: Pull complete
74 824416e23423: Pull complete
75 8d53da260408: Pull complete
76 84c8c79126f6: Pull complete
77 2e1c130fa3ec: Pull complete
78 dbfa432d1d37: Pull complete
79 4c73541e7368: Pull complete
80 1349d03ccfa2: Pull complete
81 1ba558048ea8: Pull complete
82 d66b0c8af00d: Pull complete
83 Digest: sha256:0b84ba0685adf7a7ddd4a8d336b6b66d68477ca535a63fe67a1b9c90f2d3fb04
84 Status: Downloaded newer image for katsdros/test:python-app-1.0
85 6178218a4c215f1c02745daf8e502fe5ca8f18d4df6be991c46f059df3cc2cd1
86 Saving cache for successful job
87 Creating cache /builds/drososkats/gitlab-ci-cd-course-protected...
88 /builds/drososkats/gitlab-ci-cd-course/.terraform/: found 11 matching artifact files and directories
89 Archive is up to date!
90 Created cache
91 Cleaning up project directory and file based variables
92 Job succeeded

```

Εικόνα 56. Deploy stage: Docker image - report [2]

Πρακτικά, αφού παρουσιάσαμε το deploy-linode που στην ουσία ήταν ώστε να δημιουργηθεί αυτόματα το Infrastructure πάνω στον cloud provider που έχουμε επιλέξει, μετά εκτελέστηκε το τελευταίο job, που είναι για το deploy της εφαρμογής. Όπως είδαμε στα 2 τελευταία screenshots, το Docker image της εφαρμογής έγινε deploy μέσα στο VM επιτυχώς, πράγμα που σημαίνει πως η εφαρμογή εκτελείται κανονικά στην εικονική μηχανή. Στο Κεφάλαιο 6, θα δούμε αναλυτικότερα τα αποτελέσματα από το πρακτικό σενάριο που εκτελέσαμε.

Οπότε έχουμε φτάσει στο σημείο που το CI/CD pipeline έχει εκτελέσει επιτυχώς όλα τα stages και τα jobs που έχουν δημιουργηθεί μέσα στο αρχείο .gitlab-ci.yml. Συμπερασματικά λοιπόν, μπορούμε να πούμε πως το CI/CD pipeline που παρουσιάστηκε εδώ, αυτοματοποιεί τη διαδικασία ανάπτυξης, από τον κώδικα μέχρι την ανάπτυξη της εφαρμογής στο Linode VM. Αυτός ο αυτοματισμός και η λογική του **"CODE EVERYTHING"**, όχι μόνο επιταχύνει τη διαδικασία ανάπτυξης, αλλά και εξασφαλίζει την συνέπεια, την αξιοπιστία και τον έγκαιρο εντοπισμό προβλημάτων. Συνεπώς, μελλοντικές επεκτάσεις που θα αναλυθούν και στο Κεφάλαιο 7, θα μπορούσαν να περιλαμβάνουν την ενσωμάτωση πιο προηγμένων εργαλείων παρακολούθησης όπως Prometheus και Grafana και ειδοποιήσεις για τη βελτίωση της δυνατότητας συντήρησης του συνολικού συστήματος.

## 5.5 Χρονοδιάγραμμα

Σε αυτή την υποενότητα είναι σημαντικό να δοθεί ως μια συνολική εικόνα της διπλωματικής εργασίας ένα χρονοδιάγραμμα που εργαστήκαμε ώστε να ενσωματώσουμε το θεωρητικό υπόβαθρο και το πρακτικό μέρος στο report της εργασίας. Η διπλωματική εργασία εκτείνεται από τον Φεβρουάριο έως τον Ιούλιο, περιγράφοντας τη δομημένη και σταδιακή προσέγγιση που ακολουθήσαμε για την επίτευξη των στόχων του έργου. Στο παρακάτω πίνακα φαίνεται συνοπτικά ένα χρονοδιάγραμμα της διπλωματικής εργασίας:

Πίνακας 4. Χρονοδιάγραμμα διπλωματικής εργασίας

| Μήνας       | Περιγραφή των εργασιών  |
|-------------|---|
| Φεβρουάριος | Έρευνα και βιβλιογραφία για μεθοδολογίες SDLC και εργαλεία IaC. |
| Μάρτιος     | Πειραματισμός με Cloud Providers [AWS, Azure, DigitalOcean]     |
| Απρίλιος    | Επιλογή και πειραματισμός με Terraform για παροχή υποδομής.     |
| Μάιος       | Ανάπτυξη και δοκιμή του αγωγού CI/CD με χρήση GitLab CI/CD.     |
| Ιούνιος     | Ενσωμάτωση και δοκιμή του pipeline [code commit - deployment]   |
| Ιούλιος     | Τελικός έλεγχος, επαλήθευση και τεκμηρίωση των αποτελεσμάτων.   |

