



ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Μηχανικοί DevOps: Από την θεωρία στην πράξη

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Αθανάσιου Βαρβούζου

A.M. 3252019003

Επιβλέπων : Ιωάννης Χαραλαμπίδης

Μέλη εξεταστικής επιτροπής:

Ιωάννης Χαραλαμπίδης, Καθηγητής
Ευριπίδης Λουκής, Καθηγητής
Σπύρος Κοκολάκης, Καθηγητής

Σάμος, Ιούλιος 2021

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πρόλογος και ευχαριστίες

Ευχαριστώ όσους με στήριξαν για την υλοποίηση αυτής της διπλωματικής καθώς και για την ευκαιρία να ασχοληθώ και να μάθω τι σημαίνει πραγματικά DevOps. Ευχαριστώ τον εισηγητή Ιωάννη Χαραλαμπίδη για την ευκαιρία επιλογής του συγκεκριμένου θέματος. Την εταιρία Intrasoft για την εμπιστοσύνη στο πρόσωπο μου για την σύσταση του τμήματος DevOps. Και τέλος θα ήθελα να αφιερώσω αυτό το Μεταπτυχιακό στους: Γιάννη, Κατερίνα, Μαριαλένα, Ελένη και Ανδρέα για την υπομονή και την ηθική στήριξη.

© 2021

Με επιφύλαξη παντός δικαιώματος

Αθανάσιος Βαρβούζος

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1	Εισαγωγή.....	1
1.1	Μηχανικοί DevOps στις Νέες Δομές Εταιριών	1
1.2	Αντικείμενο Διπλωματικής.....	1
1.3	Σκοπός.....	2
1.4	Δομή της Διπλωματικής	2
2	Αρχές και Αξίες DevOps για Μετασχηματισμό	3
2.1	Ιστορική Αναδρομή και Ορισμός.....	3
2.1.1	<i>Ορισμός DevOps.....</i>	<i>3</i>
2.2	Φάσεις παράδοσης του μοντέλου DevOps	4
2.3	Μεθοδολογίες που Στηρίχθηκε το DevOps.....	5
2.3.1	<i>Lean Movement</i>	<i>5</i>
2.3.2	<i>The Agile Manifesto</i>	<i>5</i>
2.3.3	<i>Agile Infrastructure and Velocity Movement</i>	<i>5</i>
2.3.4	<i>Continuous Delivery Movement.....</i>	<i>5</i>
2.3.5	<i>Toyota Kata</i>	<i>6</i>
2.3.6	<i>Βασικές Αρχές και Αξίες DevOps</i>	<i>7</i>
2.4	Αρχές της Ροής (Flow Principles) – Πρώτο Μονοπάτι	7
2.4.1	<i>Ορατότητα Εργασίας.....</i>	<i>8</i>
2.4.2	<i>Περιορισμός Εργασίας σε Εξέλιξη (Limit WIP).....</i>	<i>9</i>
2.4.3	<i>Μείωση Μεγεθών των Παραδοτέων.....</i>	<i>9</i>
2.4.4	<i>Μείωση των Εξαρτήσεων Μεταξύ των Ομάδων</i>	<i>9</i>
2.5	Αρχές Ανατροφοδότησης – Δεύτερο Μονοπάτι	9
2.5.1	<i>Ασφαλής Εργασία σε Πολύπλοκα Συστήματα.....</i>	<i>10</i>
2.5.2	<i>Έγκαιρος Εντοπισμός Προβλημάτων.....</i>	<i>10</i>
2.5.3	<i>Διαμοιρασμός της Γνώσης Μέσω Εντοπισμού Προβλημάτων</i>	<i>11</i>
2.5.4	<i>Εισαγωγή Ποιοτικών Ελέγχων σε Περισσότερα Τμήματα</i>	<i>11</i>
2.6	Συνεχής Μάθηση και Πειραματισμός – Τρίτο Μονοπάτι	12
2.6.1	<i>Υιοθέτηση Κουλτούρας για Συνεχή Μάθηση</i>	<i>12</i>
2.6.2	<i>Θεσμοθέτηση της Βελτίωσης της Καθημερινής Εργασίας.....</i>	<i>13</i>
2.6.3	<i>Μετασχηματισμός Τοπικών Ευρημάτων σε Γενικότερες Βελτιώσεις</i>	<i>13</i>
2.6.4	<i>Εισαγωγή Μεθόδων Ανθεκτικότητας</i>	<i>14</i>
3	Βέλτιστες Πρακτικές και Μεθοδολογία SAFe.....	15

3.1	Ανάλυση SWOT.....	15
3.1.1	Δυνατά Σημεία (Strengths).....	15
3.1.2	Αδύνατα Σημεία (Weaknesses).....	16
3.1.3	Ευκαιρίες (Opportunities).....	16
3.1.4	Απειλές (Threats).....	17
3.2	Θεμελιώδεις Πρακτικές DevOps.....	18
3.2.1	Επικοινωνία στην Παραδοσιακή Οργανωτική Δομή.....	18
3.2.2	Διαχείριση Διαμόρφωσης (Configuration Management)	18
3.2.3	Συνεχής Ενσωμάτωση (Continuous Integration).....	19
3.2.4	Εφαρμογή Στρατηγικής Δοκιμών	19
3.3	Διοχετεύσεις Ανάπτυξης (Deployment Pipelines).....	20
3.3.1	Ανατομία των Διοχετεύσεων/Αγωγών Ανάπτυξης	20
3.3.2	Δόμηση και Δέσμη Ενεργειών Ανάπτυξης.....	21
3.3.3	Στάδιο Δέσμευσης	21
3.3.4	Αυτοματοποιημένη Αποδοχή Δοκιμών	22
3.3.5	Ανάπτυξη και Παράδοση Εφαρμογής.....	24
3.4	Οικοσύστημα Συνεχούς Παράδοσης.....	26
3.4.1	Διαχείριση Υποδομών και Περιβαλλόντων.....	26
3.4.2	Υπολογιστικό Νέφος (Cloud Computing).....	26
3.4.3	Μονολιθική Αρχιτεκτονική	27
3.4.4	Αρχιτεκτονική Μικροϋπηρεσιών	27
3.5	Μεθοδολογία SAFe DevOps	27
3.5.1	Οι Εφτά Ικανότητες του SAFe.....	27
3.5.2	Χαρτογράφηση των Αξιών Ροής	29
3.5.3	DevOps Health Radar.....	30
4	Σύσταση Τμήματος DevOps.....	32
4.1	Δομές Τμημάτων DevOps	32
4.1.1	Εξωτερικές Ομάδες.....	32
4.1.2	Ενοποιημένες Ομάδες Ανάπτυξης και Λειτουργιών	32
4.1.3	Συνεργασία Ομάδας Ανάπτυξης και Λειτουργιών	32
4.1.4	Υποδομή ως Υπηρεσία (IaaS).....	33
4.1.5	DevOps στον Οργανισμό (εσωτερική ομάδα)	33
4.1.6	Μηχανικοί Αξιοπιστίας Ιστότοπου (Site Reliability Engineers).....	33
4.2	Κριτήρια Επιλογής Δομής DevOps	33
4.2.1	Αποφυγή Big Bang.....	33
4.2.2	Επίτευξη Μικρών Στόχων και Γρήγορες Νίκες	34

4.2.3	<i>Καθορισμός Προϊόντος και Υπηρεσίας Greenfield ή Brownfield</i>	34
4.2.4	<i>DevOps για Μεγάλους Οργανισμούς με Μονολιθικά Συστήματα</i>	34
4.2.5	<i>Καθορισμός Κατάλληλου Συστήματος Εγγραφών</i>	34
4.3	<i>Πιθανοί Ρόλοι DevOps στον Οργανισμό</i>	35
4.3.1	<i>DevOps Generalist</i>	35
4.3.2	<i>DevOps Executive</i>	36
4.3.3	<i>DevOps Project Manager</i>	36
4.3.4	<i>DevOps Product Owner</i>	37
4.3.5	<i>DevOps Architect</i>	37
4.3.6	<i>DevOps Developer</i>	38
4.3.7	<i>DevOps Operations Engineer</i>	38
4.3.8	<i>DevOps Quality Assurance Engineer</i>	38
4.3.9	<i>DevOps Information Security Engineer</i>	39
4.3.10	<i>DevOps Release Manager</i>	39
4.3.11	<i>DevOps Trainer</i>	40
4.3.12	<i>DevOps Coach</i>	40
4.4	<i>Δόμηση DevOps και Αρχιτεκτονικής</i>	40
4.4.1	<i>Δόμηση που Καθορίζει την Παράδοση Λογισμικού</i>	40
4.4.2	<i>Οργανισμοί Matrix και Λειτουργικές ομάδες</i>	41
4.4.3	<i>Ομάδες Προσανατολισμένες σε Προϊόντα και Υπηρεσίες</i>	42
4.4.4	<i>Χρηματοδότηση Οργανισμού σε Προϊόντα και Υπηρεσίες</i>	42
4.4.5	<i>Δημιουργία ομάδας DevOps και Αρχιτεκτονικής Λογισμικού</i>	43
4.4.6	<i>Μέγεθος Ομάδας DevOps</i>	43
4.5	<i>Στελέχωση και Ανθρώπινο Δυναμικό για DevOps</i>	44
4.5.1	<i>Στρατηγικός Προγραμματισμός Ανθρώπινων Πόρων</i>	45
4.5.2	<i>Ανάλυση Εργασίας</i>	46
4.5.3	<i>Προσέλκυση Υποψηφίων</i>	46
4.5.4	<i>Επιλογή Υποψηφίων</i>	47
4.5.5	<i>Εκπαίδευση και Ανάπτυξη Ανθρώπινων Πόρων</i>	47
4.5.6	<i>Αξιολόγηση Απόδοσης</i>	48
4.5.7	<i>Πολιτική και Συστήματα Αμοιβής</i>	48
4.6	<i>Μονοπάτια εκπαίδευσης DevOps</i>	49
4.6.1	<i>DevOps Beginner (Αρχάριος DevOps)</i>	49
4.6.2	<i>DevOps Intermediate (Ενδιάμεσος DevOps)</i>	51
4.6.3	<i>DevOps Advanced (Προχωρημένος DevOps)</i>	53
5	Υλοποίηση Αγωγού Ανάπτυξης CI/CD	55

5.1	Εισαγωγή και εργαλεία	55
5.2	Αγωγός Ανάπτυξης CI/CD με GIT, Jenkins και Maven	56
5.2.1	Εγκατάσταση Jenkins.....	56
5.2.2	Εγκατάσταση Git.....	68
5.2.3	Εγκατάσταση Maven.....	71
5.3	Tomcat server στον αγωγό ανάπτυξης.....	74
6	Μελέτη Περίπτωσης με SAFe.....	78
6.1	Χαρτογράφηση Αγωγού Ανάπτυξης.....	78
6.2	Παραδοχές και Αποτελέσματα Μετρήσεων	79
6.3	Μετρήσεις Health Radar	80
7	Πρόγραμμα Προπτυχιακών Σπουδών (ΠΠΣ)	82
7.1	Συνοπτική περιγραφή προγράμματος	82
7.1.1	Σε ποιους απευθύνεται.....	82
7.1.2	Σκοπός και Προσδοκώμενα Αποτελέσματα	82
7.1.3	Προαπαιτούμενα	82
7.2	Μαθήματα.....	82
7.2.1	Εισαγωγή στα DevOps (Θεωρητικό Κομμάτι)	82
7.2.2	Υλοποίηση παραδείγματος Αγωγού Ανάπτυξης (Πρακτικό κομμάτι).....	84
8	Συμπεράσματα.....	86
8.1	Γενικές Διαπιστώσεις.....	86
8.2	Βελτιώσεις με Βάση τα Αποτελέσματα της Μελέτης.....	87
8.2.1	Αυτοματοποίηση της Διαδικασίας Ανάπτυξης.....	87
8.2.2	Εισαγωγή Μικροϋπηρεσιών	87
8.2.3	Αυτοματοποίηση των Δοκιμών.....	88
8.2.4	Πολλαπλές Παραδόσεις στο FAT Περιβάλλον	88
8.2.5	Πολιτική Κλάδων (Branching Policy).....	89
8.2.6	Πιο Συχνή Παράδοση στο Παραγωγικό Περιβάλλον	89
8.2.7	Αναβάθμιση των Παραδοτέων σε Μονολιθικά Συστήματα.....	89
8.3	Σύνοψη	90
	ΒΙΒΛΙΟΓΡΑΦΙΑ	91
	ΠΑΡΑΡΤΗΜΑ.....	93

Λίστα Σχημάτων

Σχήμα 3-1: Οργανόγραμμα	18
Σχήμα 3-2: Επίπεδα στην αποδοχή δοκιμών ^[17]	22
Σχήμα 3-3: Μοντέλα Υπηρεσιών στο υπολογιστικό νέφος	26

Λίστα Εικόνων

Εικόνα 2-1: Φάσεις Παράδοσης DevOps Μοντέλου.....	4
Εικόνα 2-2: Τα τρία μονοπάτια ^[7]	7
Εικόνα 2-3: Παράδειγμα φυσικού πίνακα Kanban ^[10]	8
Εικόνα 2-4: Σχοινί Andon στην γραμμή παραγωγής ^[9]	11
Εικόνα 2-5: Ετοιμολογία Kaizen ^[12]	13
Εικόνα 3-1: Τεταρτημόρια Δοκιμών ^[19]	20
Εικόνα 3-2: Αλλαγές κατά την διοχέτευση ανάπτυξης ^[17]	21
Εικόνα 3-3: Διοχέτευση συνεχούς παράδοσης ^[20]	28
Εικόνα 3-4: Αξίες ροής με χρόνους υλοποίησης και καθυστερήσεις ^[20]	29
Εικόνα 3-5: Παράδειγμα χαρτογράφησης ^[22]	29
Εικόνα 3-6: DevOps Health Radar ^[20]	31
Εικόνα 4-1: Τρίγωνο Διαχείρισης Έργου	36
Εικόνα 5-1: Φάσεις DevOps για το παράδειγμα	55
Εικόνα 5-2: EC2 Amazon Instance	56
Εικόνα 5-3: Εισαγωγή στο διάλογο για δημιουργία διακομιστή AWS	57
Εικόνα 5-4: Επιλογή έκδοσης Λειτουργικού	57
Εικόνα 5-5: Δημιουργία Security Group για την ασφαλή πρόσβαση	58
Εικόνα 5-6: Δημιουργία keypair	59
Εικόνα 5-7: Ρυθμίσεις τερματικού MobaXterm	60
Εικόνα 5-8: Ο διακομιστής έπειτα την δημιουργία	60
Εικόνα 5-9: Είσοδος στον διακομιστή για την εγκατάσταση Jenkins	61
Εικόνα 5-10: Java home path	62
Εικόνα 5-11: Εκκίνηση Jenkins	63
Εικόνα 5-12: Ξεκλείδωμα Jenkins στον browser	64
Εικόνα 5-13: Είσοδος στο γραφικό περιβάλλον Jenkins	64
Εικόνα 5-14: Ρυθμίσεις Jenkins	65
Εικόνα 5-15: Διαμόρφωση Jenkins	65
Εικόνα 5-16: Δήλωση τοποθεσίας Java Home	66
Εικόνα 5-17: Δημιουργία Jenkins Job	66
Εικόνα 5-18: Ρυθμίσεις για το Build	67
Εικόνα 5-19: Εντολή δοκιμής echo	67
Εικόνα 5-20: Αποτελέσματα κονσόλας πρώτου Build	68
Εικόνα 5-21: Πρώτο Build	68
Εικόνα 5-22: Εγκατάσταση Git από το τερματικό (1/2)	69
Εικόνα 5-24: Εγκατάσταση Git από το τερματικό(2/2)	70
Εικόνα 5-23: Διαχείριση Plugins	70
Εικόνα 5-25: Εγκατάσταση plugin GitHub	71
Εικόνα 5-26: Global Tool Configuration path	71
Εικόνα 5-27: Ορισμός path για Maven	72
Εικόνα 5-28: Δημιουργία ενός Maven Job	73

Εικόνα 5-29: GitHub repository URL	73
Εικόνα 5-30: Ρυθμίσεις Build	74
Εικόνα 5-31: Είσοδος στον Tomcat.....	75
Εικόνα 5-32: Εγκατάσταση του Deploy to Container	75
Εικόνα 5-33: Δημιουργία του Maven Jenkins Job.....	75
Εικόνα 5-34: Configuration του Jenkins job (1/2).....	76
Εικόνα 5-35: Configuration του Jenkins job (2/2).....	76
Εικόνα 5-36: Σελίδα του webapp που κάναμε Deploy.....	77
Εικόνα 6-1: Χαρτογράφηση για την μελέτη περίπτωσης.....	79
Εικόνα 6-2: Μετρήσεις αγωγού ανάπτυξης ανάδοχου έργου Α	79
Εικόνα 6-3: Τιμές Health Radar μετά την βαθμολόγηση	81
Εικόνα 8-1: Πολιτική κλάδων στο GIT ^[23]	89

Ακρωνύμια

Ελληνικά

ΤΠ	Τεχνολογία της Πληροφορίας
----	----------------------------

Αγγλικά

%C&A	Percent Complete and Accurate
CD	Continuous Delivery or Continuous Deployment
CE	Continuous Exploration
CI	Continuous Integration
CR	Change Request
DevOps	D evelopment & O perations
FAT	Factory Acceptance Test
ITIL	Information Technology Infrastructure Library
LT	Lead Time
MMF	Minimum Marketable Feature
MVP	Minimum Viable Product
PT	Process Time
RoD	Release on Demand
SAFe	Scaled Agile Framework
SAT	Site Acceptance Test
SWOT	Strengths Weaknesses Opportunities Threats
WIP	Work In Progress

Περίληψη

Ο νέος ρόλος των μηχανικών DevOps εμφανίστηκε με την ραγδαία εξέλιξη της τεχνολογίας που επέφερε αλλαγή κουλτούρας στις εσωτερικές δομές των εταιριών Πληροφοριακών και Επικοινωνιακών συστημάτων. Οι μηχανικοί ακολουθώντας τις νέες τάσεις της ανάπτυξης λογισμικού (π.χ. Agile) καθώς και η πληθώρα εργαλείων ανοιχτού κώδικα ανέδειξε την ανάγκη αυτοματοποίησης των διαδικασιών για συνεχή ενσωμάτωση – συνεχή παράδοση (Continuous Integration / Continuous Delivery).

Σκοπός της εργασίας αποτελεί η ανάδειξη μεθοδολογιών και εργαλείων που εφαρμόστηκαν για να συσταθεί ένα νέο τμήμα DevOps, και πώς αυτές επηρεάζουν την λειτουργία άλλων τμημάτων (Developers, Project Managers, Analysts, Testers) της εταιρείας. Το τμήμα είναι νεοσύστατο και έχει ημερομηνία έναρξης αρχές του 2020. Η μεθοδολογία για την συγγραφή της εργασίας είναι μελέτη περίπτωσης βασισμένη σε βέλτιστες πρακτικές και μεθοδολογίες όπως το SAlFe DevOps.

Λέξεις Κλειδιά: *Μηχανικοί DevOps*, συνεχής ενσωμάτωση, συνεχής παράδοση

Abstract

Title: «DevOps Engineers: From theory to implementation»

The new role of DevOps engineers appeared with the rapid development of technology that brought a change of culture in the internal structures of Information and Communication systems companies. Engineers following the new trends in software development (e.g. Agile) as well as the plethora of open source tools have highlighted the need to automate processes for Continuous integration - Continuous Delivery.

The purpose of this paper is to highlight the methodologies and tools used to set up a new DevOps department, as well as the interactions with other departments (developers, project managers, analysts, testers). The department is newly established and has a start date of early 2020. The methodology for writing the paper is a case study based on best practices and methodologies such as SAFe DevOps.

Keywords: *DevOps Engineers, Continuous Integration, Continuous Delivery*

1 *Εισαγωγή*

1.1 *Μηχανικοί DevOps στις Νέες Δομές Εταιριών*

Διανύοντας πλέον την εποχή της ψηφιακού μετασχηματισμού, και έχοντας ως γνώμονα τη βελτίωση των πρακτικών σε οργανισμούς, αναγνωρίστηκε η ανάγκη στελέχωσης αυτόνομου τμήματος DevOps σε όλους τους οργανισμούς παροχής υπηρεσιών πληροφορικής, ανεξαρτήτως του μεγέθους τους. Πολλά ερωτήματα τέθηκαν εάν μπορεί να αυτονομηθεί και να δημιουργηθεί ένα ανεξάρτητο τμήμα που θα συγκεντρώνει τη γνώση των ισχυουσών πρακτικών ανάπτυξης και λειτουργίας λογισμικού ενσωματώνοντας καινούργιες εκδόσεις.

Ένα τμήμα DevOps σε καμία περίπτωση δεν αντικαθιστά τις αρχές και τις πρακτικές της ευέλικτης μεθοδολογίας (Agile Methodology), αλλά λειτουργεί παράλληλα και εναρμονισμένα με αυτές αποτελεσματικότερα και αποδοτικότερα, καθώς μικρότερα κομμάτια κώδικα υψηλής ποιότητας παραδίδονται σε συντομότερο χρονικό διάστημα. Επίσης, με την αυτοματοποίηση διαδικασιών, αρκετά κομμάτια που επιλύουν καίρια ζητήματα διαχείρισης υπηρεσιών πληροφορικής, και δεδομένου ότι οι πρακτικές DevOps απαιτούν γρήγορη αντίχρευση και επαναφορά του συστήματος σε περιπτώσεις διακοπής υπηρεσιών ή δυσλειτουργίας, κάνουν τις πρακτικές DevOps συμβατές με τις διαδικασίες ITIL.

1.2 *Αντικείμενο Διπλωματικής*

Αντικείμενο της διπλωματικής εργασίας αποτελεί:

(α) Η ανάλυση του όρου μηχανικός DevOps καθώς και τα βασικά συστατικά ενός τμήματος DevOps, εξαρτήσεις με άλλα τμήματα και διατάξεις ανάλογα και με το μέγεθος.

(β) Η ανάλυση, η υλοποίηση και η εφαρμογή των βέλτιστων πρακτικών και μεθοδολογιών καθώς και η παρουσίαση των αποτελεσμάτων τους.

(γ) Η εκτενής περιγραφή των εργαλείων για συνεχή ενσωμάτωση – συνεχή παράδοση (Continuous Integration / Continuous Delivery).

(δ) Η υλοποίηση μέρους προγράμματος σπουδών για την εκπαίδευση φοιτητών ΠΠΣ σχετικά με το αντικείμενο των μηχανικών DevOps.

1.3 Σκοπός

Σκοπός της παρούσας διπλωματικής εργασίας αποτελεί η ανάλυση και ανάδειξη των βέλτιστων πρακτικών για τη στελέχωση/σύσταση ενός Τμήματος Μηχανικών DevOps, καθώς και η παρουσίαση των αλληλεπιδράσεων του με άλλα Τμήματα του Οργανισμού. Βασίζεται σε μελέτη περίπτωσης νεοσύστατου DevOps Τμήματος Πολυεθνικής Εταιρείας.

1.4 Δομή της Διπλωματικής

Στο πρώτο κεφάλαιο της διπλωματικής εισάγεται ο σκοπός, το αντικείμενο και το πλαίσιο δομής της. Στο δεύτερο κεφάλαιο γίνεται ιστορική αναδρομή και αναφορά στις μεθοδολογίες που στηρίχτηκε το DevOps καθώς και ανάλυση των αρχών και των αξιών ροής DevOps για το μετασχηματισμό. Στο τρίτο κεφάλαιο παρατίθενται οι βέλτιστες πρακτικές και η μεθοδολογία SAFe για τη μελέτη περίπτωσης. Στο τέταρτο κεφάλαιο βρίσκονται τα βασικά συστατικά στοιχεία για τη σύσταση ενός τμήματος DevOps με παραδείγματα γνωστών εταιριών. Στο πέμπτο κεφάλαιο παρουσιάζεται η υλοποίηση αγωγού ανάπτυξης. Στο έκτο κεφάλαιο παρατίθενται τα αποτελέσματα της μελέτης βασισμένα στην μεθοδολογία του τρίτου κεφαλαίου. Στο έβδομο κεφάλαιο παρουσιάζεται ένα προτεινόμενο πρόγραμμα προπτυχιακών σπουδών για την ενίσχυση των γνώσεων μηχανικών DevOps με βάση το θεωρητικό αλλά και πρακτικό κομμάτι που αναπτύχθηκε στα προηγούμενα κεφάλαια (3^ο έως και 5^ο) της εργασίας. Τέλος, στο όγδοο κεφάλαιο αναλύονται τα συμπεράσματα και οι προοπτικές με βάση τα αποτελέσματα της μελέτης περίπτωσης του έκτου κεφαλαίου.

2 *Αρχές και Αξίες DevOps για Μετασχηματισμό*

Στο δεύτερο κεφάλαιο, «Αρχές και Αξίες DevOps για Μετασχηματισμό», γίνεται μια ιστορική αναδρομή και δίνεται ο ορισμός DevOps. Στη συνέχεια γίνεται αναφορά στις οκτώ φάσεις μοντέλου DevOps αλλά και στις μεθοδολογίες που στηρίχθηκε το DevOps. Τέλος γίνεται ανάλυση των βασικών αρχών πάνω στις οποίες στηρίζεται το DevOps.

Ο σκοπός αυτού το κεφαλαίου είναι να δώσει το θεωρητικό υπόβαθρο για την κατανόηση των βασικών εννοιών σχετικά με το DevOps.

2.1 *Ιστορική Αναδρομή και Ορισμός*

Στις αρχές του 2000 οι μηχανικοί πληροφοριακών συστημάτων, εκμεταλλευόμενοι τις τεχνολογίες της εποχής καθώς και τις νέες αρχές και πρακτικές της μεθοδολογίας Agile, κατάφεραν να μειώσουν σε εβδομάδες ή μήνες το χρόνο ανάπτυξης νέων λειτουργιών. Η ενσωμάτωση, όμως, των νέων αυτών πρακτικών στην παραγωγή δεν ήταν πάντα πετυχημένη, δημιουργώντας αρκετές φορές καταστροφικά αποτελέσματα.^[1]

Από το 2010 έχουμε την εισαγωγή των μηχανικών DevOps σε οργανισμούς που είχαν ως σκοπό την παράδοση νέων εκδόσεων στην παραγωγή, μέσω διαδικασιών ρουτίνας χαμηλού ρίσκου. Σήμερα, σε πολλούς οργανισμούς γίνονται εκατοντάδες, ακόμα και χιλιάδες, παραδόσεις αλλαγών κατά τη διάρκεια μιας ημέρας.

2.1.1 *Ορισμός DevOps*

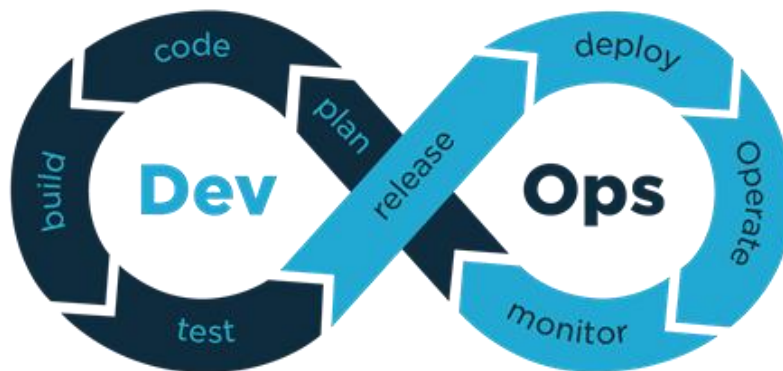
Το DevOps είναι μια νέα μεθοδολογία που συνδυάζει προγραμματιστές (Dev) και ομάδες λειτουργιών (Ops) ενσωματώνοντας διαδικασίες και τεχνολογία για μια αυτοματοποιημένη παράδοση λογισμικού που είναι ευέλικτη, κλιμακούμενη και οικονομικά αποδοτική.^[25]

Η παράδοση λογισμικού μέσω DevOps είναι ένας συνδυασμός δύο προηγούμενων μεθόδων: της ευέλικτης ανάπτυξης λογισμικού και της συνεργασίας μεταξύ προγραμματιστών και ομάδων λειτουργιών, αλλάζοντας τον τρόπο σκέψης για ανάπτυξη μιας υποδομής σε ολόκληρο τον κύκλο ζωής του προϊόντος, από το σχεδιασμό έως τη φάση της παραγωγής με συνεπή και επαναλαμβανόμενα σχέδια.

Σύμφωνα με τον DevOps Μάνατζερ της Microsoft Ντόνοβαν Μπράουν: «DevOps είναι η ένωση ανθρώπων, διεργασιών και προϊόντων για την ενεργοποίηση της αξίας της συνεχούς παράδοσης στους τελικούς χρήστες.»^[2]

2.2 Φάσεις παράδοσης του μοντέλου DevOps

Οι φάσεις παράδοσης του μοντέλου DevOps έχουν πολλές ομοιότητες με αυτό του μοντέλου της ευέλικτης μεθοδολογίας στην οποία έχει βασιστεί. Στο επόμενο κεφάλαιο με τις βέλτιστες πρακτικές και την μεθοδολογία SAFe θα γίνει εκτενέστερη αναφορά σε κάποιες από τις παρακάτω οκτώ φάσεις όπου μηχανικοί DevOps έχουν ενεργή συμμετοχή.



Εικόνα 2-1: Φάσεις Παράδοσης DevOps Μοντέλου

Οι φάσεις του μοντέλου DevOps είναι:

- **Σχεδιασμός (Plan):** Σε αυτό το στάδιο οι business owners και οι προγραμματιστές δημιουργούν το πλάνο για την επικείμενη παράδοση λογισμικού.
- **Δημιουργία Κώδικα (Code):** Οι προγραμματιστές σχεδιάζουν και δημιουργούν τα κομμάτια του κώδικα που τους έχουν ανατεθεί χρησιμοποιώντας εργαλεία όπως το GitHub για να τα αποθηκεύσουν.
- **Δόμηση (Build):** Εργαλεία όπως το Maven συλλέγουν τον κώδικα από διάφορα αποθετήρια (repositories) και τα συνδυάζουν για να δομήσουν την εφαρμογή.
- **Δοκιμή (Test):** Η εφαρμογή περνάει στο στάδιο της δοκιμής και διασφάλισης ποιότητας με αυτοματοποιημένα εργαλεία όπως το Selenium.
- **Διάθεση (Release):** Όταν οι δοκιμές έχουν τελειώσει επιτυχώς, τα νέα κομμάτια κώδικα με νέες λειτουργίες ή διορθώσεις ενσωματώνονται στην υπάρχουσα βάση του κώδικα. Ένα από τα εργαλεία για συνεχή ενσωμάτωση είναι και το Jenkins.
- **Ανάπτυξη (Deploy):** Μόλις ολοκληρωθεί η διάθεση της εφαρμογής μεταφέρεται από το περιβάλλον δοκιμών στο παραγωγικό περιβάλλον.
- **Λειτουργία (Operate):** Όταν η εφαρμογή έχει πλέον παραδοθεί, η ομάδα λειτουργιών διασφαλίζει ότι οι υποδομές και οι αναγκαίοι πόροι είναι διαθέσιμοι για την απρόσκοπτη λειτουργία της. Η διάθεση των πόρων και η υποστήριξη γίνεται με εργαλεία όπως το CHEF ή η ANSIBLE.
- **Παρακολούθηση (Monitor):** Η παρακολούθηση δίνει τη δυνατότητα για εντοπισμό συγκεκριμένων ζητημάτων σε διαφορετικές εκδόσεις καθώς και κατανόηση της επιρροής στους τελικούς χρήστες. Εργαλεία παρακολούθησης είναι το Splunk και το Nagios.

2.3 Μεθοδολογίες που Στηρίχθηκε το DevOps

2.3.1 Lean Movement

Κατά τη δεκαετία του '80 τεχνικές όπως το "Value Stream Mapping", "Kanban Boards" και "Total Productive Maintenance", υιοθετήθηκαν από την Toyota για την γραμμή παραγωγής. Αργότερα, το 1997, το ινστιτούτο Lean Enterprise άρχισε την έρευνα για την υιοθέτηση του Lean και σε άλλους τομείς, στην ευρύτερη βιομηχανία και στον τομέα της υγείας.

Δύο από τις βασικές αρχές της Lean είναι ο χρόνος παραγωγής και η κατάτμηση της εργασίας σε μικρά κομμάτια. Επικεντρώνεται στον τρόπο με τον οποίο θα δημιουργηθεί αξία στον πελάτη μέσω της σταθερότητας του τελικού σκοπού, της σκέψης των συστημάτων, της συνεχούς ροής και της ποιότητας με σεβασμό στον ανθρώπινο παράγοντα.

2.3.2 The Agile Manifesto

Το Agile Manifesto δημιουργήθηκε το 2001 από δεκαεπτά από τους κορυφαίους στοχαστές στην ανάπτυξη λογισμικού. Ήθελαν να δημιουργήσουν ένα ελαφρύ σύνολο αξιών και αρχών ενάντια σε βαριές διαδικασίες ανάπτυξης λογισμικού, όπως η ανάπτυξη καταρράκτη, και μεθοδολογίες όπως η Rational Unified Process.

Μία βασική αρχή ήταν η ταχύτερη παροχή λογισμικού, από μερικές εβδομάδες έως δύο μήνες, με προτίμηση στο μικρότερο χρονικό διάστημα και με έμφαση στα μικρά κομμάτια κώδικα και τις σταδιακές εκδόσεις.

Έμφαση δόθηκε στην ανάγκη για μικρές, αυτόνομες ομάδες, που θα εργάζονται σε ένα μοντέλο υψηλής εμπιστοσύνης. Η αύξηση της παραγωγικότητας πολλών οργανισμών λογισμικού έχει επιτευχθεί με την εφαρμογή της μεθοδολογίας Agile.

2.3.3 Agile Infrastructure and Velocity Movement

Το 2008 στο συνέδριο Agile που πραγματοποιήθηκε στο Τορόντο του Καναδά παρουσιάστηκε η εφαρμογή των αρχών Agile στις υποδομές, όχι όμως και στον κώδικα της εφαρμογής. Αργότερα, στο συνέδριο Velocity το 2009, τέθηκαν οι κοινοί στόχοι μεταξύ Dev και Ops και οι πρακτικές συνεχούς ενσωμάτωσης, ως μέρος της καθημερινής εργασίας στην Flickr. Οι συμμετέχοντες στο συνέδριο κατάλαβαν ότι αυτή η παρουσίαση ήταν ιστορικής σημασίας. Την ίδια χρονιά δημιουργήθηκε και ο όρος DevOps από τον Patrick Debois, κατά τη διάρκεια του πρώτου DevOpsDay στη Γάνδη του Βελγίου.^[5]

2.3.4 Continuous Delivery Movement

Με βάση τις αρχές της δημιουργίας, δοκιμής και ενσωμάτωσης προχώρησε η εισαγωγή της έννοιας της συνεχούς παράδοσης, μέσω διοχέτευσης ανάπτυξης (deployment pipeline) η οποία διασφαλίζει ότι και ο κώδικας αλλά και η υποδομή θα είναι πάντα σε κατάσταση ετοιμότητας και ότι τα κομμάτια κώδικα θα έχουν εκλεχθεί πριν την ενσωμάτωση στην παράγωγή.

Μπορούμε να ενεργοποιήσουμε τη συνεχή παράδοση μέσω της διοχέτευσης ανάπτυξης με τις τρεις βασικές λειτουργικότητες της:^[6]

- Ορατότητα - Κάνει ορατές όλες τις πτυχές του συστήματος παράδοσης, συμπεριλαμβανομένης της δημιουργίας, της ανάπτυξης, της δοκιμής και της ενσωμάτωσης σε κάθε μέλος της ομάδας αλλά και σε άλλες ομάδες. Προσφέρει διαφάνεια στους πελάτες και τους ενδιαφερόμενους.
- Ανατροφοδότηση - Τα μέλη της ομάδας μαθαίνουν για προβλήματα μόλις εμφανιστούν, ώστε να μπορούν να τα επιδιορθώσουν άμεσα προτού εγκαταστήσουν ένα άλλο νέο τμήμα κώδικα.
- Συνεχής ανάπτυξη - Ανάπτυξη και ενσωμάτωση οποιασδήποτε έκδοσης του λογισμικού σε οποιοδήποτε περιβάλλον (συμπεριλαμβανομένων δοκιμαστικών και παραγωγικών περιβαλλόντων), μέσω μιας πλήρως αυτοματοποιημένης διαδικασίας.

2.3.5 Toyota Kata

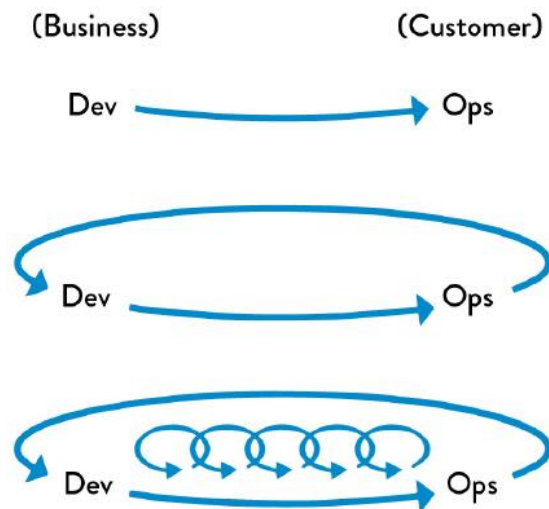
Η βελτίωση Toyota Kata χρησιμοποιείται ως ρουτίνα για να μετακινηθούμε από την τρέχουσα κατάσταση σε μια νέα, με δημιουργικό, κατευθυνόμενο και ουσιαστικό τρόπο. Βασίζεται σε ένα μοντέλο που αποτελείται από τέσσερα μέρη:^[6]

- Εκτίμηση ενός μεγαλύτερου οράματος ή βελτιωμένης κατεύθυνσης.
- Κατανόηση των τρεχουσών συνθηκών.
- Ορισμός των συνθηκών του επόμενου στόχου.
- Συνεχής και επαναληπτική προσπάθεια για να προχωρήσουμε προς αυτήν την κατάσταση στόχου.

Κατά τη διάρκεια αυτής της διαδρομής ανακαλύπτουμε εμπόδια που πρέπει να επιλυθούν και τα επιλύουμε. Σε αντίθεση με άλλες προσεγγίσεις βελτίωσης που προσπαθούμε να προβλέψουμε τη διαδρομή και να εστιάσουμε στην εφαρμογή, με την βελτίωση Toyota Kata μαθαίνουμε να βασιζόμαστε στην ανακάλυψη που συμβαίνει στην πορεία. Χρησιμοποιούμε τη βελτίωση Toyota Kata για να μάθουμε, καθώς προσπαθούμε να φτάσουμε σε μια στοχευμένη κατάσταση και την προσαρμόζουμε με βάση αυτό που μαθαίνουμε.

2.3.6 Βασικές Αρχές και Αξίες DevOps

Τον Αύγουστο του 2012 ο Gene Kim παρέθεσε ένα άρθρο^[7] το οποίο αργότερα το συμπεριέλαβε και στα βιβλία του σχετικά με τα τρία μονοπάτια, ως σύνολο των βασικών αρχών που προκύπτουν από συμπεριφορές και μοτίβα DevOps.



Εικόνα 2-2: Τα τρία μονοπάτια^[7]

Με μία σύντομη περιγραφή:

- **Αρχές Ροής:** Το πρώτο μονοπάτι επιτρέπει τη γρήγορη ροή εργασίας, από αριστερά προς τα δεξιά, από το τμήμα της Ανάπτυξης στο τμήμα Λειτουργιών του πελάτη.
- **Αρχές Ανατροφοδότησης:** Το δεύτερο μονοπάτι επιτρέπει τη γρήγορη και συνεχή ροή ανατροφοδότησης, από δεξιά προς τα αριστερά, σε όλα τα στάδια της ροής αξιών μας.
- **Συνεχής Μάθηση και Πειραματισμός:** Το τρίτο μονοπάτι επιτρέπει τη δημιουργία μιας κουλτούρας υψηλής εμπιστοσύνης που υποστηρίζει μια δυναμική, πειθαρχημένη και επιστημονική προσέγγιση στον πειραματισμό και την ανάληψη κινδύνων, διευκολύνοντας τη δημιουργία οργανωτικής μάθησης, τόσο από τις επιτυχίες όσο και από τις αποτυχίες μας.

Στα επόμενα κεφάλαια θα αναπτυχθούν εκτενέστερα οι πτυχές που θα πρέπει να επικεντρωθούμε για να έχουμε τα μέγιστα δυνατά θετικά αποτελέσματα σε κάθε ένα μονοπάτι ξεχωριστά.

2.4 Αρχές της Ροής (Flow Principles) – Πρώτο Μονοπάτι

Οι αξίες της ροής στην τεχνολογία ρέουν από το τμήμα της Ανάπτυξης στο τμήμα Λειτουργιών, τους αδιαμφισβήτητους λειτουργικούς τομείς μεταξύ μιας επιχείρησης και των πελατών της. Απαιτείται γρήγορη και ομαλή ροή από το τμήμα της Ανάπτυξης στο τμήμα Λειτουργιών, για να παρέχει αξία στους πελάτες σε σύντομο χρονικό διάστημα.

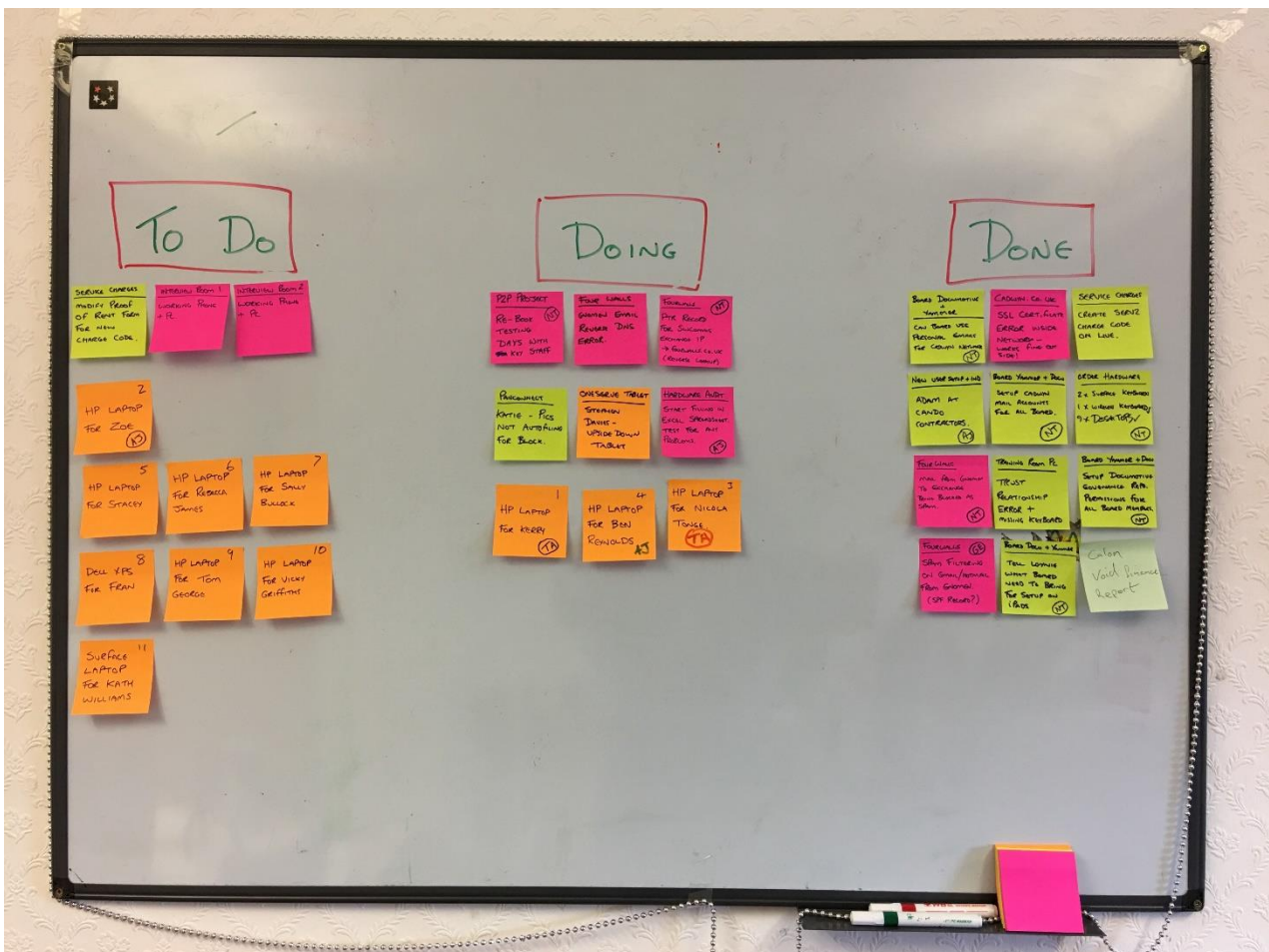
Μπορούμε να αυξήσουμε την ροή κάνοντας την εργασία ορατή, μειώνοντας το μέγεθος των παραδοτέων κομματιών, τα διαστήματα εργασίας και αναβαθμίζοντας την ποιότητα, βέβαιοι ότι δε θα παραδοθούν ελλειπτικά κομμάτια. Αυξάνοντας την ροή μέσω των αξιών ροής, μειώνουμε το χρόνο παράδοσης που απαιτείται για την πλήρωση εσωτερικών ή εξωτερικών αιτημάτων πελατών, αυξάνουμε την ποιότητα καθιστώντας μας ευέλικτους και σε θέση να πειραματιστούμε σε σχέση με τον ανταγωνισμό.

Στόχος μας είναι να μειώσουμε το χρόνο που απαιτείται για τις αλλαγές που θα χρησιμοποιηθούν στην παραγωγή και να αυξήσουμε την αξιοπιστία και την ποιότητα αυτών των υπηρεσιών. Ενδείξεις για το πώς πραγματοποιούνται σοι αξίες της ροής στην τεχνολογία μπορούν να αντληθούν από το πώς εφαρμόστηκαν οι αρχές του Lean στις αξίες της ροής στην βιομηχανία.

2.4.1 Ορατότητα Εργασίας

Μια από τις βασικές διαφορές που έχουν οι αξίες ροής μεταξύ της βιομηχανίας και της τεχνολογίας υπολογιστών είναι η ορατότητα. Σε αντίθεση με τις φυσικές διαδικασίες, είναι δύσκολο να εντοπίσουμε το στάδιο που βρίσκεται η εργασία μας. Παρ' όλα αυτά μπορούμε με τη βοήθεια της τεχνολογίας και με το πάτημα ενός κουμπιού να μεταθέσουμε την εργασία μας σε κάποιο άλλο άτομο ή Τμήμα. Αυτό αυξάνει τους κινδύνους να μην παραδοθεί στην παραγωγή η εργασία μας ή να έχουμε επιπλέον καθυστερήσεις, καθώς η ευθύνη υλοποίησης αλλάζει με εύκολο σχετικό τρόπο.

Για να επιτύχουμε ορατότητα της εργασίας μας μπορούμε να χρησιμοποιήσουμε μεθόδους οπτικοποίησης με φυσικούς (Εικόνα 2-2) ή ηλεκτρονικούς πίνακες Kanban και να απεικονίσουμε με κάρτες τις εργασίες οι οποίες ξεκινούν από την αριστερή στήλη που είναι προς υλοποίηση, πηγαίνουν στην μέση στο αντίστοιχο τμήμα που το αναλαμβάνει, και καταλήγει στη δεξιά πλευρά που είναι η στήλη με τις ολοκληρωμένες εργασίες.



Εικόνα 2-3: Παράδειγμα φυσικού πίνακα Kanban^[10]

2.4.2 Περιορισμός Εργασίας σε Εξέλιξη (Limit WIP)

Στη βιομηχανία η καθημερινή εργασία συνήθως καθορίζεται από το πρόγραμμα παραγωγής το οποίο εμπεριέχει τις εργασίες που θα πρέπει να γίνουν βάση των παραγγελιών, τις προθεσμίες, τα αποθέματα αλλά και λοιπούς παράγοντες που μπορεί να τη μεταβάλλουν.

Στο κομμάτι της τεχνολογίας και ειδικότερα στην ανάπτυξη λογισμικού τα πράγματα είναι πιο δυναμικά καθώς οι ομάδες εργασίας πρέπει να ικανοποιήσουν τις ανάγκες από διαφορετικούς εμπλεκόμενους. Αυτό έχει ως αποτέλεσμα η καθημερινή εργασία να καθορίζεται από προτεραιότητες που μπορεί να παρουσιαστούν από διάφορα μέσα όπως ηλεκτρονική αλληλογραφία (email), συστήματα καταγραφής εργασιών (ticketing system), τηλέφωνα και ιεραρχική κλιμάκωση (hierarchy escalation).

Αυτές οι προτεραιότητες μπορεί να διακόψουν την ροή του μηχανικού που δουλεύει ήδη σε κάποια εργασία που του έχει ανατεθεί. Για να επιτύχουμε τον περιορισμό των πολλαπλών εργασιών (multitasking) μπορούμε να χρησιμοποιήσουμε τον πίνακα Kanban (Εικόνα 2-2) και να θέσουμε όρια σε κάθε στήλη για την εργασία σε εξέλιξη (WIP), με περιορισμό στον ανώτερο αριθμό καρτών που θα μπορούσαν να μπουν στην στήλη.^[21]

2.4.3 Μείωση Μεγεθών των Παραδοτέων

Για την ανάπτυξη λογισμικού το παραδοτέο είναι ο κώδικας. Υπάρχουν πολλές τεχνικές για τον έλεγχο των παραδοτέων, από μικρά κομμάτια που είναι απαραίτητα για τη συνεχή παράδοση έως και παραδοσιακά βασιζόμενα σε κλάδους (branches) τμήματα όπου κώδικας από διαφορετικούς μηχανικούς ανάπτυξης λογισμικού αποστέλλονται και τελικά ενσωματώνονται όλα μαζί.

Το ισοδύναμο με τη ροή ενός κομματιού στις αξίες της ροής στην τεχνολογία πραγματοποιείται με συνεχή ανάπτυξη, όπου κάθε αλλαγή δεσμεύεται με έλεγχο έκδοσης (version control) και ενσωματώνεται, δοκιμάζεται και αναπτύσσεται στην παραγωγή.

2.4.4 Μείωση των Εξαρτήσεων Μεταξύ των Ομάδων

Στις αξίες της ροής στην τεχνολογία συχνά απαιτούνται μεγάλα χρονικά διαστήματα για την ανάπτυξη στο παραγωγικό περιβάλλον. Αυτό οφείλεται στις εκατοντάδες ή και χιλιάδες λειτουργίες που χρειάζονται για να παραδοθεί ο κώδικας από το στάδιο του ελέγχου έκδοσης έως και την παραγωγή.

Για να επιτύχουμε τη μείωση των εξαρτήσεων μπορούμε να αυτοματοποιήσουμε πολλά από τα μέρη των λειτουργιών, καθώς και να αναδιοργανώσουμε τις ομάδες ώστε να προσφέρουν αξία στον πελάτη ελαττώνοντας παράλληλα και τις αλληλεξαρτήσεις μεταξύ των ομάδων. Ως αποτέλεσμα, αυξάνουμε τη ροή μειώνοντας το χρόνο που καταναλώνει η εργασία μας στην αναμονή, καθώς και το χρόνο μη προστιθέμενης αξίας.

2.5 Αρχές Ανατροφοδότησης – Δεύτερο Μονοπάτι

Ενώ το πρώτο μονοπάτι περιγράφει τις αρχές που επιτρέπουν τη γρήγορη ροή της εργασίας από αριστερά προς τα δεξιά, το δεύτερο μονοπάτι περιγράφει τις αρχές που επιτρέπουν την

αμοιβαία, γρήγορη και συνεχή ανατροφοδότηση από δεξιά προς τα αριστερά σε όλα τα στάδια των αξιών ροής. Στόχος είναι η δημιουργία ενός ασφαλέστερου και ανθεκτικότερου συστήματος.

Κάνουμε το σύστημα ασφαλέστερο δημιουργώντας γρήγορη, συχνή και υψηλής ποιότητας ροή πληροφοριών σε όλα τα στάδια των αξιών ροής, που περιλαμβάνει σχόλια και βρόχους ανατροφοδότησης. Έτσι, εντοπίζουμε και αποκαθιστούμε σημαντικά προβλήματα, ενώ όσα από αυτά παρουσιαστούν αργότερα κοστίζουν λιγότερο και είναι ευκολότερο να διορθωθούν, αποτρέποντας έτσι πιθανή σοβαρή βλάβη, εμπλουτίζοντας παράλληλα τη γνώση για μελλοντικά πιθανά όμοια περιστατικά.

2.5.1 Ασφαλής Εργασία σε Πολύπλοκα Συστήματα

Ο σχεδιασμός απόλυτα ασφαλών συστημάτων είναι πολύ δύσκολο να επιτευχθεί, μπορούμε όμως να κάνουμε ασφαλέστερη την εργασία σε πολύπλοκα συστήματα όταν πληρούνται οι τέσσερις ακόλουθες προϋποθέσεις:^[1]

- Η πολύπλοκη εργασία διαχειρίζεται ώστε να εντοπίζονται τα προβλήματα κατά τον σχεδιασμό και την λειτουργία.
- Τα προβλήματα εντοπίζονται και επιλύονται με αποτέλεσμα την άμεση δημιουργία νέας γνώσης.
- Η νέα γνώση αξιοποιείται από όλο τον οργανισμό.
- Οι ηγέτες προωθούν και εκκολάπτουν νέους ηγέτες που υιοθετούν και αναπτύσσουν αυτές τις ικανότητες.