## 5.6 Προϋπολογισμός

Αυτή η ενότητα πρακτικά θα περιγράψει συνοπτικά τον προϋπολογισμό για την πρακτική εφαρμογή του αγωγού CI/CD, την παροχή της υποδομής, καθώς και του overleaf για την δημιουργία του report της διπλωματικής. Ο προϋπολογισμός λοιπόν περιλαμβάνει δαπάνες που σχετίζονται με την υποδομή cloud, τα εργαλεία λογισμικού και πρόσθετους πόρους που χρησιμοποιήθηκαν για να δοκιμή ώστε να ολοκληρωθεί το έργο. Ο στόχος είναι να παρέχει μια σαφή εικόνα και επισκόπηση των οικονομικών στοιχείων που δαπανήθηκαν για την επίτευξη των στόχων του έργου εντός του καθορισμένου χρονοδιαγράμματος που αναφέραμε στην προηγούμενη ενότητα. Συνεπώς, ο παρακάτω πίνακας περιγράφει λεπτομερώς το εκτιμώμενο κόστος που σχετίζεται με τη χρήση διάφορων παρόχων cloud και των υπηρεσιών του, καθώς και για το latex που έχει δημιουργηθεί το report της εργασίας, όπου χρησιμοποιήθηκε η premium έκδοση του Overleaf:

Πίνακας 5. Προϋπολογισμός διπλωματικής εργασίας

| Είδος         | Περιγραφή                                 | Κόστος        |
|---------------|---|---------------|
| Linode        | Linode 2GB, Linode 4GB                    | 30\$          |
| AWS           | Εγγραφή, t2.micro instance, VPC, Database | 4\$           |
| Azure         | Εγγραφή                                   | 1\$           |
| DigitalOcean  | Εγγραφή                                   | 5\$           |
| GitLab        | GitLab 1000xCompute                       | 10\$          |
| Overleaf      | Premium Εγγραφή                           | 9.8\$         |
| <b>Σύνολο</b> |   | <b>59.8\$</b> |



# Κεφάλαιο 6

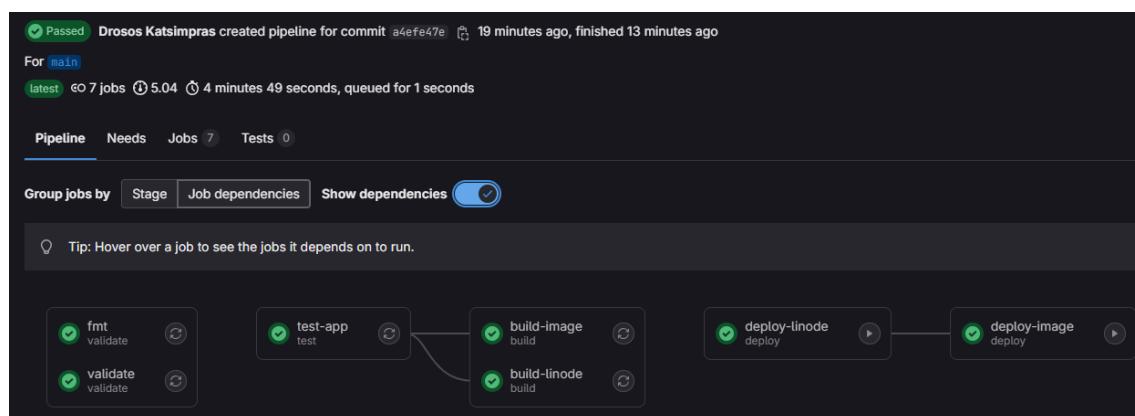
## Αποτελέσματα – Επιτεύγματα

Στο Κεφάλαιο 6, είμαστε σε θέση να παρουσιάσουμε τα αποτελέσματα και τα επιτεύγματα της υλοποίησης του CI/CD pipeline και της παροχής του Infrastructure μέσα από την αυτοματοποιημένη ανάπτυξη ολοκληρου του συστήματος. Αρχικά η εστίαση θα είναι στην αξιολόγηση της αποτελεσματικότητας του αγωγού, της απόδοσης και της προσβασιμότητας της εφαρμογής και του συνολικότερου αντίκτυπου στον κύκλο ζωής ανάπτυξης των συστημάτων λογισμικού.

Συνεπώς, καλούμαστε να δείξουμε τον τρόπο με τον οποίο αφού εκτελέστηκε επιτυχώς ο αγωγός, πως μοιάζει η εφαρμογή στο Linode VM, που είναι προσβάσιμο μέσα από τον browser μας, καθώς και την επαλήθευση που μπορούμε να κάνουμε μέσα από το terminal. Έτσι, αυτό το κεφάλαιο στοχεύει στο να επισημάνει τα οφέλη που επιτεύχθηκαν μέσω του έργου κατά την ανάπτυξη όλου του CI/CD pipeline με τις καινοτόμες τεχνολογίες που έχουμε αναλύσει διεξοδικά κατά την διάρκεια της διπλωματικής εργασίας.

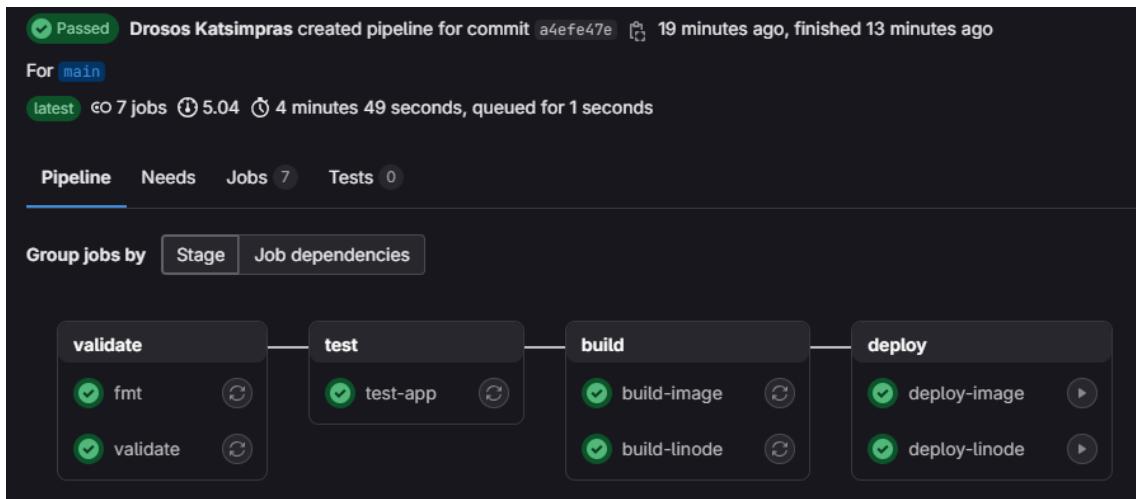
### 6.1 Αποτελέσματα κατά την εκτέλεση του CI/CD pipeline

Όπως είδαμε στο Κεφάλαιο 5 αναλυτικά, κάνοντας "kick off" το CI/CD pipeline, είχαμε ένα συνολικό αποτέλεσμα επιτυχίας σε όλα τα στάδια. Η επιτυχής επίτευξη του αγωγού είναι ένα από τα πιο βασικά επιτεύγματα της διπλωματικής εργασίας, καθώς αποδεικνύει με τον πιο σωστό τρόπο την αξιοπιστία και την αποτελεσματικότητα της αυτοματοποιημένης ροής εργασιών. Τα στάδια του CI/CD αγωγού εκτελέστηκαν απρόσκοπτα διασφαλίζοντας ότι η εφαρμογή θα αναπτυχθεί σωστά στο production environment. Παρακάτω βλέπουμε πως όλος ο αγωγός έχει ολοκληρωθεί με επιτυχία:



Εικόνα 57. CI/CD pipeline - Group by Dependencies

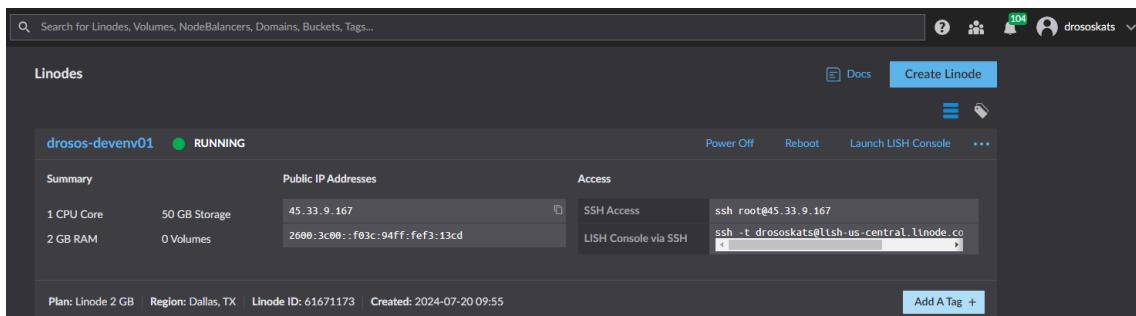
Επίσης μπορούμε μέσα από την πλατφόρμα να έχουμε οπτικοποίηση όλου του pipeline και με ομαδοποίηση των jobs από τα ίδια τα stages του αγωγού :



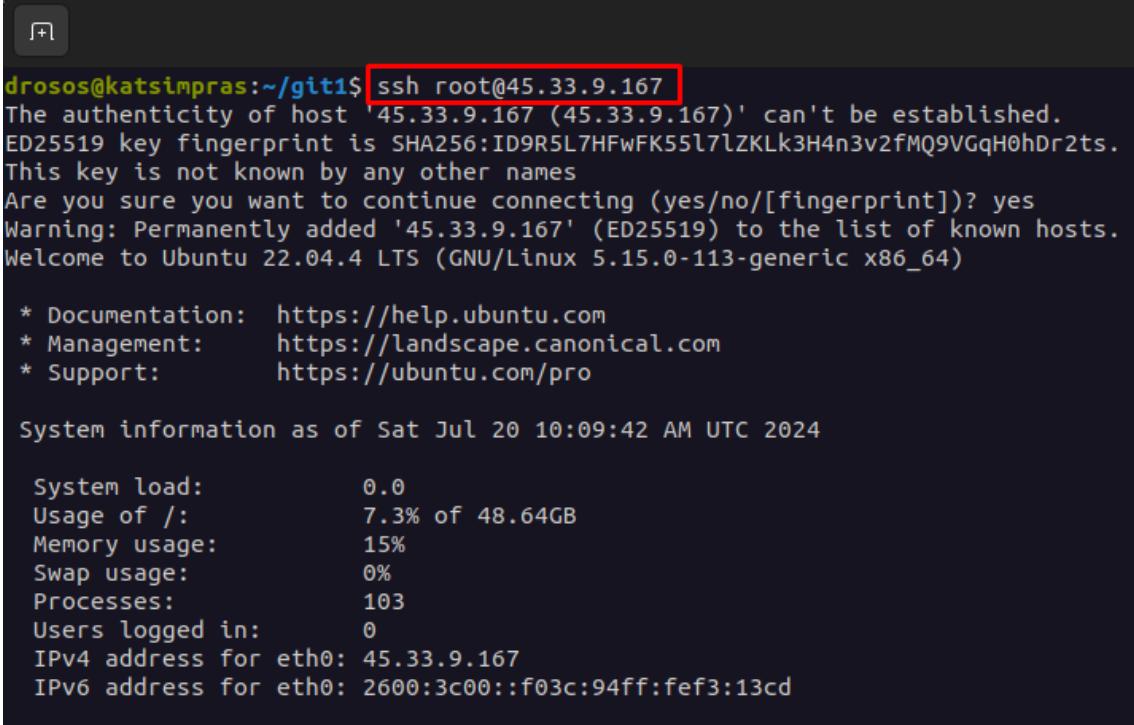
Εικόνα 58. CI/CD pipeline - Group by Stage

## 6.2 Ανάπτυξη και Προσβασιμότητα Υποδομής

Σε αυτό το σημείο, και αφού έχουμε αναλύσει πλήρως το CI/CD pipeline, το πως λειτουργεί, καθώς και ποια είναι τα χαρακτηριστικά του, είναι σημαντικό να δούμε και την προσβασιμότητα της υποδομής που έχουμε δημιουργήσει για την ανάπτυξη της εφαρμογής. Παρακάτω, παρατηρούμε πως με την εντολή `ssh root@LINODE_IP` είμαστε σε θέση μέσω του terminal να κάνουμε login στο Linode VM. Παράλληλα, έχοντας πρόσβαση στον λογαριασμό μας στον Linode πάροχο παρατηρούμε πως αυτόματα έχει δημιουργηθεί η υποδομή μας που είναι ένα VM με λογισμικό Ubuntu 22.04LTS και 1 CPUs, 2GB RAM και 50GB storage :



Εικόνα 59. Linode VM