2.5.2 Έγκαιρος Εντοπισμός Προβλημάτων

Σε ένα ασφαλές σύστημα εργασίας, πρέπει να δοκιμάζουμε συνεχώς τις υποθέσεις σχεδιασμού και λειτουργίας. Στόχος είναι να αυξήσουμε τη ροή πληροφοριών στο σύστημά μας σε όσο το δυνατόν περισσότερες περιοχές, νωρίτερα, ταχύτερα, φθηνότερα και με όσο το δυνατόν μεγαλύτερη σαφήνεια μεταξύ αιτίας και αποτελέσματος. Όσο περισσότερες υποθέσεις μπορούμε να αποκλείσουμε, τόσο πιο γρήγορα μπορούμε να βρούμε και να διορθώσουμε προβλήματα, αυξάνοντας την ανθεκτικότητα, την ευελιξία και την ικανότητά μας να μάθουμε και να καινοτομούμε.

Για την επίτευξη του στόχου πρέπει να δημιουργήσουμε γρήγορη ανατροφοδότηση και γρήγορους βρόχους. Αυτό περιλαμβάνει τη δημιουργία αυτοματοποιημένων διαδικασιών κατασκευής, ενσωμάτωσης και δοκιμών, ώστε να μπορούμε να εντοπίσουμε αμέσως πότε πραγματοποιήθηκε η αλλαγή που μας προκάλεσε το πρόβλημα. Επίσης, με μηχανισμούς τηλεμετρίας μπορούμε να δούμε πως λειτουργούν τα στοιχεία του συστήματος στην παραγωγή.

2.5.3 Διαμοιρασμός της Γνώσης Μέσω Εντοπισμού Προβλημάτων

Το 1990 στο σύστημα παραγωγής της Toyota έγινε αναφορά στο σχοινί Andon (Εικόνα 2-3) με το οποίο οποιοσδήποτε υπάλληλος μπορούσε να σταματήσει την γραμμή παραγωγής με ένα απλό τράβηγμα στην περίπτωση εντοπισμού προβλήματος κατά τη διαδικασία παραγωγής ενός οχήματος. Ο υπεύθυνος της ομάδας μαζί με τα μέλη έπρεπε να επιλύσουν το πρόβλημα προτού ξεκινήσει ξανά η παραγωγή. Μπορεί να ακούγεται ως μια απότομη και μη παραγωγική μέθοδος αλλά τα αποτελέσματα της ήταν άκρως επιτυχημένα καθώς προβλήματα που εντοπίζονταν επιλύονταν εγκαίρως.^[8]



Εικόνα 2-4: Σχοινί Andon στην γραμμή παραγωγής^[9]

Έτσι λοιπόν και στην παραγωγή λογισμικού, για να έχουμε γρήγορη και έγκαιρη ανατροφοδότηση στις αξίες της ροής, πρέπει να δημιουργήσουμε αντίστοιχο σχοινί Andon (με ηλεκτρονικά μέσα) και να δίνουμε την δυνατότητα στους μηχανικούς να «τραβούν» το σχοινί, και στους υπεύθυνους των ομάδων να συνεργάζονται ώστε να επιλύσουν το πρόβλημα. Με αυτό τον τρόπο

αποτρέπουμε την εισαγωγή νέων εργασιών μέχρι την επίλυση. Η αποτροπή εισαγωγής νέων εργασιών συμβάλλει στη συνεχή ενσωμάτωση και ανάπτυξη καθώς όλες οι αλλαγές που περνούν επιτυχώς από τις δοκιμές, παραδίδονται στην παραγωγή ενώ αυτές που αποτυγχάνουν ενεργοποιούν το σχοινί Andon μέχρι να επιλυθούν.

Με τη συγκέντρωση των ομάδων διαμοιράζεται η γνώση μεταξύ των μελών. Για να επιτύχει αυτό το εγχείρημα θα πρέπει να δημιουργηθεί κουλτούρα για την ασφαλή λειτουργία αυτού του εργαλείου και ενθάρρυνση προς όλους για την χρήση του.

2.5.4 Εισαγωγή Ποιοτικών Ελέγχων σε Περισσότερα Τμήματα

Σε πολλές περιπτώσεις υπάρχουν αστοχίες που αφορούν πολύπλοκα συστήματα επειδή στους οργανισμούς υπάρχουν μέθοδοι ελέγχου και εγκρίσεων που λειτουργούν με γραφειοκρατικές διαδικασίες. Σε πολύπλοκα συστήματα συνήθως υπάρχουν περισσότερα βήματα ελέγχων και διαδικασίες έγκρισης που αυξάνουν την πιθανότητα αστοχίας. Η λήψη αποφάσεων βασίζεται σε τμήματα ή ανθρώπους που βρίσκονται σε μεγάλη οργανωτική απόσταση από το σημείο όπου εκτελείται μία εργασία, με αποτέλεσμα την αύξηση του χρόνου του κύκλου ζωής της απόφασης και την μείωση της ανατροφοδότησης μεταξύ αιτίας και αποτελέσματος.

Για να αποτρέψουμε τέτοιες αστοχίες πρέπει να μεταθέσουμε την επίλυση προβλημάτων σε κάθε εμπλεκόμενη ομάδα, ως μέρος της καθημερινότητάς τους. Με αυτόν τον τρόπο εισάγουμε επιμέρους ποιοτικούς ελέγχους και τη λήψη αποφάσεων πιο κοντά στο σημείο όπου εκτελείται η

εργασία και δεν περιμένουμε έγκριση από κάποιο στέλεχος που είναι οργανωτικά υψηλότερα. Χρησιμοποιούμε αξιολογήσεις από συναδέλφους για την διαβεβαίωση ότι η αλλαγή μας θα λειτουργήσει όπως τη σχεδιάσαμε.

Η αυτοματοποίηση των ποιοτικών ελέγχων καθώς και οι δοκιμές από τους ίδιους τους προγραμματιστές, χωρίς να χρειάζεται κάποιος επιπλέον προγραμματισμός, δίνει την δυνατότητα για καλύτερο έλεγχο και αποδεσμεύει πόρους.

2.6 Συνεχής Μάθηση και Πειραματισμός – Τρίτο Μονοπάτι

Το πρώτο μονοπάτι περιγράφει τις αρχές που επιτρέπουν τη γρήγορη ροή της εργασίας από αριστερά προς τα δεξιά, το δεύτερο μονοπάτι τις αρχές που επιτρέπουν την αμοιβαία και συνεχή ανατροφοδότηση από δεξιά προς τα αριστερά, και τέλος το τρίτο επικεντρώνεται στη δημιουργία κουλτούρας για συνεχή μάθηση και πειραματισμό.

Στόχο αποτελεί η ανάπτυξη κουλτούρας υψηλής εμπιστοσύνης η οποία μας δίνει τη δυνατότητα, με την δημιουργία γνώσης σε μεμονωμένους μηχανικούς, να τη διευρύνουμε και να την μοιράσουμε σε όλα τα μέλη της ομάδας και σε ολόκληρο τον οργανισμό. Για την ενίσχυση της δια βίου μάθησης χρειάζεται ανάληψη πρωτοβουλιών και ρίσκων, και με την εφαρμογή επιστημονικών προσεγγίσεων σε διαδικασίες και μεθόδους παραγωγής λογισμικού, η μάθηση γεννιέται μέσα από τις επιτυχημένες αλλά και αποτυχημένες εφαρμογές αυτών των προσεγγίσεων.

2.6.1 Υιοθέτηση Κουλτούρας για Συνεχή Μάθηση

Οι οργανισμοί μπορούν να κατηγοριοποιηθούν στις τρεις παρακάτω κατηγορίες σύμφωνα με την κουλτούρα τους σχετικά με την ασφάλεια και τις επιδόσεις:^[11]

- Παθολογικοί οργανισμοί: Χαρακτηρίζονται από φόβο και απειλές. Οι άνθρωποι κρατούν τις πληροφορίες για τους εαυτούς τους ή τις αλλοιώνουν για να φαίνονται πιο επιτυχημένοι. Κάθε πιθανή αποτυχία αποκρύπτεται.
- Γραφειοκρατικοί οργανισμοί: Χαρακτηρίζονται από διαδικασίες και αυστηρούς κανόνες που εξυπηρετούν μεμονωμένους χρήστες ή και ομάδες για να διατηρήσουν τα στεγανά της ομάδας. Η αποτυχία αντιμετωπίζεται μέσα από ένα σύστημα κρίσης το οποίο έχει ως αποτέλεσμα την τιμωρία ή την απόδοση δικαιοσύνης.
- Γενετικοί οργανισμοί: Χαρακτηρίζονται από την ενεργή αναζήτηση και ανταλλαγή πληροφοριών για την επίτευξη της αποστολής του οργανισμού. Οι ευθύνες μοιράζονται σε όλο το σύστημα αξιών ροής και οι αποτυχίες οδηγούν σε περαιτέρω προβληματισμό και έρευνα.

Για να έχουμε τα επιθυμητά αποτελέσματα, των αξιών ροής στην τεχνολογία αλλά και για συνεχή εξελισσόμενη μάθηση θα πρέπει να υιοθετηθεί η κουλτούρα ενός γενετικού οργανισμού μέσα σε ένα ασφαλές σύστημα εργασίας. Σε περίπτωση αποτυχίας ή λαθών είναι αναγκαία η άμεση αναζήτηση εναλλακτικών τρόπων (πιθανός επανασχεδιασμός) για την αποτροπή της ίδιας αστοχίας στο μέλλον. Με την αφαίρεση των κατηγοριών και την στοχοποίηση σε περίπτωση αποτυχίας που οφείλεται σε ανθρώπινο παράγοντα, επιτυγχάνεται η μείωση του φόβου και η ενίσχυση της ειλικρίνειας που επιτρέπουν την πρόληψη τέτοιων συμβάντων.

Με αυτό τον τρόπο και με την άμεση ανάλυση των έκτακτων, απρογραμμάτιστων περιστατικών συμβάλουμε στην ανάπτυξη της γνώσης και το μοίρασμα. Στους οργανισμούς ενισχύεται η αυτό-διάγνωση και η αυτό-βελτίωση καθώς και η ικανότητα για έγκαιρο εντοπισμό προβλημάτων και επίλυση.

2.6.2 Θεσμοθέτηση της Βελτίωσης της Καθημερινής Εργασίας

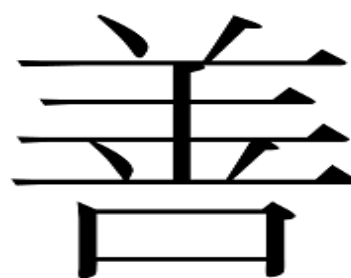
Οι ομάδες συχνά δεν θέλουν εισαγωγή αλλαγών στις διαδικασίες οι οποίες φαινομενικά λειτουργούν. Αυτό έχει ως αποτέλεσμα με το πέρασ του χρόνου να μην γίνονται βελτιώσεις και να υπάρχουν προβλήματα στην καθημερινότητα. Σε αυτά τα προβλήματα δίνονται προσωρινές λύσεις οι οποίες μακροπρόθεσμα εδραιώνουν την ύπαρξη τους και μειώνουν την παραγωγικότητα.

Για τη βελτίωση της καθημερινής εργασίας χρειάζεται δέσμευση χρόνου για να διορθωθούν πιθανά ελαττώματα καθώς και προβληματικές περιοχές. Μία από τις τεχνικές συνεχούς βελτίωσης που έχει εισαχθεί και εξελιχθεί, από το 1986 έως τις μέρες μας, είναι η τεχνική Kaizen με ετοιμολογία στην Ιαπωνική γλώσσα: Kai = Αλλαγή & Zen = Καλή (Εικόνα 2-4). Ένα από τα αξιοσημείωτα χαρακτηριστικά του Kaizen είναι ότι οι μικρές αλλαγές επιφέρουν μεγάλα αποτελέσματα με την ενεργή συμμετοχή όλων. Για την επίτευξη αυτών των αλλαγών είναι προτιμότερο να αναπτυχθούν ομάδες σε οριζόντιο επίπεδο.^[12]

Τα αποτελέσματα αυτών των πρακτικών με τη συμμετοχή όλων στον εντοπισμό και την επίλυση προβλημάτων, ως μέρος της καθημερινότητας, συμβάλλουν στην εξάλειψη των λιγότερο προφανή προβλημάτων. Με την ανίχνευση των μικρότερων αδυναμιών δεν επιδιορθώνουμε μόνο εύκολα και με λιγότερο κόστος προβλήματα αλλά ελαχιστοποιούμε και τις συνέπειες τους.



Kai=Αλλαγή



Zen=Καλή

Εικόνα 2-5: Ετοιμολογία Kaizen^[12]

2.6.3 Μετασχηματισμός Τοπικών Ευρημάτων σε Γενικότερες Βελτιώσεις

Όταν εντοπίζουμε κάτι σε επίπεδο μεμονωμένου μηχανικού ή ομάδας το οποίο μετατρέπεται σε γνώση, πρέπει να δοθεί το πλεονέκτημα σε ολόκληρο τον οργανισμό να έχει πρόσβαση σε αυτό. Αυτά τα ευρήματα είναι συνήθως που δημιουργούν εμπειρία και όπως είναι γνωστό η εμπειρία είναι αδύνατον να μεταδοθεί μέσα από γραπτό ή προφορικό λόγο. Αν, όμως, διαμοιραστεί η γνώση με ένα δομημένο και κωδικοποιημένο τρόπο, τότε μέσα από την πρακτική και την εξάσκηση μπορεί να δημιουργηθεί η εμπειρία.

Με τον παραπάνω τρόπο εξασφαλίζεται ότι οι εργασίες που έχουν ξαναγίνει στον οργανισμό βάσει της εμπειρίας άλλων συναδέλφων μπορούν να επαναχρησιμοποιηθούν. Για να είναι ορατές σε οριζόντιο επίπεδο θα πρέπει να είναι διαθέσιμες σε κάποιο αποθετήριο οι αναφορές και ο τρόπος επίλυσης των περιστατικών για τη γρήγορη επίλυση παρόμοιων προβλημάτων καθώς, επίσης, και να δημιουργηθούν κοινόχρηστα αποθετήρια πηγαίου κώδικα, βιβλιοθηκών και διαμορφώσεων συστημάτων προς επαναχρησιμοποίηση.

2.6.4 Εισαγωγή Μεθόδων Ανθεκτικότητας

Σε πολλούς οργανισμούς για να διασφαλιστεί η άρτια λειτουργία των συστημάτων γίνεται αύξηση των πόρων και των προσλήψεων εξειδικευμένου προσωπικού. Ως επακόλουθο είναι η αύξηση του κόστους και σε πολλές περιπτώσεις χωρίς την αντίστοιχη αύξηση της αποδοτικότητας και αποτελεσματικότητας των συστημάτων. Σε αντίθεση με τα παραπάνω μέτρα και έχοντας σημαντικά λιγότερο κόστος, η εισαγωγή μεθόδων ανθεκτικότητας των συστημάτων επιτυγχάνουν τα ίδια αποτελέσματα με την βελτίωση των λειτουργιών.

Μερικές από τις μεθόδους ελέγχου ανθεκτικότητας είναι η «Ημέρα Παιχνιδιού» (Game Day) όπου γίνονται προσομοιώσεις σε αστοχίες μεγάλης κλίμακας, όπως για παράδειγμα η απενεργοποίηση ολόκληρου κέντρου δεδομένων (data center). Σκοπό της μεθόδου αποτελεί η εκτέλεση των ενεργειών που θα ακολουθούσε η ομάδα σε αντίστοιχη περίπτωση ώστε να αφομοιωθεί ο τρόπος απόκρισης. Οι ημέρες παιχνιδιού θα πρέπει να καλύπτουν τομείς όπως των λειτουργιών, της ασφάλειας, της αξιοπιστίας, της απόδοσης και του κόστους.^[13]

Άλλη μία μέθοδος είναι αυτή της «Χαοτικής Μαϊμούς» (Chaos Monkey) που εφαρμόζει και η Netflix^[14] και είναι υπεύθυνη για τον τυχαίο τερματισμό διεργασιών σε παραγωγικό περιβάλλον ώστε να διασφαλιστεί ότι οι μηχανικοί εφαρμόζουν τις προβλεπόμενες ενέργειες και τα συστήματα είναι ανθεκτικά σε έκτακτα προβλήματα.

3 *Βέλτιστες Πρακτικές και Μεθοδολογία SAFe*

Στο τρίτο κεφάλαιο παρατίθενται οι βέλτιστες πρακτικές και η μεθοδολογία SAFe για τη μελέτη περίπτωσης. Στην αρχή του κεφαλαίου υπάρχει η ανάλυση SWOT με τα Δυνατά και Αδύνατα σημεία DevOps καθώς και τις Ευκαιρίες και τις Απειλές. Στην πορεία αναπτύσσονται θεμελιώδεις πρακτικές και κομμάτια των φάσεων το μοντέλου DevOps μέσα από τον αγωγό ανάπτυξης και το οικοσύστημα συνεχούς παράδοσης. Στο τέλος το κεφαλαίου βρίσκεται η ανάλυση της μεθοδολογίας SAFe DevOps πάνω στην οποία βασίστηκε και η μελέτη περίπτωσης του πέμπτου κεφαλαίου.

Ο σκοπός αυτού του κεφαλαίου είναι να παρουσιάσει σε βάθος τις πτυχές πάνω στις οποίες βασίζεται ο κύκλος των φάσεων παράδοσης λογισμικού DevOps καθώς και η θεωρητική επεξήγηση των εργαλείων της μεθοδολογίας SAFe που χρησιμοποιήθηκαν για την μελέτη περίπτωσης.

3.1 *Ανάλυση SWOT*

3.1.1 *Δυνατά Σημεία (Strengths)*

Σύμφωνα με τις ετήσιες έρευνες των εργαστηρίων της εταιρίας Puppet^[3] σχετικά με την πρόοδο των DevOps, και έπειτα από συλλογή δεδομένων περισσότερων από 25 χιλιάδων επαγγελματιών στον σχετιζόμενο χώρο, τα σημαντικότερα πλεονεκτήματα που έχουν προσθέσει αδιαμφισβήτητα μεγάλη αξία (added value) στις επιχειρήσεις λογισμικού είναι τα παρακάτω:

- Αυξημένη απόδοση:
 - Παράδοση κώδικα → 30 φορές πιο συχνά
 - Χρόνος παράδοσης → 200 φορές πιο γρήγορα
- Αυξημένη αξιοπιστία:
 - Επιτυχημένη παράδοση και ενσωμάτωση στην παραγωγή → εξήντα φορές υψηλότερο ποσοστό επιτυχίας των αλλαγών
 - Μέσος χρόνος για την επαναφορά της υπηρεσίας → 168 φορές πιο γρήγορα
- Αυξημένη απόδοση στον οργανισμό:
 - Στόχοι παραγωγικότητας, μεριδίου αγοράς και κερδοφορίας → δύο φορές πιο πιθανό να ξεπεραστούν
 - Αύξηση κεφαλαιοποίησης αγοράς → 50% υψηλότερα σε διάστημα τριών ετών

3.1.2 Αδύνατα Σημεία (Weaknesses)

Παρόλο που το DevOps έχει εδραιωθεί τα τελευταία χρόνια στον τομέα της πληροφορικής υπάρχουν ακόμα πολλές αδυναμίες τις οποίες καλούνται να επιλύσουν οι διευθυντές και οι μηχανικοί. Χρειάζεται δουλειά προς όλες τις ιεραρχικές κατευθύνσεις, από πάνω προς τα κάτω και το αντίστροφο. Μερικές από τις αδυναμίες που έχουν εντοπιστεί είναι οι εξής^[15]:

- Δυσλειτουργική κουλτούρα: Η κουλτούρα σε έναν οργανισμό είναι δύσκολο να αλλάξει καθώς απαιτεί να αντιμετωπιστούν αξίες και συμπεριφορές ανθρώπων που σχετίζονται με τη συναισθηματική νοημοσύνη. Έτσι, λοιπόν, εταιρείες παθολογικές ή γραφειοκρατικές θα πρέπει να μετασχηματιστούν σε γενετικές και να δώσουν ώθηση στην κατανομή ευθυνών αλλά και γνώσης.
- Αντίσταση στην αλλαγή: Πολλοί άνθρωποι δεν θέλουν να αλλάξουν τη ρουτίνα και τις διαδικασίες που ακολουθούν με αποτέλεσμα να υπάρχουν αντιστάσεις στον μετασχηματισμό.
- Έλλειψη σαφήνειας και οράματος: Σε πολλές περιπτώσεις η έννοια του DevOps παρερμηνεύεται και έχει ως αποτέλεσμα την απουσία ηγετικής υποστήριξης σε αυτούς που καλλιεργούν την DevOps κουλτούρα. Αν και υπάρχει επιθυμία για μετασχηματισμό, ο χρόνος για σχεδιασμό του τρόπου υιοθέτησης είναι ανεπαρκής.
- Οι ομάδες δεν συνεργάζονται: Ένας από τους κύριους στόχους του DevOps είναι η συνεργασία των ομάδων και η κατάργηση πιθανών εμποδίων. Επιπλέον δυσκολίες παρουσιάζονται στη συνεργασία όταν οι ομάδες βρίσκονται σε απομακρυσμένες γεωγραφικές περιοχές με διαφορετικές ζώνες ωρών.
- Μη τυποποιημένα περιβάλλοντα εργασίας: Όταν δεν υπάρχουν διαθέσιμα περιβάλλοντα με συγκεκριμένα στάνταρτ και πρέπει να αντιμετωπιστούν πολλαπλές εφαρμογές με διαφορετική έκδοση δημιουργούνται καθυστερήσεις στην τελική παράδοση του έργου προς τους πελάτες.
- Μη κατάλληλα εργαλεία: Η επιλογή των εργαλείων είναι πολύ σημαντικό κομμάτι καθώς θα πρέπει να εφαρμόζεται και να ευθυγραμμίζεται με τους στόχους όλων των επιμέρους συμμετεχόντων ομάδων καθώς και του οργανισμού συνολικά.
- Καθυστερήσεις στις παραδόσεις: Σε περιπτώσεις που ενσωματώνονται μεγάλα τμήματα και δεν γίνονται πολύ συχνές παραδόσεις, υπάρχει το ενδεχόμενο να υπάρξουν καθυστερήσεις από πιθανά προβλήματα που έχουν συσσωρευτεί και χρήζουν λύσης.
- Η μη αυτοματοποιημένη δοκιμή είναι χρονοβόρα: Η χειροκίνητη δοκιμή είναι πολύ χρονοβόρα και εκτελείται συχνά από ξεχωριστές ομάδες σε διαφορετικές φάσεις δημιουργώντας καθυστερήσεις

3.1.3 Ευκαιρίες (Opportunities)

- Κουλτούρα DevOps: Το DevOps μας επιτρέπει να βελτιώσουμε ταυτόχρονα την οργανωτική απόδοση, να επιτύχουμε τους στόχους όλων των επιμέρους λειτουργικών τεχνολογικών ρόλων (π.χ. Development, QA, IT Operations, Infosec) και να βελτιώσουμε την ανθρώπινη κατάσταση.
- Πλήρωση κενού μεταξύ του τμήματος ανάπτυξης λογισμικού και του τμήματος λειτουργιών: Αδιαμφισβήτητα, εισάγοντας την ομάδα DevOps στον οργανισμό, η

ενσωμάτωση και παράδοση μπορεί να λάβει μέρος κατά την διάρκεια της ημέρας, δίνοντας τη δυνατότητα σε όλες τις υπόλοιπες ομάδες να δουλεύουν σε εργάσιμες ώρες και μέρες, κάτι που στο παρελθόν μπορούσε να γίνει μόνο εκτός ωραρίου ώστε να υπάρχει αδιάλειπτη λειτουργία των συστημάτων.

- Διαφάνεια και προώθηση της συνεργασίας και του διαμοιρασμού της γνώσης σε όλο τον οργανισμό.
- Βελτιωμένη ασφάλεια και αυτοματοποίηση^[16].
- Υψηλότεροι ρυθμοί πρόσληψης και διατήρηση εργαζομένων.

3.1.4 Απειλές (Threats)

- Υποστήριξη: Το βάρος υποστήριξης μπορεί να αυξηθεί σημαντικά, ειδικά όταν οι λειτουργίες πληροφορικής αντιμετωπίζουν ξαφνικά τη διαχείριση εκατοντάδων μικροϋπηρεσιών που έχουν αναπτυχθεί σε διαφορετικές γλώσσες, την πρόσβαση σε νέες μονάδες αποθήκευσης δεδομένων και την εκτέλεση σε πλατφόρμες και υποδομές νέφους (cloud).
- Παρακολούθηση: Η διαχείριση μιας μονολιθικής εφαρμογής είναι απαιτητική, αλλά τώρα οι λειτουργίες της ΤΠ πρέπει να διασφαλίσουν ότι πολλές ακόμη διαδικασίες παραμένουν αποτελεσματικές. Με πολύ κατανεμημένα συστήματα μικροϋπηρεσιών, εμφανίζεται ένα ολοκαίνουργιο σύνολο προβληματισμών διαχείρισης. Αυτά περιλαμβάνουν καθυστέρηση δικτύου, ανοχή σφαλμάτων, προβλήματα ασύγχρονης ανταλλαγής μηνυμάτων και αξιοπιστία δικτύου.
- Συντονισμός: Η ανάπτυξη εκατοντάδων μικροϋπηρεσιών απαιτεί αυστηρές διαδικασίες ανάπτυξης που μπορεί να ξεπερνούν τις δυνατότητες των ομάδων που χρησιμοποιούν χειροκίνητες διαδικασίες και δέσμες ενεργειών.
- Αντικρουόμενες προτεραιότητες: Σε κάθε Οργανισμό πληροφορικής υπάρχουν συγκρούσεις μεταξύ του Τμήματος ανάπτυξης λογισμικού και του Τμήματος λειτουργιών που έχουν ως αποτέλεσμα την επιπλέον καθυστέρηση του χρόνου παράδοσης των νέων προϊόντων καθώς και των αναβαθμίσεών τους, ελαττωμένη ποιότητα και αυξημένες παύσεις στη λειτουργία των υποδομών.
- Μη ολοκληρωμένες διαδικασίες: Έχουν ως αποτέλεσμα τη μετακύλιση των εκκρεμοτήτων σε επόμενα στάδια και την ανάγκη για επαναληπτικές εργασίες που είναι ημιτελής ή μπλοκάρουν την ροή.
- Σιλό: Διάφοροι μηχανικοί, συνήθως οι πιο έμπειροι, γίνονται αποδέκτες των περισσότερων σύνθετων προβλημάτων διότι δεν υπάρχει διαμοιρασμός της γνώσης ή επειδή σε πολλές περιπτώσεις δεν επαρκούν οι ανθρώπινοι πόροι.
- Περιορισμένη Ανατροφοδότηση: Σε πολλούς οργανισμούς οι αρχές της ανατροφοδότησης είναι από ελάχιστες έως ανύπαρκτες.
- Δημιουργία υποκοουλτούρας: Η έλλειψη γνώσης και αντίστασης στις αλλαγές επιφέρουν μια υποκοουλτούρα που στέκεται εμπόδιο στην υιοθέτηση βέλτιστων πρακτικών.
- Πολλαπλά εργαλεία και περιβάλλοντα: Σε πολλές περιπτώσεις και λόγω των ειδικών συνθηκών, ιδιαίτερα σε έργα με ανάδοχες εταιρείες, χρησιμοποιούνται πολλαπλά εργαλεία και από τις δυο πλευρές (πελάτες και Οργανισμούς) και σε πολλές περιπτώσεις υπάρχει

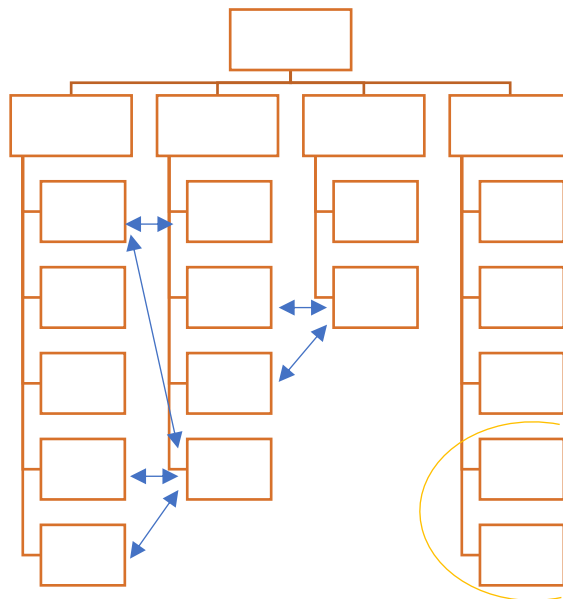
ασυμβατότητα. Επίσης, το κόστος πολλαπλών εργαλείων και ειδικά των περιβαλλόντων μπορεί να είναι αρκετά υψηλό.

3.2 Θεμελιώδεις Πρακτικές DevOps

3.2.1 Επικοινωνία στην Παραδοσιακή Οργανωτική Δομή

Οι περισσότεροι οργανισμοί έχουν απεικονίσει την οργανωτική δομή μέσα από ιεραρχικά οργανογράμματα όπως το παρακάτω σχήμα 4-1. Αυτά τα οργανογράμματα απεικονίζουν τις ομάδες, τα Τμήματα και άλλες οντότητες τους, καθώς και τη συσχέτιση μεταξύ τους. Συνήθως υπάρχει ένωση με βάση τη γραμμή της ιεραρχικής αναφοράς από πάνω μέχρι κάτω η οποία είναι και η βασική γραμμή επικοινωνίας.

Παρ' όλα αυτά υπάρχει η ανάγκη οι ξεχωριστές οντότητες/ομάδες να επικοινωνήσουν μεταξύ τους για την επίτευξη των στόχων, όπως απεικονίζεται στο Σχήμα 3-1 με τα μπλε βελάκια. Αυτό δημιουργεί πρόβλημα διότι δεν ακολουθείται η ιεραρχική γραμμή επικοινωνίας. Επίσης, υπάρχουν οντότητες οι οποίες λειτουργούν απομονωμένες και δεν υπάρχει διασύνδεση με τις υπόλοιπες ομάδες πέραν της ιεραρχικής γραμμής επικοινωνίας.^[4]



Σχήμα 3-1: Οργανόγραμμα

3.2.2 Διαχείριση Διαμόρφωσης (Configuration Management)

Η διαχείριση διαμόρφωσης είναι συνδεδεμένη με τον έλεγχο έκδοσης (version control) αλλά στην ουσία είναι η διαδικασία με την οποία όλα τα στοιχεία που σχετίζονται με το έργο, και οι μεταξύ τους εξαρτήσεις, αποθηκεύονται, ανακτώνται, αναγνωρίζονται και τροποποιούνται.^[17] Για να διασφαλίσουμε ορατότητα θα πρέπει να έχουμε έλεγχο έκδοσης στο αποθετήριο:^[1]

- Τον κώδικα της εφαρμογής και τις εξαρτήσεις (βιβλιοθήκες, στατικό περιεχόμενο).
- Τις εντολές για τη δημιουργία σχημάτων βάσεων δεδομένων και δεδομένων αναφορών εφαρμογής.

- Τα εργαλεία, τον κώδικα και τη διαμόρφωση για τη δημιουργία των περιβαλλόντων εργασίας.
- Αρχεία που χρησιμοποιούν τα κοντέινερ.
- Όλες τις αυτοματοποιημένες δοκιμές και τις εντολές για τα χειροκίνητα σενάρια δοκιμών.

Μερικά από τα πιο διαδεδομένα εργαλεία με έλεγχο έκδοσης είναι το GitLab, GitHub, Nexus, τα οποία παρέχουν διάφορες λειτουργίες για την παράδοση του κώδικα αλλά σε πολλές περιπτώσεις μπορούν να λειτουργήσουν συμπληρωματικά το ένα με το άλλο.

3.2.3 Συνεχής Ενσωμάτωση (Continuous Integration)

Η συνεχής ενσωμάτωση είναι η πρακτική για τη συγχώνευση αντιγράφων από παραδοτέα κομμάτια κώδικα προγραμματιστών σε ένα κεντρικό σημείο, πολλαπλές φορές κατά τη διάρκεια της ημέρας. Για να επιτύχουμε τη συνεχή ενσωμάτωση, εκτός από εργαλεία χρειαζόμαστε και την επικοινωνία DevOps με τους προγραμματιστές. Αν δεν υιοθετηθεί πειθαρχημένα η επικοινωνία και δεν επιδιορθώνονται εγκαίρως οι αλλαγές που προκαλούν προβλήματα στην εφαρμογή, δεν πρόκειται να έχουμε συνεχή ενσωμάτωση.

Επιπλέον, είναι αναγκαία η χρήση του ελέγχου εκδόσεων και ενός μηχανισμού αυτόματου χτισίματος. Η συνεχής ενσωμάτωση δημιουργεί ένα βρόχο ανατροφοδότησης και επιτρέπει τον εντοπισμό προβλημάτων και άμεση επίλυση τους. Ένα από τα εργαλεία αυτοματοποίησης της συνεχούς ενσωμάτωσης είναι το Jenkins^[18]. Υποστηρίζει τα περισσότερα εργαλεία ελέγχου έκδοσης.

3.2.4 Εφαρμογή Στρατηγικής Δοκιμών

Αναπόσπαστο κομμάτι της συνεχούς ενσωμάτωσης είναι και οι δοκιμές. Στην ευέλικτη μεθοδολογία καλύπτονται τα παρακάτω επίπεδα δοκιμών:^[19]

- Δοκιμή μονάδας (Unit Testing)
- Δοκιμή ολοκλήρωσης (Integration Testing)
- Δοκιμή συστήματος (System Testing)
- Δοκιμή αποδοχής χρήστη (User Acceptance Testing)

Οι τύποι των δοκιμών είναι:

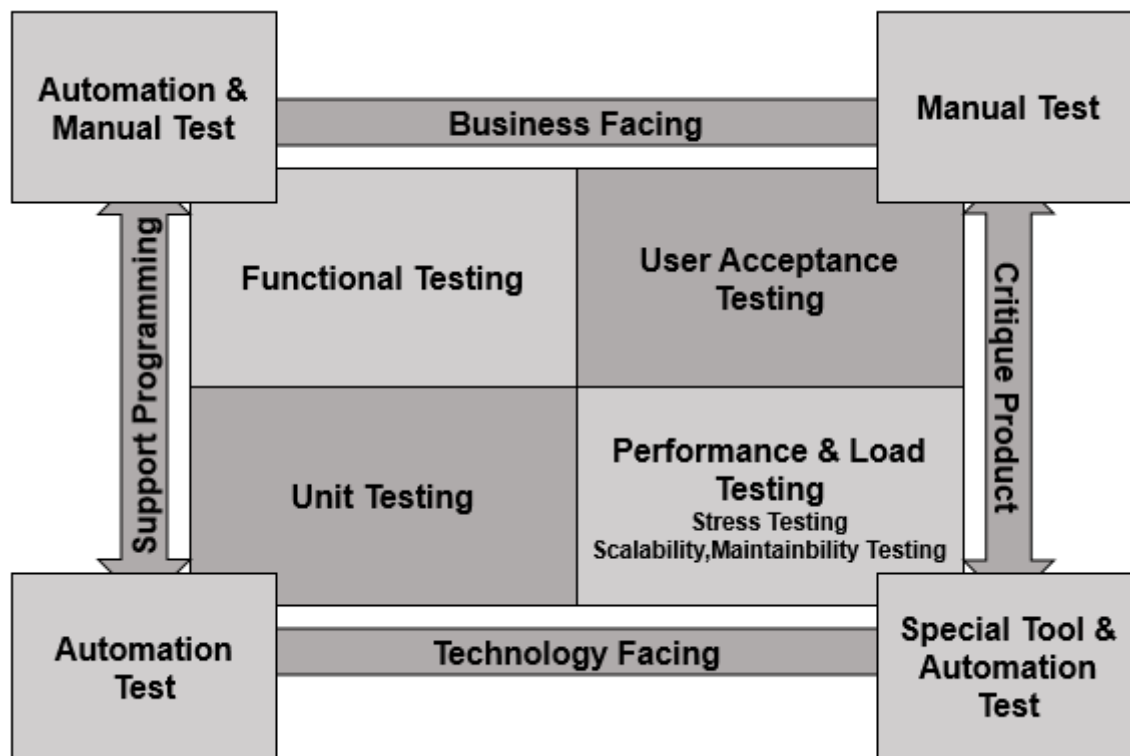
- Δοκιμές εξαρτημάτων (For unit test)
- Λειτουργικές δοκιμές (For user stories)
- Μη λειτουργικές δοκιμές (For performance, stress, load)
- Δοκιμές αποδοχής

Αυτές οι δοκιμές μπορεί να είναι για:

- Υποστήριξη του προγραμματισμού (Support Programming), ώστε να αποφασιστεί τι κώδικας πρέπει να γραφτεί για να επιτευχθούν συγκεκριμένα αποτελέσματα αλλά και για να μην υπάρχουν προβλήματα με τις υπόλοιπες λειτουργίες του συστήματος.
- Κριτική προϊόντος (Critique Product), που δεν είναι άλλο από την επαλήθευση για ανεπάρκειες στο τελικό προϊόν.

Για να καταλήξουμε ποια δοκιμή πρέπει να γίνει και πότε, πρέπει να λάβουμε υπόψιν: α) αν είναι από την τεχνολογική πλευρά (technology facing) και αφορά δοκιμές που απαντούν σε ερωτήματα

από τον τομέα της τεχνολογίας ή β) αν είναι από την επιχειρησιακή πλευρά (business facing) που απαντούν αντίστοιχα ερωτήματα επιχειρηματικής φύσεως. Έτσι, συνοψίζοντας, μπορούμε να δούμε σε κάθε τεταρτημόριο (Εικόνα 3-1) το επίπεδο δοκιμής, καθώς και το αν χρειάζεται αυτοματοποίηση ή κάποιο ειδικό εργαλείο.



Εικόνα 3-1: Τεταρτημόρια Δοκιμών^[19]

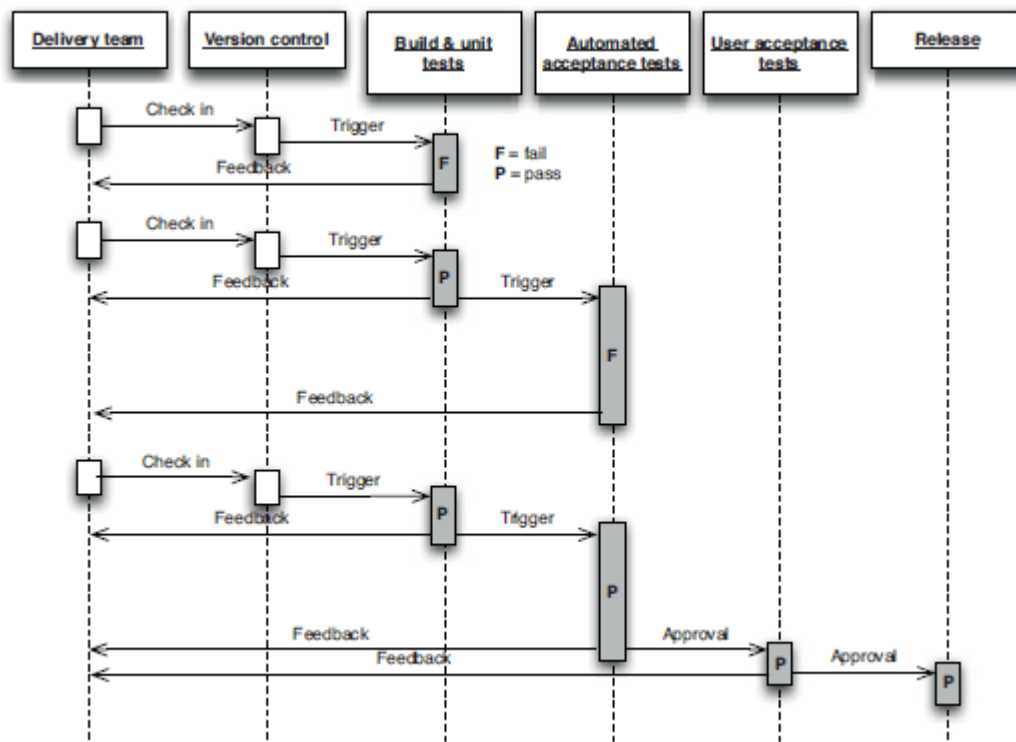
3.3 Διοχετεύσεις Ανάπτυξης (Deployment Pipelines)

3.3.1 Ανατομία των Διοχετεύσεων/Αγωγών Ανάπτυξης

Σε ένα αφηρημένο επίπεδο ένας αγωγός ή διοχέτευση ανάπτυξης (delivery pipeline) είναι μια αυτοματοποιημένη διαδικασία για την παράδοση του λογισμικού από τον έλεγχο έκδοσης στους τελικούς χρήστες. Για να φτάσουμε στην τελική παράδοση, υπάρχει μια πολύπλοκη διαδικασία δοκιμών και σταδιακής ανάπτυξης και απαιτείται η συνεργασία αρκετών ατόμων ή και ομάδων. Με τον αγωγό ανάπτυξης γίνεται μια μοντελοποίηση με εργαλεία συνεχούς ενσωμάτωσης και διαχείρισης των εκδόσεων προς παράδοση, που επιτρέπει τον έλεγχο της προόδου της μεταβολής σε κάθε στάδιο.^[17]

Οι αλλαγές κατά τον αγωγό ανάπτυξης φαίνονται στην εικόνα 3-2. Το έναυσμα για να ξεκινήσει η διαδικασία δίνεται από τη στιγμή που γίνεται το check-in στον έλεγχο έκδοσης. Στη συνέχεια προχωράει διαδοχικά στη δόμηση (build) και στη δοκιμή μονάδας (unit test). Εάν η διαδικασία είναι επιτυχημένη προχωράει στις αυτοματοποιημένες δοκιμές αποδοχής ειδικά

στέλνει ενημέρωση για την αποτυχία. Έπειτα προχωράει μετά από έγκριση στη δοκιμή αποδοχής χρήστη και τελικά πηγαίνει προς την τελική έκδοση.



Εικόνα 3-2: Αλλαγές κατά την διοχέτευση ανάπτυξης^[17]

3.3.2 Δόμηση και Δέσμη Ενεργειών Ανάπτυξης

Πολλά συστήματα δόμησης χρησιμοποιούν τον πηγαίο κώδικα που βρίσκεται στο σύστημα ελέγχου έκδοσης ως πηγή για πολλά βήματα της διαδικασίας παράδοσης λογισμικού. Ο κώδικας θα μεταγλωττιστεί σε πολλά στάδια όπως στη δέσμευση, αργότερα στην αποδοχή δοκιμής και σε κάθε επικείμενη ανάπτυξη. Ο κίνδυνος να εισάγουμε διαφορετική έκδοση μεταγλωττιστή σε σχέση με αυτό που χρησιμοποιήθηκε στις δοκιμές, ακόμα και η διαμόρφωση των ρυθμίσεων του μεταγλωττιστή μπορεί να έχουν ως αποτέλεσμα την αλλαγή συμπεριφοράς, πιθανώς και τη διακοπή της λειτουργίας της εφαρμογής.

Αξιοσημείωτο είναι ότι κάθε εργασία έχει δυο βασικά χαρακτηριστικά, τη λειτουργία και την εξάρτηση από άλλες λειτουργίες. Αυτά τα δυο χαρακτηριστικά μοντελοποιούνται σε κάθε εργαλείο δόμησης. Τα εργαλεία δόμησης έχουν μία διαφορά ως προς τον προσανατολισμό τους. Σε κάποια από τα διαθέσιμα εργαλεία (Ant, NAnt, MsBuild) προσανατολισμός είναι η εργασία και περιγράφουν τις εξαρτήσεις σε ένα σύνολο εργασιών. Στα εργαλεία (Make) που ο προσανατολισμός τους είναι το προϊόν περιγράφονται τα πράγματα σε μορφή εκτελέσιμων αρχείων.

3.3.3 Στάδιο Δέσμευσης

Το στάδιο της δέσμευσης (commit) ξεκινά από μία αλλαγή στο σύστημα ελέγχου και ολοκληρώνεται από τη δημιουργία δυαδικών αντικειμένων (binary artifacts) και συναρμολογήσεων

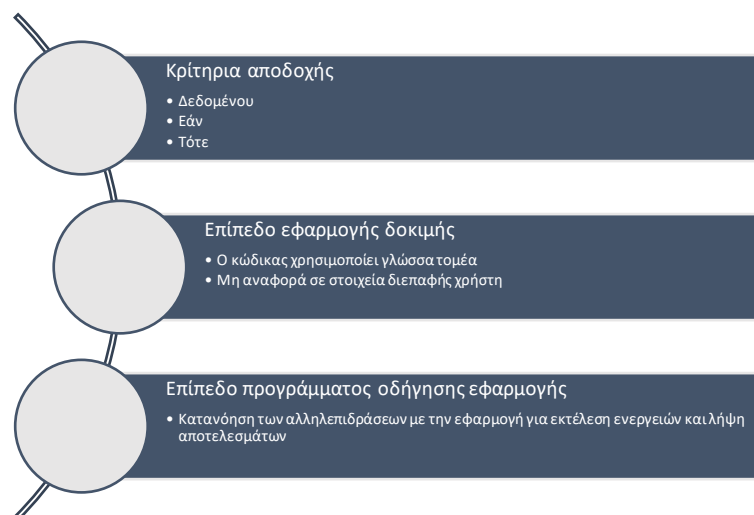
ανάπτυξης (deployable assemblies) ή σε περίπτωση αποτυχίας από μία αναφορά. Σε αυτό το στάδιο έχουμε την αρχική εισαγωγή στον αγωγό ανάπτυξης και δεν είναι μόνο η εισαγωγή των υποψηφίων εκδόσεων προς παράδοση αλλά και το σημείο αναφοράς για τις ομάδες για την έναρξη της διοχέτευσης ανάπτυξης.

Οι εργασίες που γίνονται κατά το στάδιο της δέσμευσης, και ξεκινούν από την εισαγωγή μιας αλλαγής (check in) στο σύστημα ελέγχου, είναι οι εξής:

- Ο διακομιστής συνεχούς ενσωμάτωσης εντοπίζει την αλλαγή και έπειτα από έλεγχο στον πηγαίο κώδικα προχωράει σε μεταγλώττιση, εάν είναι απαραίτητο, και σε εκτέλεση δοκιμών δέσμευσης σε αντιπαράθεση με τον ενσωματωμένο πηγαίο κώδικα.
- Δημιουργία δυαδικών αρχείων για την ανάπτυξη σε κάθε περιβάλλον, συμπεριλαμβάνοντας τη σύνταξη και συναρμολόγηση στις περιπτώσεις χρήσης μεταγλωττισμένης γλώσσας.
- Έλεγχος και ανάλυση της υγείας της βάσης του κώδικα.
- Δημιουργία αντικειμένων προς χρήση στον αγωγό ανάπτυξης (για παράδειγμα δεδομένα δοκιμών ή μετεγκαταστάσεις βάσεων δεδομένων).

3.3.4 Αυτοματοποιημένη Αποδοχή Δοκιμών

Για την αποδοχή των δοκιμών υπάρχουν τρία επίπεδα όπως φαίνονται στο σχήμα 3-2:



Σχήμα 3-2: Επίπεδα στην αποδοχή δοκιμών^[17]

Οι δοκιμές αποδοχής επαληθεύουν ότι η εφαρμογή παρέχει αξία στους χρήστες. Οι αναλυτές ορίζουν τα κριτήρια αποδοχής σε ιστορίες έτσι ώστε όταν πληρούνται όλα τα κριτήρια να είναι ολοκληρωμένη η ιστορία. Η σύνταξη των κριτηρίων αποδοχής είναι της μορφής:

- **Δεδομένο** ότι υπάρχει κάτι,
- **Εάν** συμβεί αυτό το γεγονός,
- **Τότε** έχω αυτά τα αποτελέσματα

Εργαλεία όπως το Cucumber, το JBehave, το Concordion, το Twist και το FitNesse επιτρέπουν να γράφουμε κριτήρια αποδοχής ως απλό κείμενο και να τα διατηρούμε συγχρονισμένα με την πραγματική εφαρμογή. Σε περίπτωση που η εφαρμογή είναι Web based μπορεί να καλέσουν εργαλεία όπως Selenium, Sahi ή WebDriver.

Εάν το τμήμα παράδοσης λογισμικού δεν διαθέτει αξιόπιστο αυτοματισμό δοκιμών, τότε πρέπει να υπάρχει σαφής αποσύνδεση μεταξύ των προσπαθειών ανάπτυξης και δοκιμών. Οι μηχανικοί ανάπτυξης λογισμικού λαμβάνουν σχετικά αργά τα σχόλια σχετικά με την ποιότητα της εργασίας τους, ενώ είναι ήδη απασχολημένοι με άλλες εργασίες και συνήθως χάνεται η σχέση αιτίας του αποτελέσματος μεταξύ του κώδικα που έχουν γράψει (ή δεν έχουν γράψει) και των ελαττωμάτων. Οι προσπάθειες που καταβάλλουν οι προγραμματιστές μειώνουν σαφώς την παραγωγικότητα επειδή η σύνδεση μεταξύ συστημάτων και ελαττωμάτων είναι περίπλοκη και ασαφής μετά από καθυστερημένη λήψη σχολίων. Η λήψη ποιοτικών διορθώσεων χωρίς εναλλακτική δεν είναι ούτε ρεαλιστική ούτε αναμενόμενη. Είναι δεδομένο ότι χωρίς αυτοματοποιημένο έλεγχο, ο οργανισμός χρειάζεται περισσότερο χρόνο και υψηλότερο τίμημα για να προσφέρει ποιοτικά προϊόντα και υπηρεσίες.

Η ομάδα DevOps πρέπει να αυτοματοποιήσει όσο το δυνατόν περισσότερες δοκιμαστικές περιπτώσεις. Μπορούμε να αυτοματοποιήσουμε τις δοκιμές, αλλά δεν μπορούμε να αυτοματοποιήσουμε τη γνωστική ικανότητα της ομάδας DevOps για να δημιουργήσει ποιότητα. Δεν θα πρέπει να σπαταληθεί ανθρώπινο δυναμικό για οποιαδήποτε επαναλαμβανόμενη εργασία που μπορεί να κάνει το λογισμικό.

Αρχές DevOps για καλό αυτοματισμό δοκιμών:

- Οι αυτοματοποιημένες δοκιμές θα πρέπει να δίνουν γρήγορα και έγκαιρα σχόλια σχετικά με την ποιότητα της εργασίας.
- Οι δοκιμές πρέπει να παράγουν συνεπή, ντετερμινιστικά και επαναλαμβανόμενα αποτελέσματα, υπό την προϋπόθεση ότι ισχύουν συνθήκες για διαφορετικές δοκιμές.
- Οι δοκιμές δεν πρέπει να δημιουργούν ψευδώς θετικά.
- Μικρός αριθμός αυτοματοποιημένων και αξιόπιστων περιπτώσεων δοκιμής είναι καλύτερος από έναν μεγάλο αριθμό μη αυτοματοποιημένων, αναξιόπιστων δοκιμών που μπορούν να δώσουν ασυνεπή αποτελέσματα, παρόλο που παρέχονται οι ίδιες προϋποθέσεις για διαφορετικές δοκιμές.
- Οι αυτοματοποιημένες δοκιμές θα πρέπει πρώτα να επικεντρωθούν στην επικύρωση βασικών χαρακτηριστικών και πλεονεκτημάτων που παρέχουν τα συστήματά στους εσωτερικούς και εξωτερικούς πελάτες. Ακόμα και αυτό το επίπεδο αυτοματοποιημένου ελέγχου είναι πολύ καλύτερο σε σχέση με την απουσία αυτοματοποιημένου ελέγχου.
- Αυτοματοποίηση όσων περισσότερων δοκιμαστικών περιπτώσεων είναι δυνατόν για να αξιοποιηθεί στο έπακρο το ανθρώπινο δυναμικό στις ομάδες DevOps.
- Με τις αυτοματοποιημένες δοκιμές, αποφεύγουμε αργά και περιοδικά σχόλια με γρήγορη ανατροφοδότηση κάθε φορά που ο προγραμματιστής προσπαθεί να κάνει εισαγωγή κώδικα στον αγωγό.
- Επιτάχυνση δοκιμών για να έχουμε μια αξιόπιστη πλατφόρμα συνεχούς παράδοσης και αγωγό ανάπτυξης, αλλά και μια εξαιρετική ποιότητα εξυπηρέτησης για τους πελάτες.
- Ανάπτυξη βάσει δοκιμών, αρχικά με την δημιουργία από τις αυτοματοποιημένες δοκιμές και μετά το λογισμικό.

3.3.5 Ανάπτυξη και Παράδοση Εφαρμογής

Για την ανάπτυξη και παράδοση μιας εφαρμογής υπάρχουν διαφορές μεταξύ περιβαλλόντων παραγωγής και περιβαλλόντων δοκιμών και σε τεχνικό επίπεδο αυτές οι διαφορές βρίσκονται στα αρχεία διαμόρφωσης. Για να επιτευχθεί ομαλή παράδοση χωρίς οι διαφορές να δημιουργούν εμπόδια, είναι αναγκαία η εφαρμογή στρατηγικής παράδοσης.