```
drosos@katsimpras:~/git1$ ssh root@45.33.9.167
The authenticity of host '45.33.9.167 (45.33.9.167)' can't be established.
ED25519 key fingerprint is SHA256:ID9R5L7HFwFK55l7lZKLk3H4n3v2fMQ9VGqH0hDr2ts.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '45.33.9.167' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-113-generic x86_64)

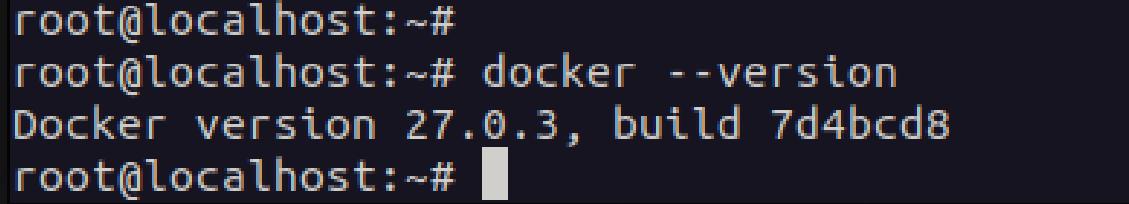
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Sat Jul 20 10:09:42 AM UTC 2024

 System load:          0.0
 Usage of /:           7.3% of 48.64GB
 Memory usage:        15%
 Swap usage:          0%
 Processes:            103
 Users logged in:     0
 IPv4 address for eth0: 45.33.9.167
 IPv6 address for eth0: 2600:3c00::f03c:94ff:fef3:13cd
```

Εικόνα 60. Linode VM

Παρατηρούμε πως είμαστε σε θέση να πραγματοποιήσουμε το connection με το Linode VM, και αν κάνουμε μια επαλήθευση με την εντολή `docker --version`, παρατηρούμε στο επόμενο screenshot πως έχει γίνει install το docker αυτόματα μέσα από τα Terraform Configuration files και συγκεκριμένα κάνοντας χρήση ενός provisioner "remote-exec" :

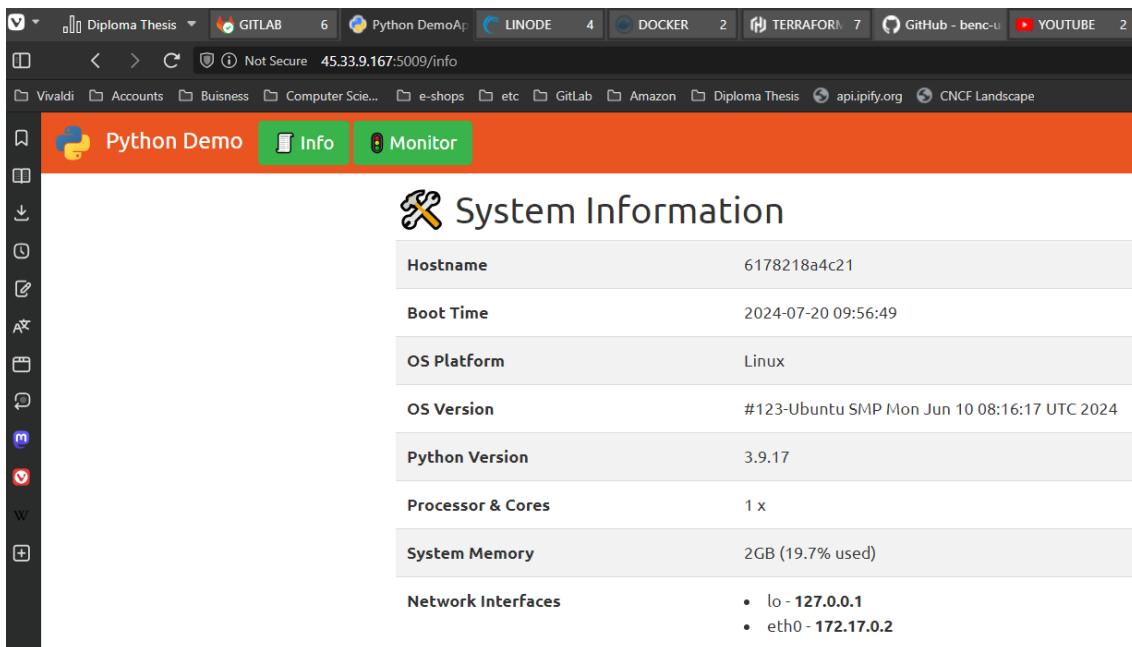


```
root@localhost:~#
root@localhost:~# docker --version
Docker version 27.0.3, build 7d4bcd8
root@localhost:~#
```

Εικόνα 61. Docker version

## 6.3 Ανάπτυξη και Προσβασιμότητα εφαρμογής

Συνεχίζοντας σε αυτή την ενότητα, και αφού είδαμε πως μετά την επιτυχή εκτέλεση του αγωγού CI/CD, η εφαρμογή αναπτύχθηκε σε ένα Linode VM. Η διαδικασία ανάπτυξης του Infrastructure και τη εφαρμογής πραγματοποιήθηκε και αυτοματοποιήθηκε με χρήση του Terraform, και του Docker διασφαλίζοντας την συνέπεια και την επαναληψιμότητα. Έτσι η αναπτυγμένη εφαρμογή μπορεί κάλλιστα να είναι προσβάσιμη μέσα από έναν browser με την IP διεύθυνση που έχουμε από το virtual machine, παρέχοντας έτσι την επικύρωση ότι λειτουργεί σωστά. Αυτό είναι πολύ σημαντικό γεγονός, καθώς επιβεβαιώνει τη λειτουργικότητα από άκρο σε άκρο όλης της αυτοματοποιημένης διαδικασίας ανάπτυξης. Παρακάτω βλέπουμε το Linode VM που έχει δημιουργηθεί αυτόματα καθώς και την εφαρμογή που είναι προσβάσιμη στο διαδίκτυο:



Εικόνα 62. Deployed Application

Και ενώ βλέπουμε ότι έχουμε κανονικά πρόσβαση με την IP του VM, μπορούμε φυσικά να κάνουμε και επαλήθευση του συστήματος ότι "τρέχει" κανονικά και μέσα από το VM με την εντολή `docker ps` που είναι μια εντολή που μας δείχνει ουσιαστικά ποιά docker containers είναι ενεργά και τρέχουν μέσα στο VM:

```
root@localhost:~# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
6178218a4c21 katsdros/test:python-app-1.0 "gunicorn -b 0.0.0.0..." 12 minutes ago Up 12 minutes 0.0.0.0:5009->5009/tcp, :::5009->5009/tcp pensive_mendeleev
root@localhost:~#
```

Εικόνα 63. Docker ps

Επίσης μπορούμε με την εντολή `systemctl status docker` να δούμε ότι το docker container service της εφαρμογής είναι ενεργό και σε **running** κατάσταση, όπως βλέπουμε στην παρακάτω εικόνα:

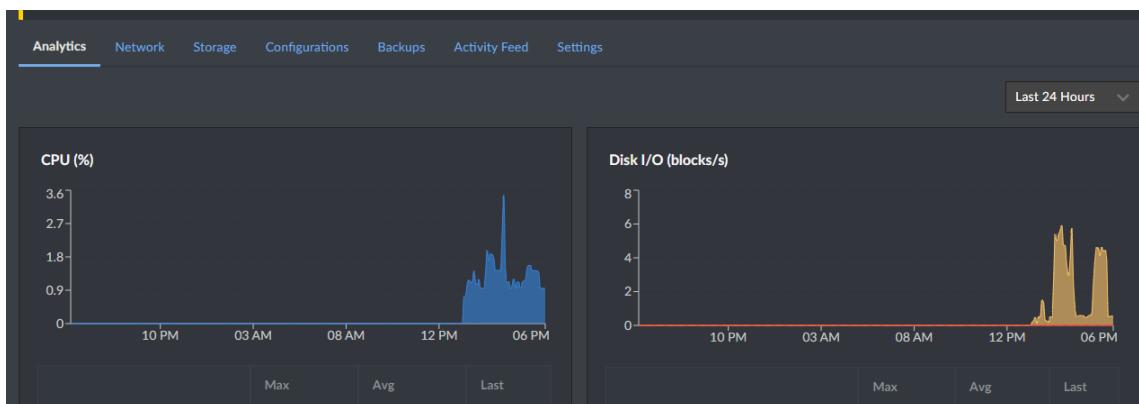
```
root@localhost:~# sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2024-07-20 09:57:40 UTC; 22min ago
TriggeredBy: ● docker.socket
   Docs: https://docs.docker.com
 Main PID: 20777 (docker)
    Tasks: 26
   Memory: 209.8M
      CPU: 3.4695
CGroup: /system.slice/docker.service
        ├─20777 /usr/bin/dockerd -H fd:// - containerd=/run/containerd/containerd.sock
        ├─2538 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 5009 -container-ip 172.17.0.2 -container-port 5009
        ├─2535 /usr/bin/docker-proxy -proto tcp -host-ip : -host-port 5009 -container-ip 172.17.0.2 -container-port 5009