Η ομάδα DevOps πρέπει να έχει ενσωματωμένους μηχανισμούς ελέγχου και μετριασμού των κινδύνων με αλλαγή της εξάρτησης από μηχανισμούς ελέγχου και μετριασμού των κινδύνων από άλλες ανεξάρτητες ομάδες σε μια αυτάρκη και ικανή ομάδα. Δεδομένου ότι όλα τα προϊόντα παραγωγής και τα περιβάλλοντα είναι στοιχεία που είναι εξίσου σημαντικά με το λογισμικό που παράγει η ομάδα DevOps, η ομάδα DevOps ακολουθεί αυτές τις τρεις σημαντικές αρχές:

- Συνοχή όλων των περιβαλλόντων όσον αφορά τα λειτουργικά συστήματα, τα στοιχεία, τις διεπαφές, τα επίπεδα ενημέρωσης κώδικα, όλες τις άλλες εξαρτήσεις και φυσικά το λογισμικό και διαμορφώσεις.
- Προτεραιότητα στη δραστηριότητα επιδιόρθωσης πιθανού εμποδίου που σπάει τη συνέπεια των περιβαλλόντων.
- Εφαρμογή ίδιων τεχνικών σε όλα τα περιβάλλοντα, έτσι η εξομοίωση των παραγωγικών περιβαλλόντων γίνεται κομμάτι από τις καθημερινές εργασίες και δραστηριότητες.

Ο αυτοματοποιημένος μηχανισμός ανάπτυξης ακολουθεί μοτίβο με δημιουργία κώδικα και εισαγωγή στον κορμό, κατασκευή από αναπτυσσόμενα πακέτα με επικύρωση των πακέτων αλλά και του περιβάλλοντος, εκτέλεση αυτοματοποιημένων δοκιμών, καταγραφή από όλα τα αποτελέσματα επικύρωσης και αυτοματοποιημένων δοκιμών για λόγους ελέγχου και συμμόρφωσης, ανάπτυξη πακέτων στο περιβάλλον προορισμού, εκτέλεση αυτοματοποιημένων δοκιμών στο περιβάλλον προορισμού, παρακολούθηση μετρήσεων συστήματος και απόδοσης και τέλος παροχή σχολίων με γρήγορο τρόπο ώστε να διορθωθούν ή για την επαναφορά της ανάπτυξης εάν κάτι πάει στραβά.

Με έναν αυτοματοποιημένο μηχανισμό για την ανάπτυξη του λογισμικού σε παραγωγικά και μη παραγωγικά περιβάλλοντα, η ομάδα DevOps έχει τη δυνατότητα να κάνει καθημερινές παραδόσεις στα συστήματά του. Επειδή κάθε μεμονωμένη ανάπτυξη δεν μπορεί να κατηγοριοποιηθεί, η ομάδα DevOps πρέπει να δημιουργήσει τις εφαρμογές και τα περιβάλλοντα με τέτοιο τρόπο ώστε οι εκδόσεις να μην απαιτούν αλλαγές κώδικα και άλλες σχετικές αναπτύξεις. Μία από τις προκλήσεις με την αυτοματοποίηση της ανάπτυξης είναι η παράδοση λογισμικού από το τελικό στάδιο των δοκιμών σε παραγωγή. Συνήθως πρέπει να γίνει γρήγορα για να ελαχιστοποιηθεί ο χρόνος διακοπής λειτουργίας. Η προσέγγιση ανάπτυξης **Blue-Green** τη διασφαλίζει με δύο περιβάλλοντα παραγωγής, όσο το δυνατόν πανομοιότυπα. Ανά πάσα στιγμή, ένα από αυτά, ας πούμε μπλε για παράδειγμα, είναι διαθέσιμο στους χρήστες. Καθώς μια νέα έκδοση του λογισμικού είναι διαθέσιμη, το τελικό στάδιο δοκιμών γίνεται στο πράσινο περιβάλλον. Μόλις το λογισμικό λειτουργήσει στο πράσινο περιβάλλον, αλλάζει η δρομολόγηση έτσι ώστε όλα τα εισερχόμενα αιτήματα να πηγαίνουν στο πράσινο περιβάλλον και το μπλε παραμένει πλέον αδρανές.

Άλλο ένα πρότυπο ανάπτυξης είναι το **Canary** γνωστό και ως **Dark Launch**. Τα καναρίνια κάποτε χρησιμοποιούνταν τακτικά στην εξόρυξη άνθρακα ως σύστημα έγκαιρης προειδοποίησης.

Τοξικά αέρια όπως το μεθάνιο ή το διοξείδιο του άνθρακα στο ορυχείο σκότωναν τα πουλιά πριν επηρεάσουν τους ανθρακωρύχους. Σημάδια κινδύνου από το πουλί έδειχναν στους ανθρακωρύχους ότι οι συνθήκες δεν ήταν ασφαλείς. Εμπνευσμένο από τα καναρίνια στη μεταλλευτική βιομηχανία, η ανάπτυξη Canary αποτελεί ένα μοτίβο για την κυκλοφορία σε ένα υποσύνολο χρηστών ή διακομιστών. Η ιδέα είναι να αναπτυχθεί πρώτα η αλλαγή σε ένα μικρό υποσύνολο διακομιστών, να δοκιμαστεί, και στη συνέχεια να μεταφερθεί στους υπόλοιπους διακομιστές. Η ανάπτυξη Canary χρησιμεύει ως δείκτης έγκαιρης προειδοποίησης με ελάχιστο αντίκτυπο στο χρόνο διακοπής λειτουργίας. Εάν η ανάπτυξη Canary αποτύχει, οι υπόλοιποι διακομιστές δεν επηρεάζονται. Το Facebook και η **Google**, μαζί με πολλούς κορυφαίους τεχνολογικούς γίγαντες, χρησιμοποιούν ένα πρότυπο ανάπτυξης Canary που ονομάζεται "The Dark Launch". Απελευθερώνουν σταδιακά και δοκιμάζουν νέες δυνατότητες σε ένα μικρό σύνολο των χρηστών πριν τις κυκλοφορήσουν σε όλους. Αυτό επιτρέπει να βλέπουν αν αρέσει και να εκτιμούν πώς επηρεάζει την απόδοση του συστήματός. Το **Facebook** το ονομάζει "**Gatekeeper**" επειδή ελέγχει την πρόσβαση των καταναλωτών σε κάθε νέα δυνατότητα. Ονομάζεται σκοτεινή κυκλοφορία (Dark Launch), επειδή αυτές οι εκδόσεις λειτουργιών συνήθως δεν δημοσιεύονται, αλλά κυκλοφορούν κρυφά και εμφανίζονται σταδιακά στους χρήστες (στο 1 τοις εκατό των χρηστών, μετά στο 5 τοις εκατό, μετά στο 30 τοις εκατό και ούτω καθεξής). Μερικές φορές, μια νέα λειτουργία θα κυκλοφορήσει σκοτεινά για λίγες μέρες και στη συνέχεια θα εξαφανιστεί. Πιθανότατα, αυτό συμβαίνει επειδή δεν είχε καλή απόδοση ή γιατί η εταιρεία απλώς ήθελε να λάβει κάποια αρχικά σχόλια για να καθοδηγήσει την ανάπτυξη. Το μοτίβο απελευθέρωσης συστήματος συμπλέγματος (**Cluster Immune System Release Pattern**) είναι μια επέκταση του μοτίβου ανάπτυξης Canary. Απαιτεί συστήματα παρακολούθησης της απόδοσης και της δραστηριότητας λογισμικού στενά ενσωματωμένα στη διαδικασία έκδοσης. Στο μοτίβο ανάπτυξης Canary, μόλις ολοκληρωθεί η αρχική ανάπτυξη στον έναν διακομιστή, τα συστήματά παρακολούθησης καταγράφουν όλες τις μετρήσεις του συστήματος, της απόδοσης και της δραστηριότητας λογισμικού και δημιουργούν έγκαιρες ειδοποιήσεις εάν τα προβλήματα αυξάνονται σε συγκεκριμένα στάδια. Αυτό έχει ως αποτέλεσμα μια αυτόματη επαναφορά εγκατεστημένου κώδικα σε αυτόν διακομιστή. Εάν οι αυτοματοποιημένες δοκιμές στον διακομιστή και στο σύστημα παρακολούθησης παρέχουν θετικά αποτελέσματα, η ομάδα DevOps είναι πλέον πιο σίγουρη να αναπτύξει τον κώδικά και στα λοιπά σύνολα διακομιστών. Συνοπτικά, το μοτίβο απελευθέρωσης συστήματος συμπλέγματος προσφέρει για την ομάδα DevOps πρόσθετη προστασία για ζητήματα που θα μπορούσαν να χαθούν από τον αυτοματοποιημένο έλεγχο και γρήγορη ανατροφοδότηση και αυτοματοποιημένη επαναφορά για ζητήματα παραγωγής.

Με εναλλαγές λειτουργιών (Feature Toggles), μπορεί να ενεργοποιηθούν και να απενεργοποιηθούν οι δυνατότητες της εφαρμογής. Επομένως, μόλις αναπτυχθούν όλα τα κομμάτια κώδικα που απαιτούνται για μια συγκεκριμένη λειτουργία στο σύστημα παραγωγής, η διάθεση αυτής της δυνατότητας γίνεται με την ενεργοποίηση της εναλλαγής. Οι εναλλαγές δυνατοτήτων είναι συνήθως ρυθμίσεις σε αρχεία διαμόρφωσης συστήματος χρόνου εκτέλεσης ή βάσεις δεδομένων διαμόρφωσης συστήματος. Χάρη στις δυνατότητες εναλλαγής, η ομάδα DevOps είναι σε θέση να απενεργοποιήσει μια λειτουργία εάν μια ανάπτυξη Canary έχει σφάλματα ή μια μη ικανοποιητική εμπειρία χρήστη. Με εναλλαγές δυνατοτήτων, έχουμε επίσης τη δυνατότητα να απενεργοποιήσουμε πόρους. Επομένως, παρόλο που το σύστημά ενδέχεται να έχει προβλήματα κλιμάκωσης με όλες τις δυνατότητές του, μπορούμε να επιτρέψουμε στους πελάτες να έχουν την καλύτερη απόδοση από τις πιο κρίσιμες δυνατότητες του λογισμικού.

3.4 Οικοσύστημα Συνεχούς Παράδοσης

3.4.1 Διαχείριση Υποδομών και Περιβαλλόντων

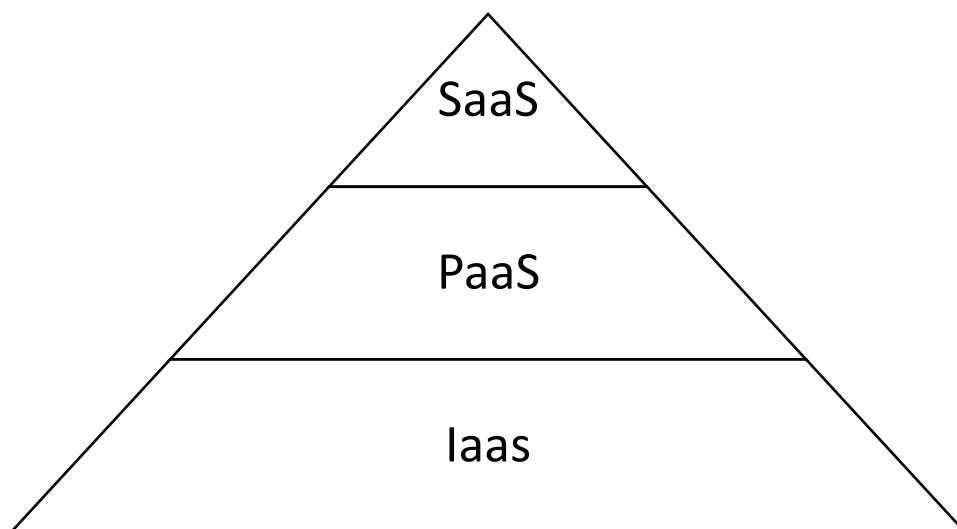
Ένα σύγχρονο λειτουργικό σύστημα έχει πολλούς τρόπους για να διαφοροποιηθεί ως προς την εγκατάσταση του (για παράδειγμα διαφορετικά προγράμματα οδήγησης ή διαφορετική παραμετροποίηση) και αυτό επηρεάζει τις εφαρμογές που τρέχουν σε αυτό. Είναι πολύ σημαντικό να διατηρούμε τις υποδομές με έλεγχο έκδοσης καθώς και να συντηρούμε όλη την παραμετροποίηση που θα είναι συμβατή με την αντίστοιχη έκδοση της εφαρμογής.

Για την παροχή διακομιστών υπάρχουν διάφοροι τρόποι ώστε να δημιουργηθεί το λειτουργικό σύστημα:

- Με πλήρως χειροκίνητη διαδικασία
- Αυτόματη απομακρυσμένη εγκατάσταση
- Με εικονικοποίηση (virtualization)

3.4.2 Υπολογιστικό Νέφος (Cloud Computing)

Το υπολογιστικό νέφος διαθέτει τρία μοντέλα υπηρεσιών:



Σχήμα 3-3: Μοντέλα Υπηρεσιών στο υπολογιστικό νέφος

1. Υποδομή-ως-υπηρεσία - Infrastructure-as-a-Service (IaaS): Είναι το χαμηλότερο επίπεδο υπολογιστικού νέφους και εξασφαλίζει την παροχή υπηρεσιών επεξεργασίας, αποθήκευσης, δικτύων και υπολογιστικών πόρων.
2. Πλατφόρμα-ως-υπηρεσία - Platform-as-a-Service (PaaS): Είναι το μεσαίο επίπεδο που παρέχει πλατφόρμα για την κατασκευή, ανάπτυξη, εκτέλεση, και διαχείριση εφαρμογών λογισμικού.

3. Λογισμικό-ως-υπηρεσία - Software-as-a-Service (SaaS): Βρίσκεται στο υψηλότερο επίπεδο και προσφέρει το λογισμικό που αναπτύσσεται στο PaaS, χωρίς οι χρήστες να έχουν την δυνατότητα ελέγχου της υποδομής και των δυνατοτήτων της εφαρμογής.

3.4.3 Μονολιθική Αρχιτεκτονική

Μια μονολιθική εφαρμογή χρησιμοποιεί ένα κύριο κώδικα και εξυπηρετεί πολλαπλές υπηρεσίες και διεπαφές. Είναι ευρέως διαδεδομένη σε περιπτώσεις ανάπτυξης μικρών εφαρμογών καθώς δεν προσθέτει κάποια αξία ή περαιτέρω κατάτμηση του βασικού κώδικα. Η συνηθέστερη αρχιτεκτονική μονολιθικών εφαρμογών είναι η πολυεπίπεδη και περιλαμβάνει το επίπεδο διεπαφής (User Interface), το στρώμα υπηρεσιών (Business Layer) και το στρώμα πρόσβασης δεδομένων (Data Access Layer) το οποίο έχει πρόσβαση σε μια βάση δεδομένων όπου αποθηκεύονται τα δεδομένα της εφαρμογής.

3.4.4 Αρχιτεκτονική Μικροϋπηρεσιών

Σε αντίθεση με την μονολιθική αρχιτεκτονική, έρχεται μια πιο ευέλικτη αρχιτεκτονική ανάπτυξης λογισμικού, η αρχιτεκτονική μικροϋπηρεσιών που παρέχει τη δυνατότητα μικρού κύκλου ανάπτυξης και συχνότερης παράδοσης. Μεγάλες εφαρμογές χωρίζονται σε μικρότερα κομμάτια που είναι πλήρως κατανοητά ως προς την λειτουργία τους. Στην κατάτμηση σημαντικός παράγοντας είναι οι διασυνδέσεις και ο χρόνος εκτέλεσης που απαιτείται για να γίνουν οι πιθανόν αυξημένες κλήσεις υπηρεσιών.

3.5 Μεθοδολογία SAFe DevOps

Το Scaled Agile Framework® (SAFe) είναι μια διαδικτυακή βάση γνώσεων αποδεδειγμένων, ολοκληρωμένων αρχών, πρακτικών και ικανοτήτων για την εφαρμογή Lean, Agile και DevOps.

3.5.1 Οι Εφτά Ικανότητες του SAFe

3.5.1.1 Αιτή- Ευέλικτη Ηγεσία (Lean-Agile Leadership).

Σε αυτό το κομμάτι η μεθοδολογία βασίζεται σε τρεις βασικούς πυλώνες όπως ο τρόπος σκέψης, η ηγεσία μέσα από παραδείγματα και η ηγεσία μέσα από αλλαγές:

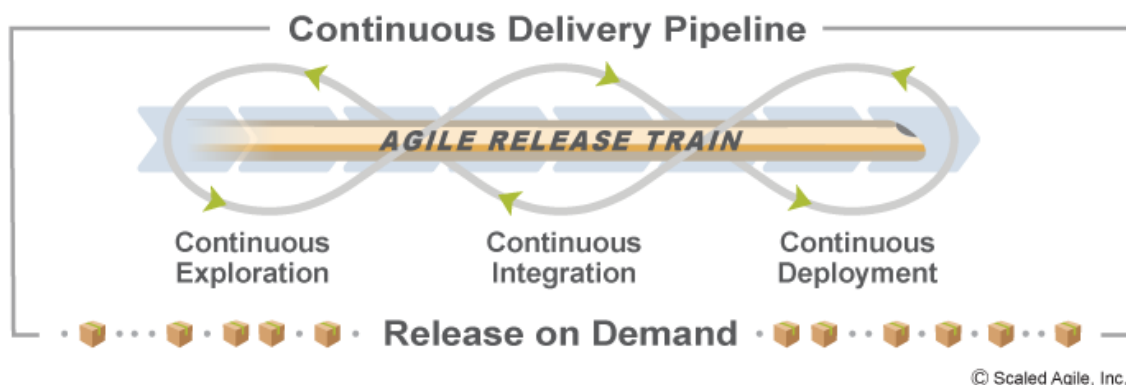
- Για την μετάβαση σε αναπτυσσόμενο τρόπο σκέψης, συνδυάζονται οι πρακτικές του Lean Movement ([Κεφάλαιο 2.2.1](#)) και του Agile Manifesto ([Κεφάλαιο 2.2.2](#)) με βασικές αξίες όπως η ευθυγράμμιση, η ενσωματωμένη ποιότητα, η διαφάνεια και η εκτέλεση του προγράμματος.
- Στην ηγεσία μέσα από παραδείγματα, οι ηγέτες πρέπει να εστιάζουν σε τομείς όπως η αυθεντικότητα, η συναισθηματική νοημοσύνη, η συνεχής γνώση, η ανάπτυξη νέων στελεχών και η αποκεντρωμένη λήψη αποφάσεων.
- Στην ηγεσία μέσα από αλλαγές, εστιάζει στην αλλαγή οράματος και ηγεσίας, στην ψυχολογική ασφάλεια καθώς και στην εκπαίδευση στο νέο τρόπο εργασίας.

3.5.1.2 Ομάδα και Τεχνική Ευελιξία (Team and Technical Agility)

Ομάδες υψηλής απόδοσης που λειτουργούν σε διαλειτουργικά σχήματα και εφαρμόζουν τις βέλτιστες πρακτικές και την ευέλικτη μεθοδολογία. Σε αυτές τις ομάδες χρειάζεται και η ενσωματωμένη ποιότητα ώστε να προσφέρουν αξία στο συντομότερο βιώσιμο χρόνο και να επιτρέπουν στις επιχειρήσεις να ανταποκρίνονται στις αλλαγές με ταχύτατους ρυθμούς.

3.5.1.3 Ευέλικτη Παράδοση Προϊόντος (Agile Product Delivery)

Βασίζεται στην πελατοκεντρική προσέγγιση και σε συνδυασμό με τη σχεδιαστική σκέψη επιβεβαιώνουν ότι οι λύσεις είναι επιθυμητές, εφικτές και βιώσιμες. Ο ρυθμός παράδοσης είναι



Εικόνα 3-3: Διοχέτευση συνεχούς παράδοσης^[20]

βασισμένος στην αγορά και στον πελάτη και επιτρέπουν στον πελάτη να έχει αυτό που θέλει τη στιγμή που το θέλει. Ο αγωγός συνεχούς παράδοσης (Εικόνα 3-3) αντιπροσωπεύει τις ροές εργασίας, τις δραστηριότητες και τους αυτοματισμούς που απαιτούνται για την υλοποίηση μιας ιδέας σε ένα παραδοτέο.

3.5.1.4 Παράδοση Λύσεων για Επιχειρήσεις (Enterprise Solution Delivery)

Οι τρεις διαστάσεις για την παράδοση λύσεων για επιχειρήσεις είναι:

- Τα λιτά συστήματα και οι μηχανισμοί λύσεων με πτυχές όπως η σταθερή βελτίωση της πρόθεσης για σταθερή/μεταβλητή λύση, η εφαρμογή πολλαπλών σχεδιασμών και η αρχιτεκτονική που επιτρέπει επεκτασιμότητα, επιδιόρθωση και συμμόρφωση.
- Ο συντονισμός των «τρένων» των λύσεων και των προμηθευτών που ευθυγραμμίζει τις αξίες της ροής σε μια κοινή αποστολή.
- Συνεχής εξέλιξη συστημάτων μέσω ενός γρήγορου και οικονομικού αγωγού συνεχούς παράδοσης.

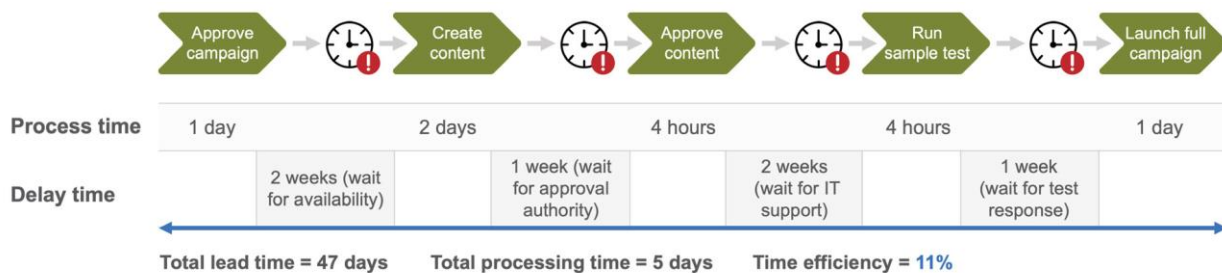
3.5.1.5 Αιτή Διαχείριση Χαρτοφυλακίου (Lean Portfolio Management)

Με τη στρατηγική και τη χρηματοδότηση διασφαλίζεται ότι ολόκληρο το χαρτοφυλάκιο έχει τους πόρους για την επίτευξη των επιχειρηματικών στόχων. Με ευέλικτες λειτουργίες χαρτοφυλακίου συντονίζονται και υποστηρίζονται οι άρτιες εκτελέσεις προγραμμάτων. Τέλος, με τη λιτή διακυβέρνηση παρέχεται η εποπτεία των δαπανών ελέγχου και συμμόρφωσης, η πρόβλεψη δαπανών και οι μετρήσεις.

3.5.1.6 Οργανωτική Ευελιξία (Organizational Agility)

Για να υπάρχει οργανωτική ευελιξία θα πρέπει όλοι οι εμπλεκόμενοι να είναι εκπαιδευμένοι σε Λιτές και Ευέλικτες μεθόδους. Με εργαλεία όπως αυτό των αξιών της ροής (Εικόνα 3-4) γίνεται η αποτύπωση του χρόνου υλοποίησης και μετάβασης από την μια κατάσταση στην άλλη, παρέχοντας τη δυνατότητα για κατανόηση των σημείων που χρήζουν βελτίωση. Για την υλοποίηση της μελέτης περίπτωσης χρησιμοποιήθηκαν μετρήσεις πάνω σε διάφορα στάδια της ροής.

3.5.1.7 Κουλτούρα Συνεχούς Εκπαίδευσης (Continuous Learning Culture)

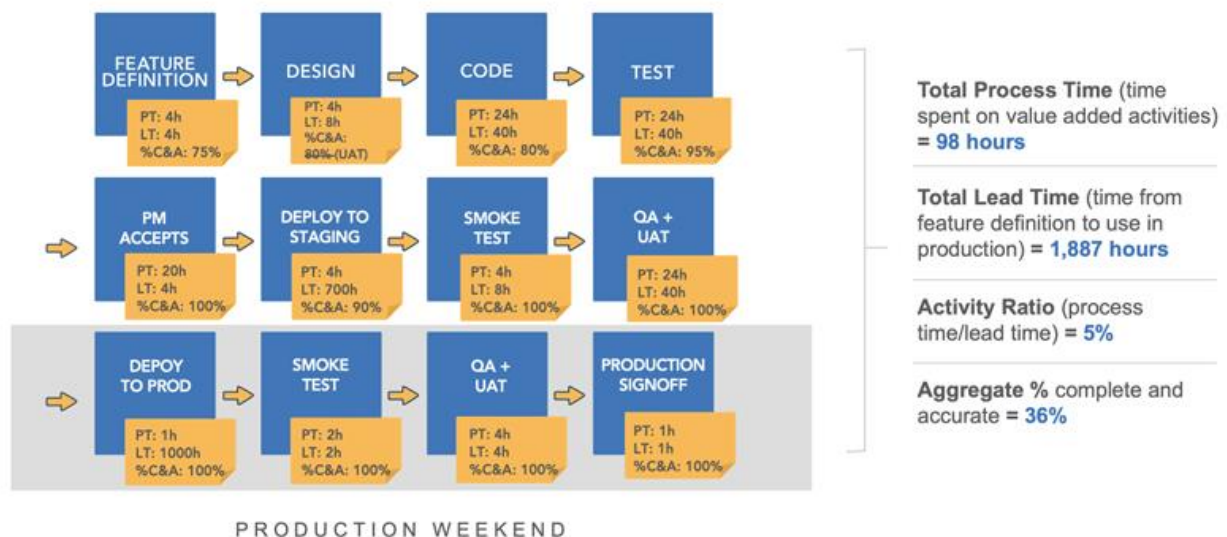


Εικόνα 3-4: Αξίες ροής με χρόνους υλοποίησης και καθυστερήσεις^[20]

Για την κουλτούρα συνεχούς εκπαίδευσης έχει γίνει εκτενής αναφορά στο κεφάλαιο 2.5 και δεν υπάρχει καμία παρέκκλιση στο SAFe σχετικά με τα περιεχόμενα που υιοθετεί η μεθοδολογία.

3.5.2 Χαρτογράφηση των Αξιών Ροής

Για τη χαρτογράφηση των αξιών ροής και τη βελτίωση τους πρέπει να επικεντρωθούμε στον αγωγό ανάπτυξης (Εικόνα 3-3). Από την στιγμή της καταγραφής όλων των σταδίων προχωράμε σε μετρήσεις σε καθένα από αυτά και καταλήγουμε στην μορφή της εικόνας 3-6.