Jul 20 09:57:40 localhost dockerd[2077]: time="2024-07-20T09:57:40.478775882Z" level=info msg="Starting up"
Jul 20 09:57:40 localhost dockerd[2077]: time="2024-07-20T09:57:40.479954916Z" level=info msg="detected 127.0.0.53 nameserver, assuming systemd-resolved, so using resolv.conf: /run/systemd/resolve/resolv.conf"
Jul 20 09:57:40 localhost dockerd[2077]: time="2024-07-20T09:57:40.5056456236Z" level=info msg="Loading containers: start."
Jul 20 09:57:40 localhost dockerd[2077]: time="2024-07-20T09:57:40.835431717Z" level=info msg="Loading containers: done."
Jul 20 09:57:40 localhost dockerd[2077]: time="2024-07-20T09:57:40.8487545852Z" level=info msg="Docker daemon" commit=d60f78c containerd-snapshotter=false storage-driver=overlay2 version=d60f78c
Jul 20 09:57:40 localhost dockerd[2077]: time="2024-07-20T09:57:40.873137415Z" level=info msg="Daemon has completed initialization"
Jul 20 09:57:40 localhost systemd[1]: Started Docker Application Container Engine
Jul 20 10:15:29 localhost dockerd[2077]: time="2024-07-20T10:15:29.055985057Z" level=error msg="Handler for GET /v1.40/services returned error: This node is not a swarm manager. Use --tlsverify=0 to disable TLS verification." file="main.go:100"
Jul 20 10:16:01 localhost dockerd[2077]: time="2024-07-20T10:16:01.738869932Z" level=error msg="Handler for GET /v1.40/services/6178218a4c21 returned error: This node is not a swarm manager. Use --tlsverify=0 to disable TLS verification." file="main.go:100"
```

Εικόνα 64. Docker ps

## 6.4 Infrastructure Monitoring

Σε αυτό το σημείο είναι σημαντικό να αναφέρουμε πως μέσα από την πλατφόρμα του cloud provider έχουμε την δυνατότητα να κάνουμε monitor τα resources με γραφήματα και να ελέγχουμε την υποδομή μας. Σε επέκταση του Monitor γενικά όλης της υποδομής θα μπορούσαμε να χρησιμοποιήσουμε το Prometheus και Grafana για να βλέπουμε αναλυτικότερα διάφορα metrics της υποδομής και να έχουμε στοχευμένο ένα dashboard για όλη την υποδομή. Παρακάτω, βλέπουμε ένα απλό γράφημα που μπορούμε να αντλήσουμε από το Linode:

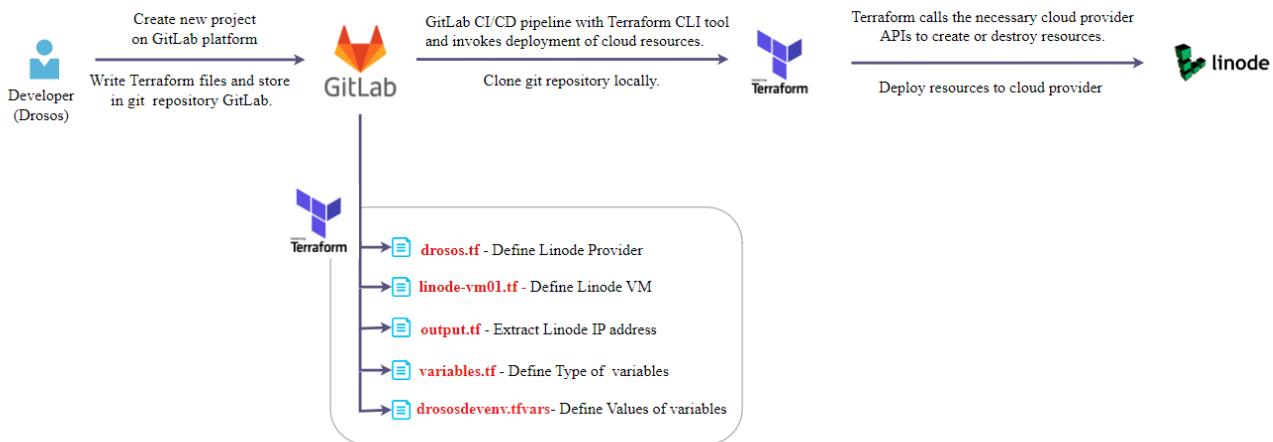


Εικόνα 65. Infrastructure Monitoring

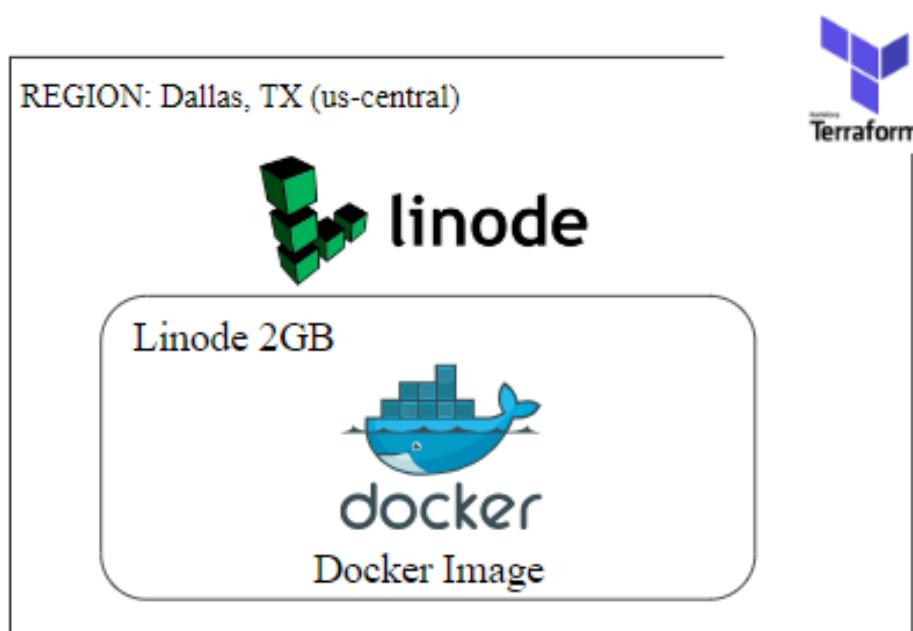
Κλείνοντας αυτό το Κεφάλαιο είναι σημαντικό να αναφέρουμε πως η ανάπτυξη της υποδομής και της εφαρμογής μέσα στο VM ήταν βασικά στοιχεία για την επιτυχή υλοποίηση του αγωγού CI/CD. Με χρήση του Terraform ως IaC εργαλείου για να επιτύχουμε automated provisioning και με χρήση του Docker container για την εφαρμογή, μας εξασφαλίζει μια ισχυρή και επεκτάσιμη εγκατάσταση υποδομής που ανάλογα το σενάριο μπορούμε εύκολα και γρήγορα να επεκτείνουμε τις απαιτήσεις. Παράλληλα με το πρωτόκολλο SSH είμαστε σε θέση να έχουμε μια αποτελεσματική απομακρυσμένη διαχείριση της υποδομής, συμβάλλοντας στην αξιοπιστία της εφαρμογής. Αυτές οι βέλτιστες και καινοτόμες προσεγγίσεις υποστήριξαν συνολικά τον στόχο του έργου για την αυτοματοποίηση της διαδικασίας ανάπτυξης.

## 6.5 Terraform Σχεδιάγραμμα

Παρακάτω μπορούμε σε σχηματική απεικόνηση τις ενέργειες που έχουν γίνει για την ανάπτυξη των Terraform Configuration αρχείων, ώστε να έχουμε μια σωστότερη απεικόνηση των όσων έχουμε αναλύσει σε προηγούμενο Κεφάλαιο:



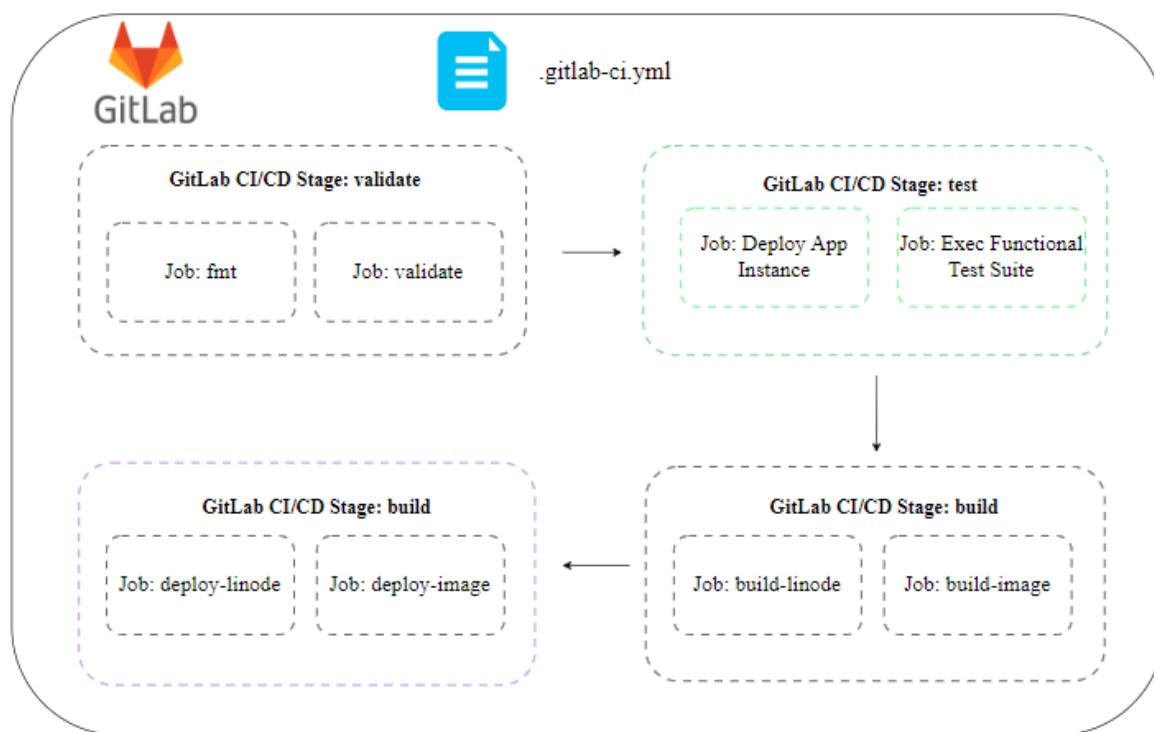
Εικόνα 66. Terraform diagram



Εικόνα 67. Terraform : Linode VM

## 6.6 GitLab CI/CD pipeline

Σε αυτή την ενότητα είναι επίσης σημαντικό να μπορέσουμε να αποδώσουμε σχεδιαγραμματικά το GitLab CI/CD pipeline και να μπορέσουμε να έχουμε την καλύτερη δυνατή αποτύπωση:



Εικόνα 68. GitLab CI/CD pipeline



## Κεφάλαιο 7

# Συμπεράσματα – Μελλοντικές επεκτάσεις

Σε αυτό το κεφάλαιο, θα παρουσιάσουμε μια περίληωη των βασικών αποτελεσμάτων που επιτεύχθηκαν μέσω της υλοποίησης του αγωγού CI/CD και της αυτοματοποιημένης παροχής υποδομής. Επίσης, θα περιγράψει πιθανές μελλοντικές επεκτάσεις που μπορούν να βελτιώσουν τις δυνατότητες ενός CI/CD pipeline, αλλά και ολόκληρου του συστήματος, διασφαλίζοντας την συνεχή βελτίωση και επεκτασιμότητα.

## 7.1 Αποτελέσματα

### 7.1.1 Αποδοτικότητα αγωγού CI/CD

Το CI/CD pipeline που υλοποιήσαμε χρησιμοποιώντας το GitLab CI/CD, και με χρήση του Terraform και του Docker, έδειξε πως υπάρχουν σημαντικές βελτιώσεις στην αποτελεσματικότητα ανάπτυξης και στην ποιότητα του κώδικα. Πιο συγκεκριμένα μπορούμε να πούμε τα εξής βασικά αποτελέσματα:

- Αυτοματοποιημένη ανάπτυξη** : Το CI/CD pipeline αυτοματοποίησε επιτυχώς όλη την διαδικασία ανάπτυξης, μειώνοντας τη χειροκίνητη παρέμβαση και τα σφάλματα σε ανθρώπινο επίπεδο.
- Ταχύτερος Release cycle** : Ο χρόνος κατά την διάρκεια όλων των στοιχείων μέχρι το deploy της εφαρμογής μειώθηκε αισθητά, επιτρέποντας με αυτόν τον τρόπο να υπάρξουν νέες εκδόσεις όλου του συστήματος.
- Συνεπή περιβάλλοντα** : Με χρήση του Docker container, εξασφαλίζουμε πως η εφαρμογή μπορεί να εκτελείται άρτια σε διαφορετικά περιβάλλοντα. ελάχιστοποιώντας ζητήματα που σχετίζονται με το περιβάλλον που έχει επιλεγεί.

### 7.1.2 Σταθερότητα Υποδομής

Συμπερασματικά, μπορούμε να πούμε πως με την χρήση του ενός Infrastructure as Code εργαλείου, το **Terraform** είχαμε την δυνατότητα να παρέχουμε μια σταθερή και επαναλαμβανόμενη ρύθμιση, ανάλογα το σενάριο και τις απαιτήσεις που χρειαζόμασταν για την υποδομή. Πιο συγκεκριμένα μπορούμε να πούμε:

- **Αξιόπιστη Υποδομή** : Με τα Terraform Configuration files, σε μια δηλωτική προσέγγιση με χρήση της **Hashicorp Configuration Language (HCL)** εξασφαλίζουμε μια αξιόπιστη παροχή και διαχείριση της υποδομής, μειώνοντας τον κίνδυνο μετατόπισης της διαμόρφωσης.
- **Επεκτασιμότητα** : Με αυτό το εργαλείο που επιλέξαμε να χρησιμοποιήσουμε, αλλά και γενικά με τα IaC εργαλεία, έχουμε την δυνατότητα η ρύθμιση όλης της υποδομής να είναι εύκολα επεκτάσιμη, επιτρέποντας την προσθήκη περισσότερων resources όπως απαιτείται κατά την υιοθέτηση ενός σεναρίου.
- **Κόστος-αποδοτικότητα** : Παρατηρήσαμε πως μπορούμε εύκολα να έχουμε ότι resources χρειαζόμαστε για την υποδομή με ένα συγκεκριμένο κόστος για κάθε προσθήκη πόρου στην υποδομή μας. Αυτό μας δίνει την δυνατότητα να επιλέγουμε το κόστος που θα έχουμε, ανάλογα με τον προϋπολογισμό μας και τις απαιτήσεις μας.

## 7.2 Μελλοντικές επεκτάσεις

### 7.2.1 Βελτιωμένη παρακολούθηση και ειδοποίηση

Όπως είδαμε και στο προηγούμενο Κεφάλαιο, το βασικό monitoring της υποδομής έγινε με χρήση ενσωματωμένων εργαλείων του Linode, διασφαλίζοντας την παρακολούθηση της χρήσης πόρων και των μετρήσεων απόδοσης σε γραφικό περιβάλλον. Παρόλ’ αυτά, η χρήση και η ενσωμάτωση προηγμένων εργαλείων παρακολούθησης όπως είναι το Prometheus και το Grafana μπορεί να μας παρέχει πιο λεπτομερείς πληροφορίες σχετικά με την απόδοση και την υγεία του συστήματος:

- **Prometheus<sup>1</sup>** : Με την χρήση αυτού του open-source εργαλείου, μπορούμε να συλλέξουμε και να αναζητήσουμε μετρήσεις (metrics).
- **Grafana<sup>2</sup>** : Με την ενσωμάτωση του Grafana, θα έχουμε την δυνατότητα για την δημιουργία visual dashboards και την ρύθμιση κατάλληλων ειδοποιήσεων σε περιβάλλοντα όπως email ή στο Slack σε channels που μπορεί να συνεργάζονται οι DevOps Engineers.

### 7.2.2 Βελτιώσεις ασφάλειας

Η εφαρμογή και συνολικότερα όλη η υποδομή μπορεί να γίνει πιο ασφαλής με χρήση πρόσθετων μέτρων ασφαλείας που μπορεί να προστατεύσει περαιτέρω όλο το έργο. Πιο συγκεκριμένα:

- **Αυτοματοποιημένη σάρωση ευπάθειας** : Με την χρήση εργαλείων όπως το Clair ή το Trivy<sup>3</sup> μπορούμε να κάνουμε σάρωση των Docker images που χρησιμοποιούμε στο συνολικότερο έργο, για να εντοπιστούν τυχόν ευπάθειες.
- **Ασφαλής Διαχείριση secrets** : Με την χρήση εργαλείων όπως το **Hashicorp Vault<sup>4</sup>** σε επίπεδο enterprise μπορούμε να έχουμε ασφάλεια στην διαχείριση των secrets (keys, passwords, etc).

<sup>1</sup><https://prometheus.io>

<sup>2</sup><https://grafana.com/grafana/>

<sup>3</sup><https://trivy.dev>

<sup>4</sup><https://www.hashicorp.com/products/vault>

- **Χρήση ενός Virtual Private Cloud :** Έχουμε την δυνατότητα μέσα από τους cloud providers να χρησιμοποιήσουμε και να διαμορφώσουμε ένα δικό μας VPC με όλα τα απαραίτητα components που χρειάζεται, ώστε να μπορούμε με ασφάλεια μέσα στο VPC να αποθηκεύσουμε μια βάση δεδομένων, να κάνουμε ελέγχους και ανάπτυξη της εφαρμογής και να εγκαταστήσουμε διάφορα εργαλεία που χρειαζόμαστε για την υποδομή μας.

## 7.3 Βελτίωση επεκτασιμότητας

Μια άλλη μελλοντική επέκταση, τόσο του έργου που υλοποιήσαμε με το CI/CD pipeline και το Infrastructure με την βοήθεια του Terraform, είναι επίσης σημαντικό να μπορέσουμε μελλοντικά να εξερευνήσουμε εργαλεία ενορχήστρωσεις (orchestration tools) όπως είναι το **Kubernetes**<sup>5</sup> και σε μεγάλης κλίμακας σενάρια μπορεί να βελτιώσει την ικανότητα του συστήματος να κλίμακωνται δυναμικά. Το Kubernetes είναι ικανό να διαχειρίζεται εφαρμογές σε containers σε κλίμακα, επιτρέποντας με αυτόν τον τρόπο την αποτελεσματική κλιμάκωση και διαχείριση των πόρων.

## 7.4 Συμπέρασμα

Συμπερασματικά λοιπόν και κλείνοντας το Κεφάλαιο 7 και συνολικά όλη την διπλωματική εργασία, η υλοποίηση του CI/CD pipeline και η αυτοματοποιημένη παροχή υποδομής, έχουν βελτιώσει σε σημαντικό επίπεδο την αποτελεσματικότητα και την αξιοπιστία της διαδικασίας ανάπτυξης συστημάτων λογισμικού. Τα αποτελέσματα του έργου καταδεικνύουν την αξία των καινοτόμων προσεγγίσεων του SDLC και των σύγχρονων πρακτικών DevOps για την επίτευξη ταχύτερων, πιο συνεπών αναπτύξεων και υψηλότερης ποιότητας κώδικα.

Συνεπώς, στο μέλλον θα υπάρχουν πολλές ευκαιρίες για την βελτίωση των δυνατοτήτων του συστήματος μέσω βελτιμένης παρακολούθησης, ασφάλειας, επεκτασιμότητας και πρακτικών συνεχούς βελτίωσης. Οι μελλοντικές επεκτάσεις που αναφέρθηκαν, θα μπορέσουν να διασφαλίσουν ότι η υποδομή θα παραμείνει σταθερή στις απαιτήσεις και στις αλλαγές που θα χρειαστεί να γίνουν, ασφαλής και ικανή να ανταποκριθεί στις εξελισσόμενες απαιτήσεις του τεχνολογικού κόσμου.

Κλείνοντας, αυτό το Κεφάλαιο, παρέχει μια συνολική επισκόπηση των αποτελεσμάτων που επιτεύχθηκαν και των πιθανών ενεργειών που μπορούμε να πραγματοποιήσουμε για μελλοντική επέκταση και βελτίωση.

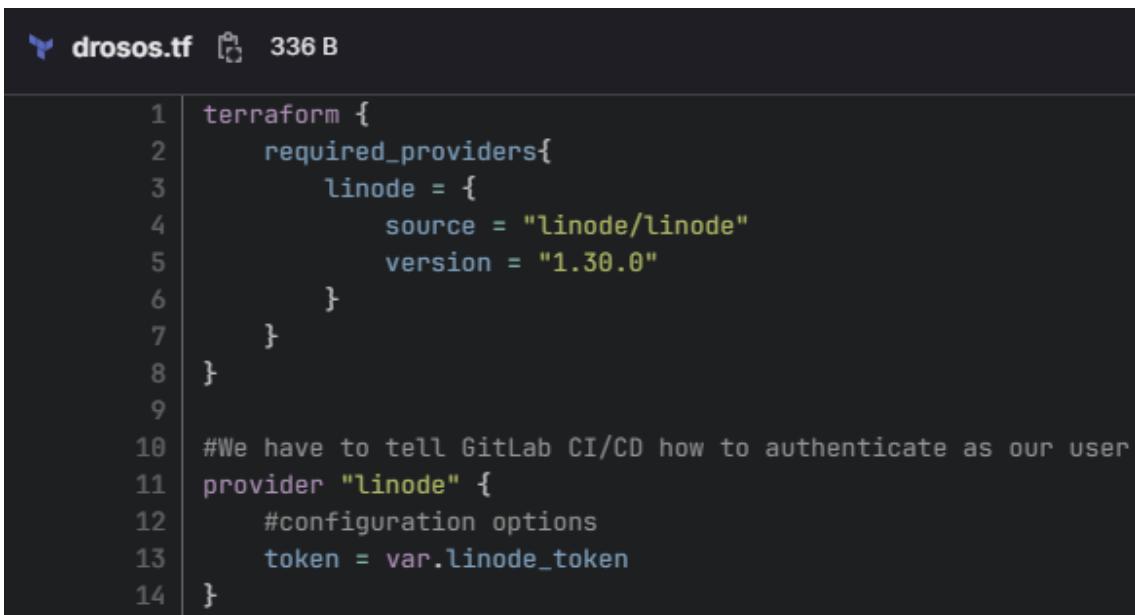
<sup>5</sup><https://kubernetes.io>



# Παράρτημα A

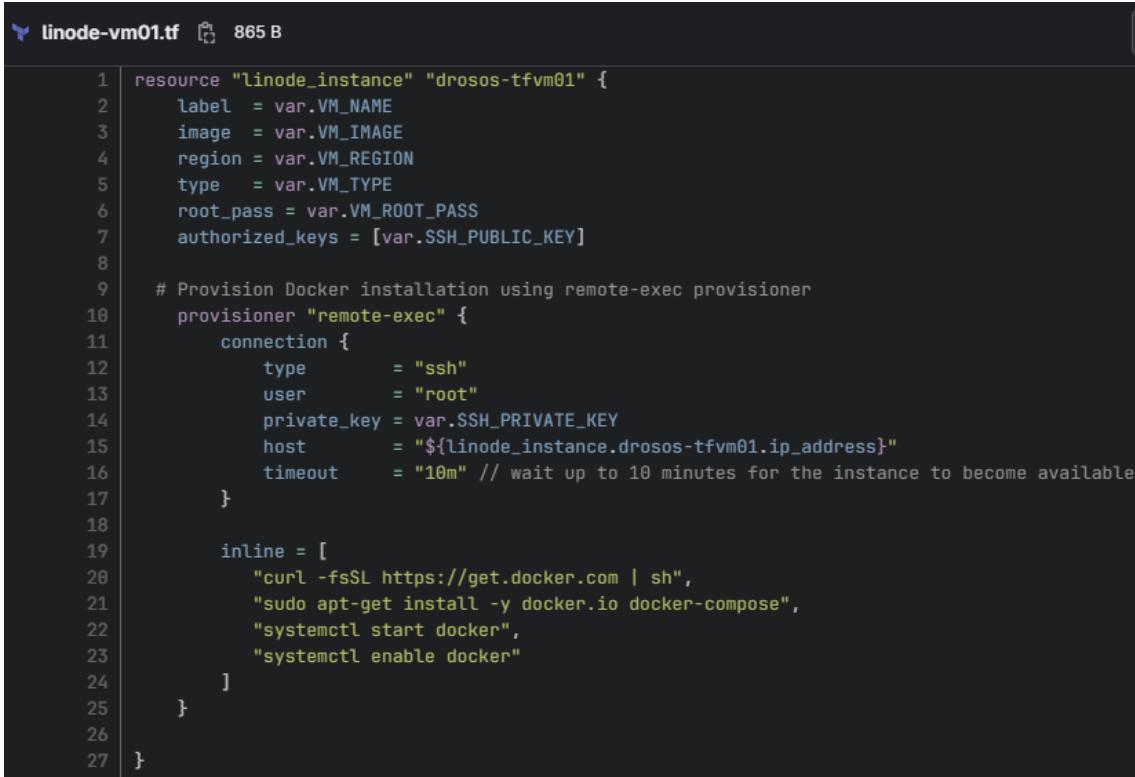
## Terraform

### A.1 Terraform Configuration files