Εικόνα 3-5: Παράδειγμα χαρτογράφησης^[22]

Για τις μετρήσεις χρησιμοποιήθηκαν τα παρακάτω:

- Χρόνος διαδικασίας (PT): είναι ο χρόνος που απαιτείται να ολοκληρωθεί η εργασία από το κάθε στάδιο. Στο παράδειγμα της εικόνας 3-6 χρειάζονται 4 ώρες για να ολοκληρωθεί το στάδιο feature definition.
- Χρόνος παράδοσης (LT): είναι ο χρόνος που απαιτείται από την στιγμή έναρξης της εργασίας (στο προηγούμενο στάδιο) μέχρι την ολοκλήρωσή της (στο τρέχον). Δηλαδή $\text{χρόνος παράδοσης} = \text{χρόνος καθυστέρησης από το προηγούμενο στάδιο} + \text{χρόνος διαδικασίας τρέχοντος σταδίου}$.
- Χρόνος καθυστέρησης: είναι χρόνος κατά τον οποίο δεν γίνεται καμία εργασία.
- Ποσοστό πληρότητας και ακρίβειας (%C&A): είναι το ποσοστό που μπορεί να επεξεργαστεί το επόμενο στάδιο χωρίς να χρειάζεται κάποια άλλη εργασία. Το ποσοστό αυτό μας βοηθά να εντοπίσουμε πιθανές καθυστερήσεις που προκλήθηκαν από την ποιότητα των παραδοτέων του κάθε σταδίου.

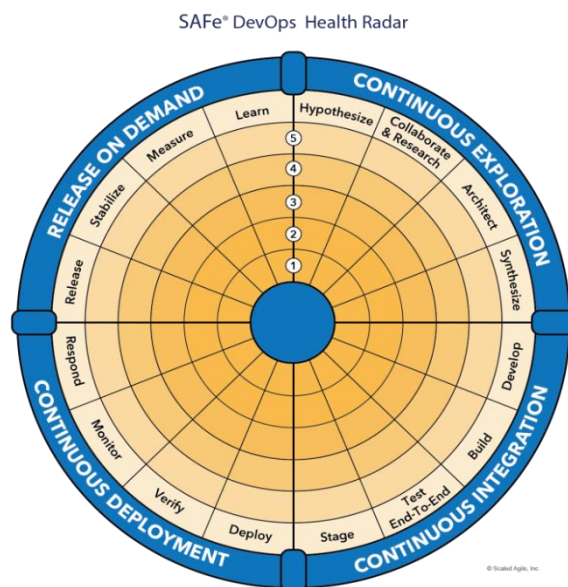
3.5.3 DevOps Health Radar

Το DevOps Health Radar (Εικόνα 3-5) δεν είναι άλλο από τον αγωγό συνεχούς παράδοσης χωρισμένο σε τέσσερα τεταρτημόρια τα οποία ονομάζονται πτυχές. Κάθε πτυχή συμπεριλαμβάνει τέσσερις δραστηριότητες οι οποίες χρησιμοποιούνται για τις μετρήσεις, ώστε να αξιολογηθεί η ωριμότητα ενός οργανισμού ως προς τις δραστηριότητες DevOps.

Οι τέσσερις πτυχές του health radar είναι:

1. Η συνεχής εξερεύνηση (CE) επικεντρώνεται στη δημιουργία ευθυγράμμισης με αυτό που πρέπει να δομηθεί. Στη συνεχή εξερεύνηση η σχεδιαστική σκέψη χρησιμοποιείται για να διασφαλιστεί ότι ο οργανισμός κατανοεί πλήρως την ανάγκη του τελικού χρήστη και την λύση που αυτή απαιτεί. Ξεκινά με μια ιδέα ή υπόθεση (hypothesize) και στην συνέχεια γίνεται ανάλυση και έρευνα (collaborate & research) οδηγώντας στην κατανόηση για το τι απαιτείται ως Ελάχιστο Βιώσιμο Προϊόν (MVP) ή Ελάχιστο Εμπορικό Χαρακτηριστικό (MMF). Τέλος, η σύγκλιση συμβαίνει με την κατανόηση των χαρακτηριστικών που πρέπει τελικά να υλοποιηθούν ώστε να καλύψουν τις ανάγκες του τελικού χρήστη και της αγοράς. Όλα αυτά συγκεντρώνονται, ορίζονται και λαμβάνουν προτεραιότητά προς υλοποίηση στην λίστα εκκρεμοτήτων του προγράμματος (Program Backlog).
2. Η συνεχής ενσωμάτωση (CI) επικεντρώνεται στη λήψη λειτουργιών από την λίστα εκκρεμοτήτων του προγράμματος και την υλοποίησή τους. Στη συνεχή ενσωμάτωση, η εφαρμογή εργαλείων σχεδιαστικής σκέψης στον προβληματικό χώρο επικεντρώνεται στη βελτίωση των χαρακτηριστικών (π.χ. σχεδιασμός ενός χάρτη ιστοριών χρήστη), που μπορεί να παρακινήσει περισσότερη έρευνα και τη χρήση εργαλείων λύσεων (όπως τα σχόλια των χρηστών σε ένα πρωτότυπο). Αφού κατανοηθούν συγκεκριμένα χαρακτηριστικά, οι ομάδες Agile τις εφαρμόζουν. Η ολοκληρωμένη εργασία δεσμεύεται για έλεγχο έκδοσης, δόμηση και ενσωμάτωση σε ένα πλήρες σύστημα ή μία λύση, και δοκιμές από άκρο σε άκρο προτού επικυρωθεί σε περιβάλλον σταδίου (staging environment).

3. Η συνεχής ανάπτυξη (CD) λαμβάνει τις αλλαγές από το περιβάλλον σταδίου και τις αναπτύσσει στην παραγωγή. Σε αυτό το σημείο, επαληθεύονται και παρακολουθούνται για να βεβαιωθούν ότι λειτουργούν σωστά. Αυτό το βήμα καθιστά τα χαρακτηριστικά διαθέσιμα στην παραγωγή, όπου ο πελάτης καθορίζει τον κατάλληλο χρόνο για να τις διαθέσει. Αυτή η πτυχή επιτρέπει επίσης στον οργανισμό να ανταποκρίνεται, να επαναφέρει ή να διορθώνει όταν είναι απαραίτητο.
4. Η διάθεση κατ' απαίτηση (RoD) είναι η δυνατότητα να προσφέρεται αξία στους πελάτες ταυτόχρονα ή σε μια σταδιακή αγορά με βάση τις τάσεις. Αυτό παρέχει μετρήσεις για την υπόθεση και γνώση, με βάση την απόκριση του τελικού χρήστη στην αξία που διατέθηκε ώστε να προγραμματιστούν τα επόμενα βήματα.



Εικόνα 3-6: DevOps Health Radar^[20]

4

Σύσταση Τμήματος DevOps

Στο τέταρτο κεφάλαιο αναλύονται όλες οι πτυχές που πρέπει να ληφθούν υπόψιν για την σύσταση ενός τμήματος DevOps. Γίνεται ανάλυση των δομών τμημάτων και αρχιτεκτονική, τα κριτήρια επιλογής για την δομή, όλοι οι πιθανοί ρόλοι DevOps που μπορούμε να συναντήσουμε και τέλος κάποια ενδεικτικά παραδείγματα εκπαίδευσης βάση του επιπέδου των μηχανικών.

Ο σκοπός του κεφαλαίου αυτού είναι να δώσει την γενικότερη εικόνα, βασισμένο σε πραγματικά παραδείγματα, ώστε να δομηθεί και να συσταθεί ένα τμήμα DevOps. Έχοντας αποκομίσει από τα προηγούμενα δυο κεφάλαια το θεωρητικό υπόβαθρο με τα καθήκοντα και τις εργασίες που αναλαμβάνει ένας μηχανικός DevOps ή μια ομάδα, σε αυτό το κεφάλαιο δένονται με τις δεξιότητες αλλά και τα οργανωτικά κομμάτια DevOps.

4.1 Δομές Τμημάτων DevOps

4.1.1 Εξωτερικές Ομάδες

Συνήθως συναντάμε τη μορφή τέτοιας ομάδας σε οργανισμούς μικρού ή μεσαίου μεγέθους που δεν έχουν την οικονομική δυνατότητα αλλά και την εμπειρία ώστε να καλύψουν τις ανάγκες με σύσταση εσωτερικής αυτόνομης ομάδας. Με εξωτερική ανάθεση σε κάποιον πάροχο ή συμβουλευτική εταιρία καλύπτουν τις ανάγκες για ταχύτερη παράδοση των προϊόντων.

4.1.2 Ενοποιημένες Ομάδες Ανάπτυξης και Λειτουργιών

Τέτοια δομή ομάδας συναντούμε συνήθως σε εταιρίες που διαθέτουν κάποιο ψηφιακό προϊόν. Για παράδειγμα Facebook, YouTube και Netflix είναι μερικές από τις εταιρίες που έχουν υιοθετήσει την κουλτούρα της ενοποίησης των ομάδων ανάπτυξης και λειτουργιών και δεν δουλεύουν πλέον σε σιλό. Οι στόχοι και το όραμα και των δυο αυτών ομάδων είναι κοινοί και με στενή συνεργασία δημιουργούν μια ομάδα DevOps. Επειδή όμως αποτελείται από δυο ξεχωριστές οντότητες συναντάμε και την ονομασία NoOps.

4.1.3 Συνεργασία Ομάδας Ανάπτυξης και Λειτουργιών

Αυτός ο τύπος δομής ομάδας είναι κατάλληλος για οργανισμούς με υγιές τεχνικό υπόβαθρο όπως είναι μια εταιρεία πληροφορικής. Εδώ η ομάδα λειτουργιών και η ομάδα ανάπτυξης είναι ανεξάρτητες ομάδες αλλά συνεργάζονται πολύ καλά. Για παράδειγμα, μετά την ανάπτυξη, εάν υπάρχουν προβλήματα στην παραγωγή, η ομάδα λειτουργιών στέλνει σχόλια στην ομάδα ανάπτυξης και δοκιμής για διόρθωση. Ομοίως, η ομάδα ανάπτυξης και η ομάδα δοκιμών κάνουν τις ενημερώσεις στους διακομιστές και οποιοδήποτε μήνυμα μεταφέρεται στην ομάδα λειτουργιών.

Εδώ η ανάπτυξη μπορεί να έχει πολλά έργα και πολλές ομάδες, ομοίως, η ομάδα λειτουργιών μπορεί να έχει πολλές ομάδες για παρακολούθηση των θεμάτων παραγωγής και συντήρησης.

4.1.4 Υποδομή ως Υπηρεσία (IaaS)

Αυτό είναι σαν μια ομάδα λειτουργιών που εργάζεται σε ένα περιβάλλον νέφους όπως GCP, Azure ή AWS. Πρόκειται για μια προσέγγιση που ταιριάζει στην παραδοσιακή ομάδα λειτουργιών πληροφορικής που έχει αναπτυχθεί βαθύτερα και κλιμακωθεί στο βαθμό που δεν μπορεί να μετατραπεί αρκετά γρήγορα σε μια ομάδα DevOps και δεν μπορεί να συνδυαστεί με τις ομάδες ανάπτυξης. Είναι σαν να δουλεύεις με το Amazon EC2 που είναι μια επεκτάσιμη ιστοσελίδα που παρέχει μια δυνατότητα υπολογισμού με δυνατότητα αλλαγής μεγέθους. Σε αυτόν τον τύπο δομής ομάδας η ομάδα DevOps λειτουργεί ξεχωριστά από την πλευρά της ανάπτυξης. αυτή είναι η μικρή ομάδα που συνδυάζει ομάδες λειτουργιών και ομάδες ανάπτυξης. Η λειτουργία της ομάδας DevOps είναι η παρακολούθηση των υπηρεσιών και της υποδομής, η κατανομή πόρων, οι μετρήσεις και τα χαρακτηριστικά των συστημάτων, καθώς και η επικοινωνία και η συνεργασία με την ομάδα IaaS.

4.1.5 DevOps στον Οργανισμό (εσωτερική ομάδα)

Εδώ υπάρχουν αρκετοί άνθρωποι στην ομάδα DevOps που ενεργούν ως σύμβουλοι της ομάδας πληροφορικής και ανάπτυξης. Ωστόσο, αυτό δεν μπορεί να ονομαστεί ως ώριμη δομή ομάδας και είναι πιο κατάλληλη για μικρές και μεσαίες επιχειρήσεις. Με την έναρξη της στελέχωσης εσωτερικής ομάδας DevOps θα εισέλθουν αργά τα οφέλη και θα εξυπηρετήσουν τους γενικούς στόχους και την κουλτούρα DevOps. Στις μέρες μας, τα οφέλη δεν είναι ορατά καθώς η πληροφορική και η ομάδες ανάπτυξης λειτουργούν ακόμη σε σιλό.

4.1.6 Μηχανικοί Αξιοπιστίας Ιστότοπου (Site Reliability Engineers)

Αυτό είναι ένα νέο μοντέλο (**Google Model**) για τις εταιρείες που έχουν ώριμες ομάδες λειτουργίας και ανάπτυξης. Εδώ το μοντέλο συνεργάζεται με την ομάδα DevOps και SRE και μαζί σχηματίζουν το σύνδεσμο μεταξύ της ομάδας ανάπτυξης και λειτουργίας. Το SRE σημαίνει το Site Reliability Engineering, το οποίο είναι μια νέα ομάδα που σχηματίζεται και παίρνει τον κώδικα από την ομάδα ανάπτυξης με τις μετρήσεις και σχετική απόδειξη ότι ο κώδικας είναι κατάλληλος για ανάπτυξη στην παραγωγή. Το SRE διασφαλίζει ότι το προϊόν πληροί όλες τις απαιτήσεις λειτουργίας και είναι έτοιμο για την παραγωγή. Οι ομάδες SRE και ανάπτυξης συνεργάζονται και η SRE εγκρίνει τον κώδικα με βάση τις λειτουργικές μετρήσεις. Σε αυτήν την περίπτωση, οι ομάδες ανάπτυξης έχουν την ομάδα DevOps τους.

4.2 Κριτήρια Επιλογής Δομής DevOps

4.2.1 Αποφυγή Big Bang

Ο τρόπος που ξεκινάμε το ταξίδι DevOps στον Οργανισμό θα μας υποδείξει και το πόσο επιτυχής θα αποδειχθεί η πρωτοβουλία μας για DevOps. Πρέπει να αξιοποιήσουμε στο έπακρο τα δεδομένα και τους πόρους ξεκινώντας με μικρά βήματα και μικρές αλλαγές με αποφυγή του Big Bang. Δεν μπορούμε μέσα σε μια μέρα, με μία κίνηση, να ανακοινώσουμε ριζικές αλλαγές για έναν οργανωτικό μετασχηματισμό DevOps. Πρέπει πρώτα να προσδιορίσουμε τις ομάδες:

- που αντιμετωπίζουν συνεχώς προβλήματα και ανεκπλήρωτους στόχους.
- που είναι σίγουρα συνειδητές σχετικά με την αναγκαιότητα βελτιώσεων και οι οποίες είναι πιο δεκτικές στο να εξετάσουν έναν διαφορετικό τρόπο εργασίας.
- που είναι ανεπιφύλακτα πρόθυμες να εξυπηρετήσουν την οργάνωση και τους πελάτες τους.

4.2.2 *Επίτευξη Μικρών Στόχων και Γρήγορες Νίκες*

Στόχος είναι η επίτευξη γρήγορων και πρώιμων νικών από την πρωτοβουλία στο DevOps και η ανάδειξη για τους σκεπτικιστές και αρνητές των προβλημάτων που επιλύουμε προβάλλοντας ταυτόχρονα την αξία που δημιουργούμε με το DevOps. Στη συνέχεια αναπαράγουμε βήμα προς βήμα αυτές τις βελτιώσεις στους άλλους τομείς του οργανισμού.

4.2.3 *Καθορισμός Προϊόντος και Υπηρεσίας Greenfield ή Brownfield*

Ένα άλλο ζήτημα για να ξεκινήσουμε με την πρωτοβουλία DevOps είναι εάν θα πρέπει να επιλέξουμε ένα προϊόν και υπηρεσία Greenfield ή Brownfield. Ένα προϊόν και υπηρεσία Greenfield είναι νέο στον οργανισμό. Επομένως, είναι προς το συμφέρον μας να δημιουργήσουμε μια νέα, ξεχωριστή ομάδα που εργάζεται μεμονωμένα, ειδικά όταν ξεκινάμε. Χάρη σε αυτήν την απομόνωση, τα μέλη της ομάδας θα έχουν περισσότερες πιθανότητες να σπάσουν το υπάρχον σιλό με δημιουργική σκέψη. Επιπλέον, η ομάδα θα διακόπτεται λιγότερο συχνά λόγω μη ύπαρξης προηγούμενων ευθυνών τους στον οργανισμό. Η πρωτοβουλία στο **Greenfield** DevOps δεν βασίζεται σε κώδικα και εξαρτήσεις παλαιού τύπου και θέτει λιγότερους κινδύνους. Ωστόσο, η επιτυχία θα έχει λιγότερη επίδραση στον οργανισμό, καθώς η πρωτοβουλία DevOps στο Greenfield δεν θα αποδεικνύει εάν το DevOps μπορεί να λύσει τα υπάρχοντα προβλήματα στον οργανισμό σας και εάν η επιτυχία μπορεί να αναπαραχθεί και σε άλλους τομείς. Από την άλλη πλευρά, μπορούμε να ξεκινήσουμε το DevOps με ένα προϊόν και υπηρεσία **Brownfield**. Πολλά προϊόντα και υπηρεσίες Brownfield λειτουργούν ήδη στα συστήματα παραγωγής και εξυπηρετούν την επιχείρηση και τους πελάτες. Η εφαρμογή λογισμικού Brownfield είναι πιθανό να έχει σημαντικό τεχνικό χρέος και δεν έχει πολύ αξιόπιστο αυτοματοποιημένο έλεγχο.

4.2.4 *DevOps για Μεγάλους Οργανισμούς με Μονολιθικά Συστήματα*

Πολλοί άνθρωποι πιστεύουν λανθασμένα ότι το DevOps δεν είναι προσαρμοσμένο για προϊόντα και υπηρεσίες Brownfield. Αυτή η πεποίθηση ενισχύεται ακόμη περισσότερο για να δηλώσει ότι το DevOps προορίζεται μόνο για νεοσύστατες εταιρείες. Ωστόσο, μερικές από τις πιο υποδειγματικές πρωτοβουλίες DevOps ξεκίνησαν σε εταιρείες με γιγάντιους και ώριμους οργανισμούς πληροφορικής όπως η **Amazon**, **Easy** και **Facebook**. Όπως περιγράφεται παραπάνω μπορούμε να ξεκινήσουμε το DevOps σε έναν οργανισμό με προϊόντα και υπηρεσίες Brownfield. Στην πραγματικότητα, επειδή τα περισσότερα προϊόντα και υπηρεσίες Brownfield έχουν μεγάλο χάσμα μεταξύ της απόδοσης και των προσδοκιών των πελατών, είναι συνήθως τα πιο κατάλληλα για να επωφεληθούν από τον μετασχηματισμό στο DevOps.

4.2.5 *Καθορισμός Κατάλληλου Συστήματος Εγγραφών*

Αρχικά, θα χρειαστεί να επενδύσουμε χρόνο για να αφαιρέσουμε ή τουλάχιστον να μειώσουμε το τεχνικό χρέος και, στη συνέχεια, να δημιουργήσουμε ελλιπείς αυτοματοποιημένες

δοκιμές για υπάρχουσες δυνατότητες και οφέλη του προϊόντος. Διαφορετικά, το προϊόν και η υπηρεσία Brownfield δεν θα επιτρέψει ποτέ να κάνουμε αξιόπιστες εφαρμογές παραγωγής σε μικρότερους κύκλους. Προσδιορίζουμε ένα κατάλληλο σύστημα εγγραφών ή ένα σύστημα δέσμευσης. Κατά το πρώτο εγχείρημα DevOps, μπορούμε να επιλέξουμε είτε ένα σύστημα εγγραφών είτε ένα σύστημα αφοσίωσης. Τα συστήματα αρχείων είναι κεντρικά συστήματα για τη διαχείριση μιας μοναδικής πηγής αλήθειας στον Οργανισμό μας. Κυριαρχούν σε μεγάλο βαθμό στη διαχείριση των δεδομένων, συμπεριλαμβανομένων πελατών, συναλλαγών, ασφάλειας και απορρήτου. Τα συστήματα αφοσίωσης είναι ως επί το πλείστον αποκεντρωμένες εφαρμογές για να επιτρέπουν και να ενθαρρύνουν τους πελάτες και τους χρήστες να αλληλοεπιδρούν με τον οργανισμό μέσω απομακρυσμένων καναλιών ή εφαρμογών για κινητά, tablet και επιτραπέζιους υπολογιστές.

Με την προτίμηση για ένα σύστημα εγγραφής, το πλεονέκτημα θα είναι ότι θα μπορέσουμε να αποδείξουμε πώς η ομάδα DevOps χειρίζεται αποτελεσματικά χρονοβόρες και εξαιρετικά γραφειοκρατικές εργασίες, όπως η συμμόρφωση ασφάλειας και το απόρρητο. Το επιτυγχάνουμε δημιουργώντας επιστημονικά και απτά στοιχεία και χρησιμοποιώντας αυτοματοποιημένο έλεγχο και τηλεμετρία καθ' όλη τη διάρκεια του κύκλου ζωής ανάπτυξης και λειτουργίας του προϊόντος και της υπηρεσίας. Ένα τυπικό παράδειγμα μπορεί να είναι ένα πρόγραμμα μετεγκατάστασης των γεωγραφικά κατανεμημένων δεδομένων από τους διακομιστές βάσης δεδομένων εγκατάστασης στο υπολογιστικό νέφος (cloud), διασφαλίζοντας ότι η μοναδική πηγή αλήθειας (source of truth) εξακολουθεί να συμμορφώνεται με τους νομικούς, βιομηχανικούς, τοπικούς και διεθνείς κανόνες και κανονισμούς.

4.3 Πιθανοί Ρόλοι DevOps στον Οργανισμό

Οι ρόλοι που προτείνονται παρακάτω είναι ενδεικτικοί ρόλοι και μπορούν να αποδοθούν σε πολλά διαφορετικά προφίλ σε έναν οργανισμό, ανεξάρτητα από την ιεράρχηση και το γνωστικό υπόβαθρο αυτών των προφίλ. Σε μια εταιρία που έχει υιοθετήσει την DevOps κουλτούρα βρίσκουμε στην καθημερινότητα των εργασιών των εργαζομένων την ενασχόληση με κάποιους από τους παρακάτω ρόλους, σύμφωνα με τις περιγραφές τους, χωρίς αυτό να σημαίνει ότι έχουν και τον αντίστοιχο τίτλο.

4.3.1 DevOps Generalist

Ο κύριος ρόλος του DevOps Generalist είναι να διασφαλίσει την ομαλή δημιουργία, την αποτελεσματική, την υγιή πρόοδο και τη συνεχή βελτίωση των Πρακτικών DevOps σε έναν οργανισμό DevOps και στις ομάδες DevOps. Επομένως, η ικανότητα και η προοπτική κάθε εργαζομένου σε έναν οργανισμό με DevOps να μπορεί να ενεργεί με τις ομάδες DevOps είναι ένας θεμελιώδης παράγοντας που καθορίζει το επίπεδο επιτυχίας και τη διάρκεια ζωής των οργανισμών DevOps. Είτε ως μέλη ενός οργανισμού με DevOps είτε απλά συνεργαζόμενοι με άλλες ομάδες DevOps, είναι εξαιρετικά σημαντικό να έχουμε μια σαφή κατανόηση για το πώς και τι καθιστά τη μεθοδολογία DevOps πολύ πιο επιτυχημένη, αποτελεσματική και ευχάριστη στη συνεργασία σε σχέση με άλλες μεθοδολογίες ανάπτυξης και παράδοσης λογισμικού. Επομένως, ανεξάρτητα από το εάν κάποιος είναι επαγγελματίας πληροφορικής, λογισμικού και τεχνολογίας ή ακόμα και

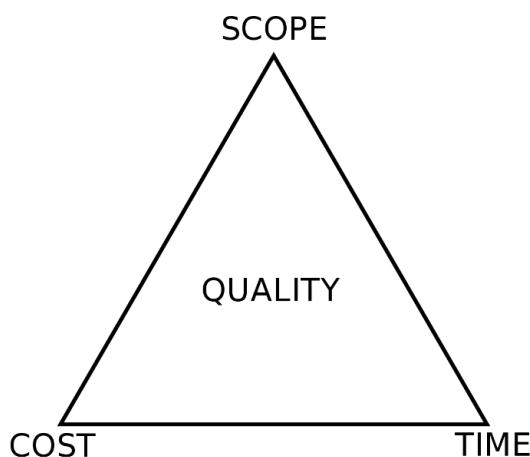
διευθυντής, σε αυτή την τρέχουσα εποχή ψηφιοποίησης, συνιστάται ιδιαίτερα η υιοθέτηση του ρόλου DevOps Generalist.

4.3.2 DevOps Executive

Τα στελέχη DevOps Executive είναι υπεύθυνα για την επιτυχή χρήση και εφαρμογή της τεχνογνωσίας DevOps στους Οργανισμούς τους. Αποτελούν βασικούς παίκτες για τις ομάδες DevOps που επιτρέπουν την πολιτιστική μετατόπιση του να κάνουν και να σκέφτονται με τρόπο DevOps. Έχουν αποδεδειγμένη ικανότητα να επηρεάζουν εποικοδομητικά τις ομάδες και τη διοίκηση. Τα στελέχη DevOps Executive υποστηρίζουν την ευθυγράμμιση των ομάδων DevOps με τις επιχειρηματικές στρατηγικές και είναι υπεύθυνες για τον εντοπισμό, την επιλογή, το πεδίο εφαρμογής, την ιεράρχηση και τελικά τη διαχείριση της διείσδυσης των DevOps στους οργανισμούς. Τα στελέχη DevOps Executive επιλέγουν βασικά μέλη DevOps και διασφαλίζουν ότι όλοι οι επαγγελματίες DevOps έχουν εκπαιδευτεί, ανατεθεί και αναπτυχθεί σε έργα DevOps που ταιριάζουν καλύτερα στα επίπεδα δεξιοτήτων και εμπειριών τους. Εργάζονται σκληρά για να καθοδηγήσουν τις ομάδες DevOps, να υποστηρίξουν τον προγραμματισμό πόρων και να αφαιρέσουν ζητήματα και εμπόδια, για να κάνουν ολόκληρο τον οργανισμό DevOps επιτυχημένο. Τα στελέχη DevOps Executive αναφέρονται στα ανώτατα στελέχη διαχείρισης (C-Level: CEO, CTO, CFO, etc) όσον αφορά τους προκαθορισμένους στόχους επιχειρηματικής αποστολής DevOps και προσδιορίζουν τις μετρήσεις επιχειρησιακής απόδοσης. Προωθούν λύσεις βέλτιστων πρακτικών, ιστορίες επιτυχίας και την αξιοποίησή τους σε ολόκληρο τον οργανισμό. Δουλεύουν πολύ στενά με όλες τις ομάδες DevOps, αλλά με ιδιαίτερη έμφαση στους DevOps Project Managers, DevOps Product Owners, DevOps Release Managers, DevOps Trainers and DevOps Coaches λόγω της ιδιαίτερης οργανωτικής τους αποστολής που προσπαθούν να πετύχουν.