```
 1 terraform {  
 2     required_providers{  
 3         linode = {  
 4             source = "linode/linode"  
 5             version = "1.30.0"  
 6         }  
 7     }  
 8 }  
 9  
10 #We have to tell GitLab CI/CD how to authenticate as our user  
11 provider "linode" {  
12     #configuration options  
13     token = var.linode_token  
14 }
```

Εικόνα 69. Cloud Provider : Linode

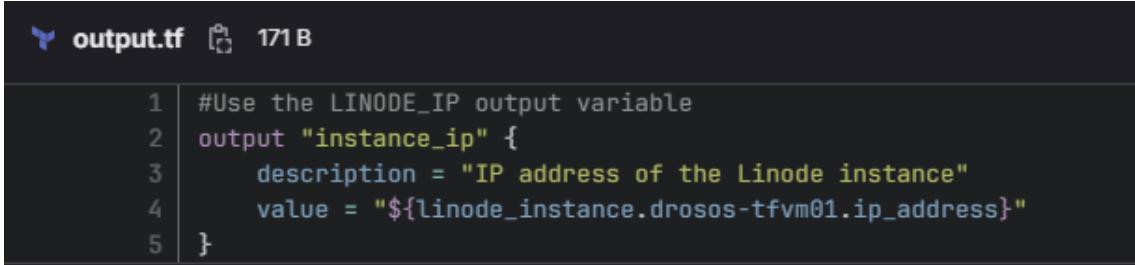