4.3.3 DevOps Project Manager

Ο DevOps Project Manager είναι το άτομο που είναι υπεύθυνο για την επίτευξη των δηλωθέντων στόχων του έργου. Οι βασικές αρμοδιότητες του διαχειριστή έργου DevOps περιλαμβάνουν τη δημιουργία σαφών και εφικτών στόχων έργου, τη δημιουργία των απαιτήσεων του έργου και τη διαχείριση των περιορισμών του τριγώνου διαχείρισης έργου, που είναι το κόστος, το πρόγραμμα, το πεδίο εφαρμογής και η ποιότητα (Εικόνα 4-1).



Εικόνα 4-1: Τρίγωνο Διαχείρισης Έργου

Ο DevOps Project Manager είναι συχνά ένας εκπρόσωπος του πελάτη και πρέπει να καθορίσει και να εφαρμόσει τις ακριβείς ανάγκες του πελάτη, με βάση τις γνώσεις της εταιρείας που εκπροσωπεί. Ο Διαχειριστής έργου DevOps συνδέει το χάσμα γεφύρωσης μεταξύ της ομάδας ανάπτυξης / παράδοσης και του πελάτη. Επομένως, ο Διαχειριστής έργου DevOps έχει μια ολοκληρωμένη γνώση του κλάδου στον οποίο δραστηριοποιείται, έτσι ώστε να είναι σε θέση να κατανοήσει και να συζητήσει τα προβλήματα με την ομάδα

παράδοσης και τον πελάτη. Έχει την ικανότητα προσαρμογής στις διάφορες εσωτερικές διαδικασίες, και δημιουργεί στενούς δεσμούς με όλους τους εμπλεκόμενους, για να διασφαλιστεί ότι τα βασικά ζητήματα που σχετίζονται με το κόστος, το χρονοδιάγραμμα, το εύρος και την ποιότητα μπορούν να επιλυθούν αποτελεσματικά και, πάνω από όλα, να ικανοποιήσουν τον πελάτη. Ο όρος και ο τίτλος "DevOps Project Manager" περιγράφει το άτομο που έχει την ευθύνη να ολοκληρώσει ένα έργο. Ο DevOps Project Manager έχει το απαιτούμενο επίπεδο ευθύνης και εξουσίας για την επίτευξη των επιθυμητών στόχων του έργου εντός του προϋπολογισμού του έργου, εγκαίρως και με την υψηλότερη δυνατή ποιότητα.

4.3.4 DevOps Product Owner

Ο ρόλος του ιδιοκτήτη προϊόντων DevOps είναι ένας πολύ μοναδικός και ευρύς ρόλος στο DevOps που συνδυάζει όλες τις απαιτητικές πτυχές των παραδοσιακών ρόλων του Project Manager και του Product Manager. Επιπλέον, ο DevOps Product Owner αντιπροσωπεύει την άποψη των πελατών σε μια ομάδα DevOps διασφαλίζοντας ότι η σωστή εργασία γίνεται την κατάλληλη στιγμή. Πρέπει να ενσωματωθεί με τις συνολικές ομάδες και διαδικασίες ανάπτυξης και παράδοσης λογισμικού DevOps για να εξασφαλιστεί η μέγιστη προστιθέμενη αξία για κάθε κυκλοφορία προϊόντος. Οι ιδιοκτήτες προϊόντων DevOps έχουν καθορισμένες βασικές ευθύνες.

Κάποιες από αυτές είναι:

- Διαχείριση αποθεμάτων προϊόντων (Product Backlog) DevOps.
- Υποστήριξη δημιουργιών "οδικών χαρτών" κυκλοφορίας προϊόντων, και σχέδια διάθεσης.
- Προσδιορισμός των εξαρτήσεων των προϊόντων και κατάλληλη ιεράρχηση προτεραιοτήτων.
- Διαχείριση και επικοινωνία με τα ενδιαφερόμενα μέρη.

4.3.5 Devops Architect

Ο Devops Architect έχει την ευθύνη για την αρχιτεκτονική, το σχεδιασμό και την ανάπτυξη εργαλείων και διαδικασιών ανάπτυξης προϊόντων. Σε αυτόν τον ρόλο, ο DevOps Architect αναμένεται να δημιουργήσει και να αναπτύξει καινοτόμες λύσεις για την κατασκευή και συντήρηση της αρχιτεκτονικής των προϊόντων, των σχετικών εργαλείων και διαδικασίες για συνεχή ολοκλήρωση με αγωγό συνεχούς ανάπτυξης. Οι αρχιτέκτονες του DevOps είναι υπεύθυνοι για τον καθορισμό ενός χαλαρά συνδεδεμένου συνόλου υπηρεσιών των οποίων οι καταναλωτές επηρεάζονται ελάχιστα από αλλαγές σε αυτήν την υπηρεσία ή στο περιβάλλον τους.

Κάποια σύζευξη είναι προφανώς αναπόφευκτη δεδομένου ότι ο καταναλωτής πρέπει να κάνει χρήση της υπηρεσίας, αλλά οι αρχιτέκτονες DevOps ελαχιστοποιούν αυτές τις εξαρτήσεις μέσω διαχωρισμού διεπαφής και υλοποίησης, πολιτικών εκδόσεων, συμβάσεων χρόνου εκτέλεσης, ανεξαρτησίας πλατφόρμας και διαφάνειας τοποθεσίας. Οι αρχιτέκτονες DevOps γίνονται εξαιρετικά κρίσιμοι για την επιτυχία των οργανισμών και ομάδων DevOps. Στην κορυφή της δημιουργίας μιας βέλτιστης χαλαρά συνδεδεμένης αρχιτεκτονικής για προϊόντα και υπηρεσίες, οι οργανισμοί με DevOps πρέπει να διαθέτουν ένα εξαιρετικά αξιόπιστο περιβάλλον, πλήρως αυτοματοποιημένο και απαλλαγμένο από εμπόδια.

4.3.6 DevOps Developer

Ο προγραμματιστής DevOps, μεταμορφώνει τους επιχειρηματικούς στόχους των πελατών σε λύσεις λογισμικού και συστήματα. Η κύρια διαφορά μεταξύ ενός απλού προγραμματιστή και ενός προγραμματιστή DevOps είναι ότι ο προγραμματιστής DevOps, καθ' όλη τη διάρκεια της εργασίας, γνωρίζει τους επιχειρηματικούς στόχους και τις επιχειρηματικές απαιτήσεις των πελατών. Επιπλέον, ο προγραμματιστής DevOps, είναι ο αξιόπιστος συνεργάτης τεχνολογίας των πελατών και εμπνέει στους εργοδότες και τους πελάτες την πλήρη εμπιστοσύνη. Είναι ένα άτομο που συνδέει τους επιχειρηματικούς στόχους και τις επιχειρησιακές απαιτήσεις με τα σχέδια λογισμικού και τις γραμμές κώδικα λογισμικού και υπεύθυνος για το σχεδιασμό ολόκληρου του κύκλου ζωής λογισμικού από την ανάλυση απαιτήσεων, το αρχιτεκτονικό όραμα και τον αυτοματοποιημένο έλεγχο έως την εμπειρία χρήστη, τις επεκτάσεις, τις λειτουργίες, τη συντήρηση έως το τέλος της ζωής του. Ο προγραμματιστής DevOps, είναι υπεύθυνος για την ανάπτυξη λύσεων λογισμικού που εκτελούνται σε μικρές και συχνές επαναλήψεις.

4.3.7 DevOps Operations Engineer

Ο Μηχανικός λειτουργιών DevOps είναι υπεύθυνος για την παρακολούθηση, τη συντήρηση και την ανάπτυξη του υπερσύγχρονου λογισμικού και της υποδομής πίσω από την τεχνολογία των προϊόντων τους. Αναπτύσσει και συντηρεί υποδομές δικτύου και διακομιστές στα κέντρα δεδομένων. Συμμετέχει επίσης σε ομάδες παράδοσης και ανάπτυξης DevOps και αναπτύσσει σχέδια έκτακτης ανάγκης παράδοσης και ανάπτυξης προϊόντων. Υπάρχει άμεση και στενή συνεργασία με τους προγραμματιστές DevOps για να διασφαλιστεί η διαθεσιμότητα, η συντήρηση, και η δυνατότητα παρακολούθησης των λειτουργιών των συστημάτων. Πίσω από όλα όσα βλέπουν οι πελάτες του οργανισμού είναι η αρχιτεκτονική που χτίστηκε από τους DevOps Operations Engineers. Οι DevOps Operations Engineers είναι υπεύθυνοι για τη διατήρηση και λειτουργία αυτών των συστημάτων. Από την ανάπτυξη και τη συντήρηση προϊόντων και υπηρεσιών έως την οικοδόμηση υποδομών της επόμενης γενιάς. Οι Μηχανικοί Λειτουργίας DevOps καθιστούν δυνατή την οργάνωσή του χαρτοφυλακίου προϊόντων. Είναι μηχανικοί και βρίσκονται πάντα σε ετοιμότητα για να διατηρήσουν τα συστήματα και τα δίκτυά σε λειτουργία, διασφαλίζοντας ότι οι πελάτες έχουν την καλύτερη και ταχύτερη δυνατή εμπειρία.

4.3.8 DevOps Quality Assurance Engineer

Οι μηχανικοί διασφάλισης ποιότητας DevOps διαδραματίζουν ενεργό ρόλο με γόμωνα την επεξεργασία των απαιτήσεων, των περιπτώσεων χρήσης, της αρχιτεκτονικής λογισμικού και του υλικού σχεδιασμού λογισμικού έτσι ώστε να ανακαλύψουν τους επιθυμητούς τύπους δοκιμών για την επικύρωση της ποιότητας του προϊόντος. Συνεργάζονται με τους προγραμματιστές DevOps και τους διαχειριστές έργων DevOps για να καθορίσουν τις μεθόδους και τα εργαλεία υλοποίησης δοκιμής για την εκτέλεση και τη λειτουργία της δοκιμαστικής εργασίας. Οι μηχανικοί διασφάλισης

ποιότητας DevOps γράφουν λίστες δημιουργικών δοκιμαστικών περιπτώσεων. Κατατάσσουν επίσης τα επίπεδα προτεραιότητας και δυσκολίας των υποθέσεων δοκιμής τους. Δημιουργούν και αναθεωρούν λεπτομερή τεκμηρίωση δοκιμαστικών περιπτώσεων για να βεβαιωθούν ότι οι δοκιμαστικές υποθέσεις γίνονται κατανοητές από την υπόλοιπη ομάδα DevOps.

Οι Μηχανικοί Διασφάλισης Ποιότητας DevOps παρακολουθούν συνεχώς και επανεξετάζουν τις φάσεις εκτέλεσης δοκιμών για να διασφαλίσουν ότι οι εφαρμοσμένες δοκιμές υλοποιούν τους στόχους τους. Λειτουργούν ως αξιόπιστοι ειδικοί για να υπογραμμίσουν τον στόχο και τη σημασία των υποθέσεων δοκιμής. Αυξάνουν και αναθεωρούν ελαττώματα και διασφαλίζουν ότι δεν υπάρχει συμβιβασμός από τις προγραμματισμένες δυνατότητες προϊόντος. Επιπλέον, οι Μηχανικοί Διασφάλισης Ποιότητας DevOps καλλιεργούν ολόκληρη την Ομάδα DevOps για να επανεξετάσουν τα σχέδια δοκιμών και να επεξεργαστούν τα σχόλια για να προσθέσουν επιπλέον δημιουργικές δοκιμαστικές περιπτώσεις και για να μειώσουν τις απολύσεις στα δοκιμαστικά τους σχέδια. Τέλος, κατά τη διάρκεια ενός σημαντικού μέρους του χρόνου τους, αυτοματοποιούν, εκτελούν δοκιμές ξανά και ξανά για την επικύρωση διορθώσεων σφαλμάτων που προέρχονται από τις ομάδες ανάπτυξης λογισμικού.

4.3.9 DevOps Information Security Engineer

Στις παραδοσιακές διαμορφώσεις των οργανισμών πληροφορικής, η ασφάλεια των πληροφοριών βρίσκονται σε πρώιμα στάδια ως άλλη μια μη λειτουργική απαίτηση που συχνά λαμβάνεται υπόψη όταν πια είναι πιο δύσκολος και ακριβός ο εντοπισμός και η επίλυση των προβλημάτων. Οι Μηχανικοί Ασφάλειας Πληροφοριών DevOps σχεδιάζουν τη μεγάλη στρατηγική ασφάλειας των οργανισμών τους, ενώ παράλληλα καθορίζουν τις λεπτομέρειες ενός σχεδίου εφαρμογής. Κατανοούν τη συνεχή ανάγκη εξισορρόπησης των πλεονεκτημάτων των αυξητικών μέτρων ασφαλείας με τις πιθανές επιβαρύνσεις για την επιχείρηση. Βρίσκουν προληπτικά και επιδιορθώνουν προβλήματα ασφαλείας σε σχέδια στα πρώτα στάδια της μηχανικής των προϊόντων και σε συστήματα λογισμικού που λειτουργούν. Παρακολουθούν τα δίκτυα, τη τηλεμετρία λογισμικού και δίνουν προτεραιότητα στις σφάλματα βάσει κινδύνου. Οι οργανισμοί με DevOps διαθέτουν μηχανικούς ασφαλείας πληροφοριών DevOps συνεργαζόμενοι με τους προγραμματιστές DevOps και τους μηχανικούς λειτουργίας DevOps. Οι Μηχανικοί Ασφάλειας Πληροφοριών DevOps επιτρέπουν στους οργανισμούς τους να ενσωματώνουν ασφάλεια στο προϊόν καθ' όλη τη διάρκεια του κύκλου ζωής παράδοσης από άκρο σε άκρο.

4.3.10 DevOps Release Manager

Οι διαχειριστές διάθεσης DevOps εργάζονται για την αντιμετώπιση της διαχείρισης και του συντονισμού της διάθεσης του προϊόντος, από την ανάπτυξη έως την παραγωγή. Συνήθως εργάζονται στις περισσότερες από τις τεχνικές λεπτομέρειες και εμπόδια στα οποία δεν μπορεί να εμπλακεί ένας παραδοσιακός διαχειριστής έργου. Οι διαχειριστές διάθεσης DevOps επιβλέπουν το συντονισμό, την ενσωμάτωση και τη ροή ανάπτυξης, δοκιμών και ανάπτυξης για την υποστήριξη της συνεχούς παράδοσης. Επικεντρώνονται όχι μόνο στη δημιουργία, αλλά και στη διατήρηση της εργαλειοθήκης παράδοσης εφαρμογών από άκρο σε άκρο. Οι διαχειριστές διάθεσης DevOps

συνεργάζονται στενά με τους διαχειριστές έργων DevOps και τους ιδιοκτήτες προϊόντων DevOps για να δημιουργήσουν "οδικούς χάρτες" έκδοσης προϊόντων, σχέδια διάθεσης, να προσδιορίσουν εξαρτήσεις, να διασφαλίσουν ορατότητα και την ιεράρχηση των εξαρτημένων εργασιών από τις ομάδες DevOps, τη διαχείριση υποστήριξης των αποθεμάτων προϊόντων DevOps, τη διαχείριση ενδιαφερομένων και την επικοινωνία .

4.3.11 DevOps Trainer

Οι εκπαιδευτές DevOps είναι ταλαντούχοι υποστηρικτές του DevOps για να διασφαλίσουν τη σωστή εκπαίδευση των πρακτικών του DevOps μέσα σε οργανισμούς. Οι εκπαιδευτές DevOps μπορούν να είναι ανεξάρτητα άτομα από τις ομάδες DevOps στους οργανισμούς και μπορούν να χρηματοδοτηθούν απευθείας από τα ανώτερα στελέχη για να επιτρέψουν την κορυφαία προς τα κάτω οργανωτική εκπαίδευση των DevOps. Εναλλακτικά, οι εκπαιδευτές DevOps μπορούν να συμμετέχουν σε ομάδες DevOps και να συνεργάζονται με στελέχη για να λάβουν την απαιτούμενη υποστήριξη και να εκπαιδεύουν με συνέπεια άλλα μέρη οργανισμών ώστε να ταιριάζουν στις ομάδες DevOps.

4.3.12 DevOps Coach

Παρόμοια με τους εκπαιδευτές DevOps, οι DevOps Coaches είναι επίσης ταλαντούχοι υποστηρικτές του DevOps για να διασφαλίσουν τη σωστή κατανόηση και υιοθέτηση πρακτικών DevOps εντός οργανισμών. Οι DevOps Coaches μπορούν να είναι ανεξάρτητα άτομα από τις ομάδες DevOps στους οργανισμούς και μπορούν να χρηματοδοτηθούν άμεσα από τα ανώτερα στελέχη για να επιτρέψουν την κορυφαία προς τα κάτω οργανωτική και πολιτιστική υιοθέτηση του DevOps. Εναλλακτικά, οι DevOps Coaches μπορούν να συμμετέχουν σε ομάδες DevOps και να συνεργάζονται με στελέχη για να λάβουν την απαιτούμενη υποστήριξη και να εξελίσσουν με συνέπεια άλλα μέρη οργανισμών για να ταιριάζουν στις ομάδες DevOps.

4.4 Δόμηση DevOps και Αρχιτεκτονικής

4.4.1 Δόμηση που Καθορίζει την Παράδοση Λογισμικού

Οι οργανισμοί που σχεδιάζουν συστήματα λογισμικού υποχρεούνται να παράγουν συστήματα που είναι αντίγραφα των δικών τους επικοινωνιακών δομών. Ο τρόπος δόμησης των ομάδων επηρεάζει την αρχιτεκτονική του λογισμικού και την πληροφορική.

Οι περισσότερες εταιρείες διαχωρίζουν τους οργανισμούς παράδοσης λογισμικού σε πολλές ομάδες και καταλήγουν να παράγουν το λογισμικό τους με τον ίδιο αριθμό επιπέδων. Τα ελεγχόμενα πειράματα έχουν επίσης αποδείξει ότι όταν ζητήθηκε από έναν οργανισμό 6 ομάδων να δημιουργήσουν ένα λογισμικό, οι ομάδες τους βρήκαν μια αρχιτεκτονική 6 επιπέδων. Όταν ένας άλλος οργανισμός με 3 ομάδες κλήθηκε να δημιουργήσει το ίδιο λογισμικό, βρήκε μια αρχιτεκτονική 3 επιπέδων. Στην πραγματικότητα, υπάρχουν πολλά παρόμοια παραδείγματα.

Ως παράδειγμα, μια διάσημη ασφαλιστική εταιρεία με 86000 υπαλλήλους σε όλο τον κόσμο δομήθηκε σε 2 μεγάλες ομάδες λειτουργιών για έναν από τους οργανισμούς πληροφορικής τους που παρέχουν το backend για τη διαχείριση των πελατών, των συμβολαίων, των τιμολογίων και των υπηρεσιών τους. Μία από αυτές τις ομάδες λειτουργιών ήταν προσανατολισμένη στη γλώσσα προγραμματισμού Java και η άλλη ομάδα λειτουργιών προσανατολίστηκε σε διαδικασίες αποθήκευσης PL/SQL. Επιπλέον, υπήρχαν εξαρτήσεις από λειτουργίες πληροφορικής σε επίπεδο οργανισμού και συγκεντρωτικές, βάσεων δεδομένων, υποδομών πληροφορικής, δικτύωσης, ασφάλειας και υπηρεσιών διασφάλισης ποιότητας που έκαναν τη ζωή τους ακόμα πιο δύσκολη για να μπορούν να παραδώσουν με ασφάλεια το λογισμικό τους. Η άλλη πρόκληση αυτής της οργάνωσης ήταν ότι αν και οι ομάδες ήταν ξεχωριστές, στην δική τους αρχιτεκτονική δεν μπορούσαν να διαχωρίσουν με ικανοποίηση τη λογική της εντοπισμένης στην Java από τη λογική πρόσβασης δεδομένων σε PL/SQL. Κάθε αίτημα αλλαγής και νέου χαρακτηριστικού απαιτούσε δεκάδες μεταβιβάσεις από τη μία ομάδα στην άλλη και είχαν ως αποτέλεσμα μεγάλες ουρές αναμονής και χρόνους παράδοσης για να γίνουν ακόμη και μικρές αλλαγές. Επιπλέον, σε αυτό το περίπλοκο περιβάλλον, ο αντίκτυπος ακόμη και των μικρών αλλαγών ήταν πολύ δύσκολο να προβλεφθεί, αφού έσπαγαν συνεχώς τα υπάρχοντα χαρακτηριστικά στην παραγωγή και προκάλεσαν διακοπές λειτουργίας που απαιτούσαν επίσης μεγάλους χρόνους παράδοσης, επιπλέον εργασία, κλιμακώσεις και άγχος για επίλυση. Για να αποκατασταθεί αυτό το ζήτημα, οι ομάδες λειτουργιών συγχωνεύθηκαν σε μία ομάδα προϊόντων. Σταδιακά επανασχεδιάστηκε το λογισμικό μετατρέποντας το επίπεδο πρόσβασης δεδομένων τους σε ένα σύνολο λειτουργιών API. Επιπλέον, δημιουργήθηκε ένα νέο επιχειρηματικό σύστημα πλήρως αποσυνδεδεμένο από την εσωτερική δυναμική του συστήματος πρόσβασης δεδομένων. Ακόμα και στο αρχικό στάδιο, αυτή η πρωτοβουλία βελτίωσε το ηθικό της ομάδας, επειδή ξεκίνησαν τόσο οι ειδικοί της Java όσο και των PL/SQL να εργάζονται για την επιτυχία της κοινής ομάδας προϊόντων αντί για τα διαφορετικά κίνητρα των προηγούμενων λειτουργικών σιλό. Καθώς δημιουργήθηκε μια χαλαρά συνδεδεμένη αρχιτεκτονική, ο αντίκτυπος των αλλαγών ήταν πιο εύκολο να εντοπιστεί, οι αλλαγές έγιναν ευκολότερες και γρηγορότερες στην εφαρμογή και τα ελαττώματα πιο εύκολο να εντοπιστούν και να διορθωθούν. Ως αποτέλεσμα, ο μέσος χρόνος παράδοσης των νέων λειτουργιών μειώθηκε από 4 μήνες σε 3 εβδομάδες και η ουρά συμβάντων της ομάδας είναι πλέον σχεδόν κενή, οπότε κερδίζουν από αυτό την ικανότητα αναθεώρησης και βελτίωσης της βάσης κώδικα.

4.4.2 Οργανισμοί Matrix και Λειτουργικές ομάδες

Οι εταιρείες είναι οργανωμένες με μια σειρά από λειτουργικές ομάδες γύρω από διάφορες γνώσεις σε θέματα όπως βάσεις δεδομένων, δικτύωση, λειτουργίες, ασφάλεια πληροφοριών, διασφάλιση ποιότητας, διαχείριση εκδόσεων, διαχείριση έργων, διαχείριση προϊόντων και ούτω καθεξής. Οι ομάδες του έργου κατασκευάζονται από άτομα από διαφορετικά λειτουργικά σιλό. Σε αυτούς τους οργανισμούς έργων Matrix που επιχειρούν να παραδώσουν προϊόντα και υπηρεσίες, οι διαχειριστές έργων είναι συνήθως υπεύθυνοι για την παράδοση και οι διαχειριστές προϊόντων είναι υπεύθυνοι για τη διασφάλιση της παράδοσης του σωστού παραδοτέου. Το πρόβλημά σε αυτήν την οργανωτική διαμόρφωση είναι ότι οι λειτουργικές ομάδες δεν έχουν κατανόηση της έκτασης της εργασίας που συνεισφέρουν. Σε ακραίες αλλά συχνά τυπικές περιπτώσεις, η λειτουργική ομάδα δεν ενδιαφέρεται ούτε για τη μεγάλη εικόνα, ούτε για τη συνολική απόδοση της πληροφορικής και την

επιχειρηματική υπόσταση του προϊόντος και της υπηρεσίας που συμβάλλουν. Δεδομένου ότι οι λειτουργικές ομάδες πρέπει συνήθως να διαχειρίζονται μεγάλες ουρές συμβάντων, συνήθως απαιτούν μεγάλους χρόνους παράδοσης για την υποστήριξη του έργου. Επειδή τα έργα πασχίζουν για λειτουργικούς πόρους, οι κλιμακώσεις είναι ο μόνος τρόπος για να ληφθεί γρήγορα προσοχή για το έργο. Η κλιμάκωση μολύνει προφανώς το εργασιακό κλίμα και την εμπιστοσύνη μεταξύ των ομάδων. Πολλαπλές μεταβιβάσεις από τη μία ομάδα στην άλλη, οι καθυστερήσεις, τα προβλήματα ποιότητας, οι ανακατασκευές, τα σημεία συμφόρησης και το άγχος αποτελούν πλέον μέρος της καθημερινής εργασίας. Αυτό επειδή η οργάνωση πρέπει να επικεντρώνεται σε μια αδιαφανή και ψεύτικη ψευδαίσθηση βελτιστοποίησης κόστους. Στην πραγματικότητα, λόγω ζητημάτων ποιότητας, ανακατασκευών και καθυστερήσεων, οι λειτουργικοί οργανισμοί είναι πιθανώς ακόμη πιο ακριβοί από οποιαδήποτε άλλη τυχαιά αναδιοργάνωση. Εκτός από αυτό, εάν ο οργανισμός έχει ήδη καταφέρει να αναθέσει ορισμένες από τις ζωτικές λειτουργικές δεξιότητές του, όπως η διασφάλιση ποιότητας και οι λειτουργίες πληροφορικής σε γεωγραφικά απομακρυσμένες τοποθεσίες, τότε ο οργανισμός πληροφορικής πρέπει να παραδώσει και να ικανοποιήσει με ασφάλεια απαιτήσεις που είναι κρίσιμες για τη συνέχεια της επιχείρησής.