```

1 resource "linode_instance" "drosos-tfvm01" {
2   label  = var.VM_NAME
3   image  = var.VM_IMAGE
4   region = var.VM_REGION
5   type   = var.VM_TYPE
6   root_pass = var.VM_ROOT_PASS
7   authorized_keys = [var.SSH_PUBLIC_KEY]
8
9   # Provision Docker installation using remote-exec provisioner
10  provisioner "remote-exec" {
11    connection {
12      type     = "ssh"
13      user     = "root"
14      private_key = var.SSH_PRIVATE_KEY
15      host     = "${linode_instance.drosos-tfvm01.ip_address}"
16      timeout   = "10m" // wait up to 10 minutes for the instance to become available
17    }
18
19    inline = [
20      "curl -fsSL https://get.docker.com | sh",
21      "sudo apt-get install -y docker.io docker-compose",
22      "systemctl start docker",
23      "systemctl enable docker"
24    ]
25  }
26
27 }

```

Εικόνα 70. Linode VM



```

1 #Use the LINODE_IP output variable
2 output "instance_ip" {
3   description = "IP address of the Linode instance"
4   value       = "${linode_instance.drosos-tfvm01.ip_address}"
5 }

```

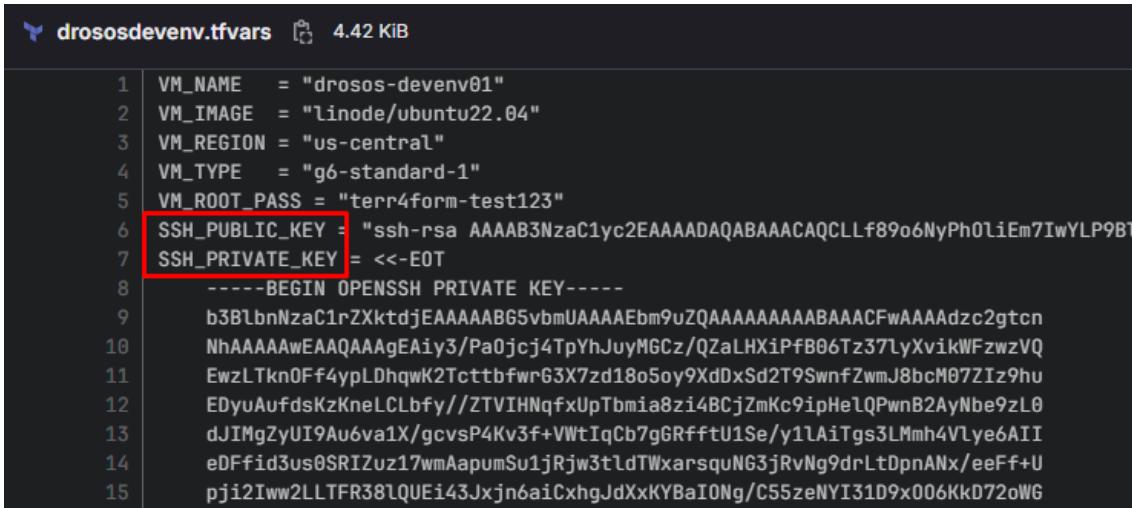
Εικόνα 71. Linode VM : instance ip



The screenshot shows a code editor window with a dark theme. At the top left is a blue icon resembling a tree or a 'T'. To its right is the file name "variables.tf". Next to it is a small icon of a document with a gear, followed by the text "445 B". The main area contains 33 numbered lines of Terraform configuration code. The code defines several variables:

```
1 variable "linode_token" {
2     type = string
3 }
4
5 variable "VM_NAME" {
6     type = string
7 }
8
9 variable "VM_TYPE" {
10    type = string
11 }
12
13 variable "VM_REGION" {
14    type = string
15 }
16
17 variable "VM_IMAGE" {
18    type = string
19 }
20
21 variable "VM_ROOT_PASS" {
22    type = string
23 }
24
25 variable "SSH_PUBLIC_KEY" {
26     description = "The SSH PUBLIC KEY content"
27     type = string
28 }
29
30 variable "SSH_PRIVATE_KEY" {
31     description = "The SSH PRIVATE KEY content"
32     type = string
33 }
```

Εικόνα 72. Αρχικοποίηση Terraform variables



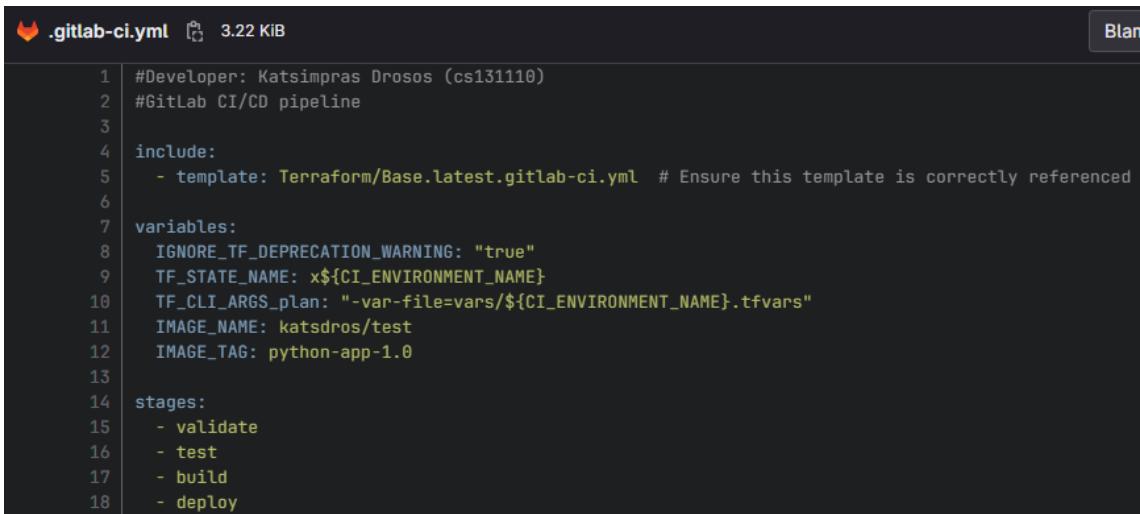
```
 1 VM_NAME      = "drosos-devenv01"
 2 VM_IMAGE     = "linode/ubuntu22.04"
 3 VM_REGION    = "us-central"
 4 VM_TYPE      = "g6-standard-1"
 5 VM_ROOT_PASS = "terr4form-test123"
 6 SSH_PUBLIC_KEY = "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCLLf89o6NyPh0LiEm7IwYLP9BL
 7 SSH_PRIVATE_KEY = <<-EOT
 8 -----BEGIN OPENSSH PRIVATE KEY-----
 9 b3BlnNzaC1rZXktdjEAAAABG5vbmAAAAEb9uZQAAAAAAAAABAAACFwAAAAdzc2gtcn
10 NhAAAAAwEAAQAAgEAiy3/Pa0jcj4TpYhJuyMGcz/QZaLHXiPfb06Tz37LyXvikWFzwzVQ
11 EwzLTkn0Ff4ypLDhqwK2Tcttbfr63X7zd18o5oy9XdDxSd2T9SwnfZwmJ8bcM077Iz9hu
12 EDyuAufdsKzKneLCLbfy//ZTVIHNqfxUpTbmia8zi4BCjZmKc9ipHelQPwnB2AyNbe9zL0
13 dJIMgZyUI9Au6va1X/gcvsP4Kv3f+VWtIqCb7gGRfftU1Se/y1lAiTgs3LMmh4Vlye6AII
14 eDFFid3us0SRIZuz17wmAapumSu1jRjw3tldTWxarsquNG3jRvNg9drLtDpnANx/eeFf+U
15 pji2Iww2LLTFR38lQUEi43Jxjn6aiCxhgJdXxKYBaIONg/C55zeNYI31D9x006Kkd72oWG
```

Εικόνα 73. Τιμές των Terraform variables

## Παράρτημα Β

# GitLab CI/CD pipeline

### B.1 Αρχείο .gitlab-ci.yml



```
#Developer: Katsimpras Drosos (cs131110)
#GitLab CI/CD pipeline

include:
  - template: Terraform/Base.latest.gitlab-ci.yml # Ensure this template is correctly referenced

variables:
  IGNORE_TF_DEPRECATED_WARNING: "true"
  TF_STATE_NAME: x${CI_ENVIRONMENT_NAME}
  TF_CLI_ARGS_plan: "-var-file=vars/${CI_ENVIRONMENT_NAME}.tfvars"
  IMAGE_NAME: katsdros/test
  IMAGE_TAG: python-app-1.0

stages:
  - validate
  - test
  - build
  - deploy
```

Εικόνα 74. .gitlab-ci.yml : Template, variables, stages

```

20   fmt:
21     extends: .terraform:fmt
22     needs: []
23     allow_failure: true
24
25 validate:
26   extends: .terraform:validate
27   needs: []
28   script:
29     - terraform --version # Print Terraform version for debugging
30
31 test-app:
32   stage: test
33   image: python:3.9-slim-buster
34   before_script:
35     - apt-get update && apt-get install -y make git
36     - pip install --upgrade pip
37     - pip install -r src/requirements.txt
38     - python --version
39     - pip list
40   script:
41     - make test
42

```

Εικόνα 75. .gitlab-ci.yml : Validate, test-app

```

53
54 build-linode:
55   extends: .terraform:build
56   needs:
57     - test-app
58   environment:
59     name: drososdevenv
60     action: prepare
61
62 build-image:
63   extends: .terraform:build
64   needs:
65     - test-app
66   environment:
67     name: drososdevenv
68     action: prepare
69   image: docker:20.10.16
70   services:
71     - docker:20.10.16-dind
72   before_script:
73     - docker login -u $REGISTRY_USER -p $REGISTRY_PASS
74   script:
75     - docker build -t $IMAGE_NAME:$IMAGE_TAG -f Dockerfile .
76     - docker push $IMAGE_NAME:$IMAGE_TAG
77

```

Εικόνα 76. .gitlab-ci.yml : build-linode, build-image

```
79 deploy-linode:
80   extends: .terraform:deploy
81   dependencies:
82     - build-linode
83   environment:
84     name: drososdevenv
85     action: start
86   after_script:
87     - echo "Linode deployment complete!!!"
88     - INSTANCE_IP=$(terraform output -json instance_ip | jq -r '.')
89     - echo "Connecting to $INSTANCE_IP ...."
90     - echo "$INSTANCE_IP" > instance_ip.txt # Save INSTANCE_IP to a text file
91   artifacts:
92     paths:
93       - instance_ip.txt
94
95 deploy-image:
96   extends: .terraform:deploy
97   image: mcr.microsoft.com/powershell
98   environment:
99     name: drososdevenv
100    action: prepare
101   before_script:
102     - chmod 400 $SSH_KEY
103   script: |
104     LINODE_IP=$(pwsh -Command "Get-Content -Path instance_ip.txt")
105     sleep 10
106     echo "Connecting to $LINODE_IP ...."
107     ssh -o StrictHostKeyChecking=no -i $SSH_KEY root@$LINODE_IP "
108       docker login -u $REGISTRY_USER -p $REGISTRY_PASS &&
109       docker run -d -p 5009:5009 $IMAGE_NAME:$IMAGE_TAG"
110   needs:
111     - deploy-linode
112
```

Εικόνα 77. .gitlab-ci.yml : deploy-linode, deploy-image



# Βιβλιογραφικές Αναφορές

- [1] S. Bairagi και A. Bang. *Cloud Computing: History, Architecture, Security Issues*. Μαρ. 2015.
- [2] Cloud Academy. *Cons of Cloud Computing*. 2023. URL: <https://cloudacademy.com/blog/disadvantages-of-cloud-computing/>.
- [3] «Domain 4—Cloud Application Security». Στο: *CCSP For Dummies with Online Practice*. For Dummies, 2023. Κεφ. 6.
- [4] A. Agarwal, A. Agarwal, D. K. Verma, D. Tiwari και R. Pandey. «A Review on Software Development Life Cycle». Στο: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 9.3 (2023), σσ. 384–388. DOI: [10.32628/CSEIT2390387](https://doi.org/10.32628/CSEIT2390387).
- [5] Argonaut. *The Top Infrastructure as Code (IaC) Tools for 2023*. Απρ. 2023. URL: <https://argonaut.dev/blog/infrastructure-as-code-tools>.
- [6] Spacelift. *Infrastructure as Code: Best Practices, Benefits & Examples*. 2024. URL: [https://spacelift.io/blog/\[slug\]](https://spacelift.io/blog/[slug]).
- [7] A. Abdullahi. *Top 8 Infrastructure as Code (IaC) Tools for 2024*. Φεβ. 2023. URL: <https://geekflare.com/infrastructure-as-code-iac-tools/>.
- [8] Splunk. *Infrastructure as Code (IaC)*. 2023. URL: <https://www.splunk.com/en-us/blog/learn/infrastructure-as-code-iac.html>.
- [9] Spiceworks Inc. *Infrastructure as Code Working and Benefits*. 2022. URL: <https://www.spiceworks.com/tech/cloud/articles/what-is-infrastructure-as-code/>.
- [10] LinkedIn. *(3) Common IAC Challenges and How to Overcome Them*. Μαρ. 2023. URL: <https://www.linkedin.com/pulse/common-iac-challenges-how-overcome-them-cloudmatoz/>.
- [11] AltexSoft. *Infrastructure as Code: Benefits, Types, and Tools*. URL: <https://www.altexsoft.com/blog/infrastructure-as-code/>.
- [12] GitGuardian. *Infrastructure as Code—Everything You Need to Know*. Ιαν. 2022. URL: <https://blog.gitguardian.com/infrastructure-as-code-everything-you-need-to-know/>.
- [13] TechTarget. *What Is Infrastructure as Code (IaC)? | Definition from TechTarget*. URL: <https://www.techtarget.com/searchitoperations/definition/Infrastructure-as-Code-IAC>.
- [14] M. Howard. «Terraform—Automating Infrastructure as a Service». Στο: *arXiv* (2022). DOI: [10.48550/arXiv.2205.10676](https://doi.org/10.48550/arXiv.2205.10676).
- [15] A. C. al. *How Ansible works*. Ansible Collaborative. Ιούν. 2024. URL: <https://www.ansible.com/how-ansible-works/>.

- [16] Technologies, E. *What is Ansible—Advantages and Disadvantages | Ansible Benefits*. Ezeelive.Com. Φεβ. 2024. URL: <https://ezeelive.com/ansible-advantages-disadvantages/>.
- [17] E. Özdogan, O. Ceran και M. T. Üstündağ. «Systematic Analysis of Infrastructure as Code Technologies». Στο: *Gazi University Journal of Science Part A: Engineering and Innovation* 10.4 (2023), σσ. 452–471. DOI: [10.54287/gujsa.1373305](https://doi.org/10.54287/gujsa.1373305).
- [18] Pulumi Inc. *Pulumi vs Terraform*. URL: <https://www.pulumi.com/docs/concepts/vs/terraform/>.
- [19] Encore.dev. *Pulumi: Imperative vs Declarative Models*. URL: <https://encore.dev/resources/pulumi#imperative-vs-declarative-models>.
- [20] Pulumi Inc. *How Pulumi Works*. URL: <https://www.pulumi.com/docs/concepts/how-pulumi-works/>.
- [21] Puppet Labs. *Puppet Overview*. URL: [https://www.puppet.com/docs/puppet/6/puppet\\_overview.html](https://www.puppet.com/docs/puppet/6/puppet_overview.html).
- [22] Microsoft. *Overview of Azure Resource Manager*. URL: <https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview>.
- [23] Encore Dev. *Google Cloud Deployment Manager*. URL: <https://encore.dev/resources/google-cloud-deployment-manager>.
- [24] N. Azad και S. Hyrynsalmi. «DevOps critical success factors—A systematic literature review». Στο: *Information and Software Technology* 157 (2023), σ. 107150. DOI: [10.1016/j.infsof.2023.107150](https://doi.org/10.1016/j.infsof.2023.107150).
- [25] GitLab. *DevOps Topics*. URL: <https://about.gitlab.com/topics/devops/>.
- [26] BrowserStack. *DevOps Lifecycle*. URL: <https://www.browserstack.com/guide/devops-lifecycle>.
- [27] Devopedia. *DevOps*. URL: <https://devopedia.org/devops>.
- [28] *What is a CI/CD pipeline?* URL: <https://about.gitlab.com/topics/ci-cd/cicd-pipeline/>.
- [29] Benc-Uk. *python-demoapp*. <https://github.com/benc-uk/python-demoapp>. Python application. 2021.
- [30] Spiceworks Inc. *Terraform vs. Ansible: Key Differences*. Σεπτ. 2022. URL: <https://www.spiceworks.com/tech/devops/articles/terraform-vs-ansible/>.
- [31] Chef Software, Inc. *Platform Overview*. URL: [https://docs.chef.io/platform\\_overview/](https://docs.chef.io/platform_overview/).
- [32] Google Cloud. *Deployment Manager documentation*. URL: <https://cloud.google.com/deployment-manager/docs>.
- [33] Mi. Guerriero, M. Garriga, D. A. Tamburri και F. Palomba. «Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry». Στο: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019, σσ. 580–589. DOI: [10.1109/ICSME.2019.00092](https://doi.org/10.1109/ICSME.2019.00092).
- [34] I. Kumara, M. Garriga, A. U. Romeu, D. Di Nucci, F. Palomba, D. A. Tamburri και W.-J. van den Heuvel. «The do's and don'ts of infrastructure code: A systematic gray literature review». Στο: *Information and Software Technology* 137 (2021), σ. 106593. DOI: [10.1016/j.infsof.2021.106593](https://doi.org/10.1016/j.infsof.2021.106593).

- [35] A. Agarwal, A. Agarwal, D. K. Verma, D. Tiwari και R. Pandey. «A Review on Software Development Life Cycle». Στο: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 9.3 (2023), σσ. 384–388. DOI: [10.32628/CSEIT2390387](https://doi.org/10.32628/CSEIT2390387).
- [36] R. S. Ghumatkar και A. Date. «Software Development Life Cycle (SDLC)». Στο: *International Journal for Research in Applied Science and Engineering Technology* 11.11 (2023), σσ. 1162–1165. DOI: [10.22214/ijraset.2023.56554](https://doi.org/10.22214/ijraset.2023.56554).
- [37] K. V. Kulkarni. *The pros and cons of infrastructure-as-code*. Red Hat, Inc. URL: <https://www.redhat.com/sysadmin/pros-and-cons-infrastructure-code>.
- [38] M. Kumar, S. Mishra, N. K. Lathar και P. Singh. «Infrastructure as Code (IaC): Insights on Various Platforms». Στο: *Sentiment Analysis and Deep Learning*. Επιμέλεια υπό S. Shakya, K.-L. Du και K. Ntalianis. Springer Nature, 2023, σσ. 439–449. DOI: [10.1007/978-981-19-5443-6\\_33](https://doi.org/10.1007/978-981-19-5443-6_33).
- [39] A. Rahman, R. Mahdavi-Hezaveh και L. Williams. «A systematic mapping study of infrastructure as code research». Στο: *Information and Software Technology* 108 (2019), σσ. 65–77. DOI: [10.1016/j.infsof.2018.12.004](https://doi.org/10.1016/j.infsof.2018.12.004).
- [40] *The Tower User Interface—Ansible Tower User Guide v3.8.6*. URL: [https://docs.ansible.com/ansible/latest/html/userguide/main\\_menu.html](https://docs.ansible.com/ansible/latest/html/userguide/main_menu.html).
- [41] Ravi Chaganti. *Azure Resource Manager—Introduction*. Ioύλ. 2020. URL: <https://ravichaganti.com/blog/azure-resource-manager-introduction/> (επίσκεψη 26/06/2024).
- [42] R. S. Ghumatkar και A. Date. «Software Development Life Cycle (SDLC)». Στο: *International Journal for Research in Applied Science and Engineering Technology* 11.11 (2023), σσ. 1162–1165. DOI: [10.22214/ijraset.2023.56554](https://doi.org/10.22214/ijraset.2023.56554).
- [43] M. D. Elradi. *Ansible: A Reliable Tool for Automation*.