4.4.3 Ομάδες Προσανατολισμένες σε Προϊόντα και Υπηρεσίες

Προκειμένου να επιλυθεί το πρόβλημα του παραπάνω κεφαλαίου, πρέπει να αλλάξει η ταχύτητα από την ψευδαίσθηση βελτιστοποίησης κόστους των λειτουργικών ομάδων στην έγκυρη και αποδεδειγμένη βελτιστοποίηση ταχύτητας του DevOps. Στην πραγματικότητα το DevOps θα σας επιτρέψει ούτως ή άλλως να εξοικονομήσετε κόστος ενώ η ομάδα θα παραδίδει πιο γρήγορα και συνεχώς. Για να επιτευχθεί αυτό πρέπει οι ομάδες να είναι προσανατολισμένες σε προϊόντα και υπηρεσίες. Οι ομάδες που προσανατολίζονται σε προϊόντα και υπηρεσίες είναι ανεξάρτητες, δια λειτουργικές, αυτόνομες, αυτοτελείς και συνήθως μικρές ομάδες που μπορούν να καλύψουν όλες τις φάσεις του κύκλου ζωής λογισμικού ενός δεδομένου προϊόντος και υπηρεσίας (αλλά όχι ενός προσωρινού έργου), συμπεριλαμβανομένης της αρχιτεκτονικής, του σχεδιασμού, της κωδικοποίησης, της δοκιμής, του πειραματισμού, της ανάπτυξης, της λειτουργίας αλλά και συντήρησης λογισμικού. Ως παράδειγμα, οι πιο επιτυχημένες ομάδες DevOps στην **Amazon** και το **Netflix** δεν εγκαταλείπουν τα προϊόντα και τις υπηρεσίες τους μόλις τα κυκλοφορήσουν και είναι υπεύθυνοι για τη λειτουργία και τη συντήρηση του λογισμικού στα δικά τους συστήματα παραγωγής.

4.4.4 Χρηματοδότηση Οργανισμού σε Προϊόντα και Υπηρεσίες

Για να βελτιώσει την επιχειρηματική απόδοση, ο οργανισμός δεν πρέπει να χρηματοδοτεί προσωρινά έργα των οποίων τα απτά επιχειρηματικά αποτελέσματα δεν είναι ασήμαντα για μακροπρόθεσμη αξιολόγηση. Με το DevOps, ο οργανισμός χρηματοδοτεί τη δική του αποστολή, τον δικό του σκοπό και τα δικά του προϊόντα και υπηρεσίες που σχετίζονται με αυτήν την αποστολή και σκοπό. Η επιτυχία των ομάδων αξιολογεί και αξιολογείται βάσει της απόδοσης της πληροφορικής και της επιχείρησής τους. Βασίζεται στην απόδοση της επένδυσης εντός των συγκεκριμένων επιχειρηματικών τομέων όπου εξυπηρετούν τα προϊόντα, τις υπηρεσίες και τις

μικροϋπηρεσίες τους στους εσωτερικούς ή και εξωτερικούς πελάτες. Οι ομάδες ενεργούν σαν ιδιοκτήτες προϊόντων και υπηρεσιών που δημιουργούν και παρέχουν, αντί να είναι απλά μέλη λειτουργικών σιλό που δεν δίνουν μεγάλη προσοχή στα επιχειρηματικά αποτελέσματα.

4.4.5 Δημιουργία ομάδας DevOps και Αρχιτεκτονικής Λογισμικού

Με την δημιουργία DevOps δεν προτείνεται η διακοπή και να αναδιοργάνωση σε όλα τα τρέχοντα έργα στον οργανισμό με μία κίνηση. Ένας μη-αποδιοργανωτικός, που επηρεάζει τον τρόπο προσαρμογής των ομάδων για τη μεθοδολογία DevOps είναι η εισαγωγή εποπτών λειτουργίας στις ομάδες έργων. Μόλις οι ομάδες αποκτήσουν επόπτες λειτουργίας στους επιθυμητούς τομείς, όπως διασφάλιση ποιότητας, ασφάλεια πληροφοριών, βάσεις δεδομένων, δικτύωση, διακομιστές, λειτουργίες κ.λπ.. τότε μπορούμε να περιμένουμε από αυτές τις ομάδες να σχεδιάσουν, να αναπτύξουν και να διατηρήσουν ανεξάρτητα και με αυτάρκεια τα προϊόντα λογισμικού και τις υπηρεσίες. Σε αυτές τις νέες ομάδες DevOps προσανατολισμένων σε προϊόντα και υπηρεσίες, η διαθεσιμότητα, η ποιότητα, η απόδοση, η ασφάλεια των πληροφοριών και η συμμόρφωση αποτελούν καθημερινή δουλειά όλων. Οι ομάδες πρέπει να δώσουν προσοχή σε αυτές τις μη λειτουργικές απαιτήσεις μέχρι να ολοκληρώσουν το σχεδιασμό και την ανάπτυξη τους. Οι εξωτερικοί επόπτες πρέπει να κρίνουν και να επικυρώσουν την ασφάλεια και την ποιότητα των εφαρμογών λογισμικού χωρίς να εμπλακούν σε οποιοδήποτε στάδιο τεχνικής λογισμικού των προϊόντων και των υπηρεσιών. Για αυτό το λόγο οι ομάδες υψηλής απόδοσης DevOps βασίζονται σε εξωτερικούς επόπτες θεμάτων μόνο για να λάβουν συμβουλές, αλλά εξακολουθούν να κατέχουν πλήρως όλες τις μη λειτουργικές απαιτήσεις σε κάθε στάδιο του κύκλου ζωής της μηχανικής λογισμικού.

Ένας εξαιρετικά επιτυχημένος οργανισμός με DevOps που εξακολουθεί να χρησιμοποιεί σε μεγάλο βαθμό λειτουργικές ομάδες σε συνδυασμό με ομάδες DevOps προσανατολισμένες σε προϊόντα και υπηρεσίες είναι η **Google**. Στην Google οι ομάδες DevOps και οι ομάδες λειτουργιών βλέπουν την οργανωτική τους αποστολή με τα προϊόντα και τις υπηρεσίες τους ως κοινό στόχο. Οι ομάδες λειτουργιών δημιουργούν σχεδιαγράμματα όπως πρότυπα, λίστες ελέγχου, σκελετούς εφαρμογών κ.λπ. και συστήματα αυτοεξυπηρέτησης, για παράδειγμα ανεξάρτητη παροχή και ρύθμιση παραμέτρων διακομιστών και δικτύωσης για την βεβαίωση ότι δεν υπάρχουν ποτέ σημεία συμφόρησης που επιβραδύνουν το προϊόν και την υπηρεσία της ομάδας DevOps. Οι ομάδες λειτουργιών της Google συνεργάζονται και υποστηρίζουν ομάδες προϊόντων και υπηρεσιών σε πρώιμα στάδια του κύκλου ζωής της μηχανικής λογισμικού, πραγματοποιούν εποικοδομητικά σχόλια για σχέδια, κώδικες και δοκιμές και παρέχουν συνεχώς συμβουλές σε ομάδες προϊόντων και υπηρεσιών DevOps.

4.4.6 Μέγεθος Ομάδας DevOps

Το ιδανικό μέγεθος για μια ομάδα DevOps είναι από 5 έως 10 άτομα. Ένα τόσο περιορισμένο μέγεθος ομάδας μειώνει την πολυπλοκότητα της επικοινωνίας και της εύρυθμης λειτουργίας εντός της ομάδας. Επιπλέον, ο επικεφαλής της ομάδας και τα μέλη της ομάδας δεν σπαταλούν πολύ χρόνο με επιπλέον καθήκοντα. Αυτό διατηρεί επίσης το μέγεθος του προϊόντος και της υπηρεσίας που η

ομάδα είναι υπεύθυνη για ένα συγκεκριμένο όριο, το οποίο μειώνει περαιτέρω την πολυπλοκότητα, τη συντήρηση και τη δυσκολία λειτουργίας των εφαρμογών λογισμικού. Κάθε μέλος της ομάδας σε τόσο μικρές ομάδες βλέπει τη μεγάλη εικόνα και όλοι συγκεντρώνουν ηγετική εμπειρία με το να γίνουν μέρος μιας κρίσιμης αποστολής για την οργάνωσή τους. Ο επικεφαλής της ομάδας συνεργάζεται με την ανώτερη διοίκηση για να κατανοήσει τους στόχους και να τους μεταφράσει στα μέλη της ομάδας σας. Σε περίπτωση που το προϊόν και η υπηρεσία δεν μπορούν πλέον να χειριστούν από μια ομάδα έως 10 ατόμων, δημιουργούμε ένα νέο προϊόν και υπηρεσία και στελεχώνουμε μια νέα ομάδα DevOps που τα αναλαμβάνει. Οι έννοιες προϊόντων και υπηρεσιών δεν είναι μόνο οντότητες που εξυπηρετούνται και παρέχονται σε εξωτερικούς πελάτες αλλά επίσης εσωτερικά προϊόντα, υπηρεσίες ή τα λεγόμενα "API μικροϋπηρεσιών" με τις αντίστοιχες ομάδες DevOps για τους εσωτερικούς πελάτες. Για παράδειγμα, εάν το σύστημα λογιστηρίου γίνει πολύ μεγάλο για μια ομάδα έως 10 ατόμων, τότε θα πρέπει να δημιουργηθεί μια άλλη ομάδα DevOps που αναλαμβάνει το API πρόσβασης βάσης δεδομένων. Μπορεί άλλη μια να αναλάβει το διαδικτυακό API πληρωμών. Φυσικά, όλες αυτές οι ομάδες θα πρέπει να χρησιμοποιούν ένα κοινό αποθετήριο κώδικα και ένα κοινό αγωγό ανάπτυξης για τη διασφάλιση συνεχούς ενσωμάτωσης και συνεχούς παράδοσης.

Σε μια στενά συνδεδεμένη αρχιτεκτονική, μικρές αλλαγές σε μία εφαρμογή μπορεί τελικά να προκαλέσουν πολλές δυσμενείς επιπτώσεις σε πολλές ροές εργασίας. Επομένως, τα προϊόντα, οι υπηρεσίες και τα API μικροϋπηρεσιών στην αρχιτεκτονική σας πρέπει να συνδέονται χαλαρά. Κάθε ομάδα DevOps πρέπει να είναι υπεύθυνη μόνο για ένα κομμάτι μιας αρχιτεκτονικής χαλαρής ζεύξης. Κάθε ομάδα DevOps μπορεί ανεξάρτητα να σχεδιάσει, να αναπτύξει το λογισμικό. Ο μηχανισμός έγκαιρης προειδοποίησης που είναι ενσωματωμένος στον αγωγό ανάπτυξης θα πρέπει να ενημερώνει αυτόματα και γρήγορα τις ομάδες DevOps σχετικά με τις πιθανές δυσμενείς επιπτώσεις που προκαλεί ο έλεγχος κώδικα. Για να μπορούν τέτοιες μικρές ομάδες να χειριστούν με αυτάρκεια όλες τις φάσεις του κύκλου ζωής του λογισμικού, από την αρχιτεκτονική έως τις λειτουργίες από μόνες τους, θα πρέπει να ενθαρρύνουμε όλους στην ομάδα να γίνουν [DevOps Generalists](#). Πρέπει να τους ενθαρρύνουμε και να τους επιτρέψουμε να δημιουργούν συνεχώς νέες δεξιότητες. Θα πρέπει να προσλάβουμε μόνο μέλη της ομάδας που είναι πρόθυμα να μάθουν και να αναπτυχθούν ανεξάρτητα από το επίπεδο τεχνογνωσίας και εμπειρίας. Πρέπει να αποφευχθούν αυστηρά τα άτομα που περιμένουν να αξιολογηθούν σε ένα καθορισμένο σύνολο ρόλων και ευθυνών.

4.5 Στελέχωση και Ανθρώπινο Δυναμικό για DevOps

Για την απόκτηση και διατήρηση ικανών εργαζομένων για την θέση DevOps, η πολυεθνική για την οποία έγινε αναφορά στην εισαγωγή της διπλωματικής έπρεπε να προχωρήσει στις παρακάτω στρατηγικές και λειτουργικές ενέργειες^[24]:

- Προγραμματισμός του ανθρώπινου δυναμικού: Μία ενέργεια στρατηγικού χαρακτήρα με σκοπό την πρόβλεψη των αναγκών για τις μελλοντικές θέσεις DevOps.
- Ανάλυση εργασίας: Περιγραφή του έργου αλλά και των τυπικών και ουσιαστικών προσόντων των φορέων για την θέση DevOps.

- Προσέλκυση υποψηφίων: Εντοπισμός δυνητικών υποψηφίων, πρόσκληση ενδιαφέροντος, επιλογή καταλληλότερων, υποδοχή και ένταξη στον οργανισμό νέων προσλήψεων.
- Εκπαίδευση και ανάπτυξη του ανθρώπινου δυναμικού: Περιλαμβάνει την εκπαίδευση και κατάρτιση, την ανάθεση διευρυμένων καθηκόντων και την παροχή δυνατοτήτων προσωπικής ανάπτυξης.
- Ανταμοιβή εργαζομένων: Αναφέρεται στο σχεδιασμό και στην υλοποίηση μιας πολιτικής αμοιβών και ειδικών σχεδίων μισθών, ημερομισθίων, οικονομικών κινήτρων και παροχών.

4.5.1 Στρατηγικός Προγραμματισμός Ανθρώπινων Πόρων

Ο Στρατηγικός Προγραμματισμός Ανθρώπινων Πόρων διασφαλίζει ότι το δυναμικό που διαθέτει ο οργανισμός ή που θα αποκτηθεί στο μέλλον θα είναι διαθέσιμο και ικανό να υλοποιήσει τα στρατηγικά σχέδια και να επιτύχει τους στόχους. Στον στρατηγικό προγραμματισμό περιλαμβάνονται τα εξής βήματα:

- Ανάλυση εξωτερικού περιβάλλοντος για την εκτίμηση της εξέλιξης στην αγορά εργασίας.
- Ανάλυση εσωτερικού περιβάλλοντος για την αποτύπωση σε ποιοτικό αλλά και ποσοτικό επίπεδο του υφιστάμενου ανθρωπίνου δυναμικού.
- Πρόβλεψη ζήτησης και προσφοράς.
- Σύγκριση ζήτησης και προσφοράς.
- Στόχοι και σχέδια δράσης.

Σημαντικό και αναπόσπαστο κομμάτι είναι και ο κύκλος ζωής του οργανισμού καθώς και η οργανωτική κουλτούρα του. Ανάλογα με το κύκλο ζωής και το σημείο στο οποίο βρισκόμαστε, αν για παράδειγμα έχουμε μια νεοσύστατη εταιρία ή μια αναπτυσσόμενη ή μια ώριμη, αντίστοιχα προσαρμόζουμε και τους στόχους που αφορούν το ανθρώπινο δυναμικό.

Στην περίπτωση μας και όπως είχε γίνει αναφορά στην εισαγωγή της διπλωματικής, ο στρατηγικός προγραμματισμός εφαρμόστηκε για την δημιουργία νεοσύστατου τμήματος DevOps σε ώριμο οργανισμό, που κρίθηκε αναγκαίο έπειτα από την στρατηγική απόφαση για υιοθέτηση σε νέες μεθοδολογίες για την ανάπτυξη λογισμικού, όπως αυτή του Agile.

Για τους εξωτερικούς παράγοντες δεν έγινε ιδιαίτερη ανάλυση καθώς οι μηχανικοί DevOps ακολουθούν τα ίδια μοτίβα με τους υπόλοιπους μηχανικούς πληροφορικής. Παρόλα αυτά κατά το στάδιο της προσέλκυσης υποψηφίων έγινε αναθεώρηση για την πολιτική προσέλκυσης και συγκεκριμένα το πακέτο αποδοχών καθώς οι προσφορές των ανταγωνιστών ήταν ιδιαίτερα υψηλές σε σχέση με την προϋπηρεσία και τα προσόντα των υποψηφίων.

Ανάλυση εσωτερικού περιβάλλοντος υλοποιήθηκε και κρίθηκε απαραίτητη η σύσταση νέου τμήματος καθώς εργασίες αυτοματοποίησης, συνεχούς παράδοσης/συνεχούς ενσωμάτωσης και γενικότερες εργασίες DevOps ήταν κατανεμημένες στον οργανισμό σε ρόλους όπως των προγραμματιστών, των αρχιτεκτόνων αλλά και του τμήματος λειτουργιών. Εδώ υπήρχε ένα ζήτημα καθώς αυτές οι ομάδες/ρόλοι δεν είχαν κοινούς στόχους και ήταν χρονοβόρα η παράδοση του τελικού προϊόντος. Βάση της εσωτερικής ανάλυσης έτρεξε και μια καμπάνια ενημέρωσης εσωτερικά όλων των μηχανικών για πιθανή μετάβαση και αλλαγή μονοπατιού καριέρας σε DevOps.

Στην πρόβλεψη ζήτησης και προσφοράς αλλά και την σύγκριση κατά την διάρκεια του στρατηγικού προγραμματισμού, και με βάση των αναγκών των εκάστοτε έργων, είδαμε ότι η

ζήτηση ήταν συνεχώς αυξανόμενη (νέα έργα προστέθηκαν κατά την διάρκεια σύστασης του τμήματος) αλλά στον αντίποδα παρόλο που η προσφορά ήταν ικανοποιητική, δεν υπήρχε ανταπόκριση από τους ενδιαφερόμενους καθώς το πακέτο αποδοχών με βάση τα εσωτερικά κριτήρια αξιολόγησης ήταν χαμηλά. Αυτό είχε ως αποτέλεσμα την απόρριψη από τους ίδιους τους υποψήφιους στην προσφορά εργασίας.

Στόχοι και σχέδια δράσης μετά την πρώτη προσπάθεια για την κάλυψη των θέσεων τέθηκαν ξανά για συζήτηση. Αναθεωρήθηκαν τα πακέτα αποδοχών καθώς και αναγνωρίστηκε η ανάγκη για την εκπαίδευση νέων μηχανικών με ελάχιστη, έως καθόλου, προϋπηρεσία.

4.5.2 Ανάλυση Εργασίας

Χρειάστηκε να περάσουν αρκετοί μήνες καθώς σε υψηλότερο επίπεδο υπήρχαν τα καθήκοντα και οι υποχρεώσεις, οι δεξιότητες και οι ευθύνες των DevOps αλλά δεν ήταν ξεκάθαρο καθώς αναλάμβαναν εργασίες που μέχρι πρότινος τις υλοποιούσαν άλλα τμήματα/ρόλοι. Πολύ σημαντικό στοιχείο ήταν η αρνητική στάση των εργαζομένων στις αλλαγές. Με την προσθήκη DevOps, πολλές εργασίες μεμονωμένων μηχανικών πέρασαν στην ομάδα των DevOps με αποτέλεσμα τον επαναπροσδιορισμό των καθηκόντων.

Για την ανάλυση της εργασίας χρησιμοποιήθηκε η έννοια του Χαρτοφυλακίου το οποίο έγινε με λεπτομερή καταγραφή ανά έργο και αποτυπώθηκε στο Confluence. Επίσης για την καλύτερη κατανόηση του επιπέδου των μηχανικών δημιουργήθηκε ένας πίνακας με όλους τους μηχανικούς και όλες τις τεχνολογίες/εργαλεία. Σε αυτό τον πίνακα υπάρχει βαθμονόμηση από το 0 έως το 5 για το πόσο καταρτισμένοι είναι οι μηχανικοί και σύμφωνα με τις ανάγκες των έργων αλλά και τις νέες τάσεις οργανώνονται επιμορφωτικά σεμινάρια και πιστοποιήσεις.

Οι μέθοδοι που χρησιμοποιήθηκαν για την ανάλυση της εργασίας ήταν η παρατήρηση και το ημερολόγιο εργαζομένου. Και οι δυο μέθοδοι είναι χρονοβόρες και επιβαρύνουν τα μέλη της ομάδας. Για την περιγραφή και προδιαγραφή της θέσης εκτός από τις βασικές αρχές που πρέπει να διέπουν τους μηχανικούς πληροφορικής προστέθηκαν και τα εργαλεία και οι μεθοδολογίες.

4.5.3 Προσέλκυση Υποψηφίων

Με την ολοκλήρωση της ανάλυσης εργασίας και την συλλογή των απαραίτητων στοιχείων γίνεται η γνωστοποίηση της θέσης για εξωτερική προσέλκυση. Εκτός από την σελίδα της εταιρίας, γίνεται και ανάρτηση της αγγελίας σε συνεργαζόμενες διαδικτυακές εταιρίες εύρεσης προσωπικού. Η συλλογή των υποψηφίων γίνεται αυτόματα από λογισμικό “Luceo” που υποστηρίζει την διαδικασία στελέχωσης με σύνδεσμο που βρίσκεται στην αγγελία. Οι εμπλεκόμενοι για την πρόσληψη λαμβάνουν ειδοποίηση για να προχωρήσουν σε περαιτέρω έλεγχο και διαχωρισμό των βιογραφικών για τα επόμενα στάδια. Στο ίδιο σύστημα μπαίνουν και υποψήφιοι από εσωτερικές συστάσεις.

Εκτός από την γνωστοποίηση της θέσης, υπάρχουν και οι ημέρες καριέρας από διάφορους φορείς στις οποίες οι ενδιαφερόμενοι ενημερώνονται για τις ανοιχτές θέσεις. Σε περίπτωση ενδιαφέροντος τα στοιχεία μπαίνουν στο ίδιο λογισμικό ώστε να συνεχίσει η διαδικασία της στρατολόγησης. Τέλος υπάρχουν και “κυνηγοί κεφαλών” οι οποίοι προσεγγίζουν διάφορους υποψήφιους βάση του προφίλ τους (για παράδειγμα μέσω **LinkedIn**) ώστε να τους περιγράψουν την θέση εργασίας και να κεντρίσουν το ενδιαφέρον για πιθανή συνεργασία.

4.5.4 Επιλογή Υποψηφίων

Με την διαδικασία συγκέντρωσης πληροφοριών και αξιολόγησης για τους υποψηφίους γίνεται η επιλογή των πιο κατάλληλων για τις θέσεις. Είναι πολύ σημαντικό το άτομο που θα επιλεγεί να είναι το κατάλληλο και να ταιριάζει στην θέση έτσι ώστε να υπάρχουν λιγότερες πιθανότητες να εγκαταλείψει την θέση έπειτα από σύντομο χρονικό διάστημα. Κατά την διάρκεια των συνεντεύξεων πρέπει να γίνεται σαφής περιγραφή του ρόλου καθώς και οι προοπτικές εξέλιξης.

Στην διαδικασία επιλογής υπάρχει μια προκαταρκτική εξέταση-συνέντευξη (screening) κατά την οποία έπειτα από εξέταση των βιογραφικών γίνεται η πρώτη συνέντευξη με υποψήφιους που πληρούν τις ελάχιστες απαιτήσεις. Έπειτα, όσοι πληρούν τις προδιαγραφές θα κληθούν για την τεχνική συνέντευξη κατά την οποία ο Resource Manager με τον Technical Lead, αφού περιγράψουν το τεχνικό πλαίσιο και τις αρμοδιότητες του τμήματος DevOps, θέτουν ερωτήσεις πάνω στις τεχνολογίες στις οποίες ο υποψήφιος έχει γνώση. Επειδή υπάρχει μια μεγάλη γκάμα από τεχνολογικές απαιτήσεις στο DevOps η προσέγγιση είναι να βρεθούν προφίλ που έχουν τις βασικές απαιτήσεις και έπειτα από εκπαίδευση να αναπτύξουν και τις υπόλοιπες.

Έπειτα από την τεχνική συνέντευξη, το τμήμα ανθρώπινου δυναμικού αναλύει την αξιολόγηση βάση της οποίας κατατάσσει και τον ενδιαφερόμενο στην αντίστοιχη εσωτερική βαθμίδα ώστε να προχωρήσει στην προσφορά με το πακέτο αποδοχών με μισθό και τυχόν πρόσθετες παροχές. Η τελική επιλογή είναι αποκλειστικά στην απόφαση του υποψηφίου για την αποδοχή ή όχι της προσφοράς για τη θέση.

Στο πλαίσιο της υποδοχής και ένταξης των νέων προσλήψεων DevOps γίνεται η ανάθεση τους σε έναν συνάδελφο, συνήθως με καλή γνώση του αντικειμένου και των διαδικασιών, ο οποίος αναλαμβάνει ως “Intra-buddy” την καθοδήγηση και την ομαλή ένταξη στην ομάδα. Παράλληλα γίνονται ανά τακτά χρονικά διαστήματα εκπαιδεύσεις από το τμήμα ανθρωπίνου δυναμικού με στόχο την ενημέρωση για τους στόχους της εταιρίας, τις αξίες καθώς και διαδικασίες για την ομαλή λειτουργία.

4.5.5 Εκπαίδευση και Ανάπτυξη Ανθρωπίνων Πόρων

Η εκπαίδευση και ανάπτυξη δεν αφορά μόνο τους νέους αλλά και όσους εργάζονται ήδη. Οι περιοχές στις οποίες εστιάζουμε για την εκπαίδευση και ανάπτυξη μηχανικών DevOps:

- Βελτίωση απόδοσης
- Αναβάθμιση ικανοτήτων
- Ικανοποίηση της ανάγκης για προσωπική ανάπτυξη

Χρειάζεται καθορισμός των σταδίων της εκπαιδευτικής διαδικασίας όπως η εκτίμηση εκπαιδευτικών αναγκών, ο σχεδιασμός εκπαιδευτικών προγραμμάτων, η υλοποίηση εκπαιδευτικών προγραμμάτων και η αξιολόγηση του εκπαιδευτικού έργου.

Το σύνολο των ενεργειών και των μεθόδων που εφαρμόζει η επιχείρηση για να αποκτήσουν τα στελέχη της γνώσεις και ικανότητες που θα τις χρειαστούν για την αποτελεσματικότερη άσκηση των καθηκόντων τους περιλαμβάνει μεθόδους ανάπτυξης στη θέση εργασίας και μεθόδους ανάπτυξης εκτός θέσης εργασίας. Για την εκπαίδευση δίνονται παραδείγματα και εργαλεία με μονοπάτια εκπαίδευσης για διαφορετικούς τύπους εργαζομένων σε επόμενο κεφάλαιο.

4.5.6 Αξιολόγηση Απόδοσης

Αξιολόγηση απόδοσης είναι η διαδικασία με την οποία υπολογίζεται το επίπεδο του έργου που εκτελεί ο κάθε εργαζόμενος, σύμφωνα με συγκεκριμένα κριτήρια που έχουν καθορισθεί εκ των προτέρων. Στοχεύει στην συγκέντρωση στοιχείων καθώς και την ενημέρωση των αξιολογημένων για λήψη αποφάσεων για τυχόν βελτιώσεις.

Οι μέθοδοι αξιολόγησης της απόδοσης:

- Μέθοδοι με βάση τα χαρακτηριστικά που πρέπει να έχει κανείς για να αποδώσει ικανοποιητικά στην εργασία του. Αδυναμία της μεθόδου είναι ότι λόγω της μεγάλης υποκειμενικότητας, μπορεί να αξιολογούνται τα χαρακτηριστικά, αλλά όχι συμπεριφορά.
- Μέθοδοι με βάση τη συμπεριφορά. Αφού έχουν οριστεί οι απαιτούμενες κρίσιμες συμπεριφορές για κάθε θέση εργασίας, μετριέται σε τι βαθμό τις επιδεικνύει ο εργαζόμενος. Αυτή η μέθοδος είναι πιο κατάλληλη για εργασίες που στηρίζονται κατά πολύ στη συμπεριφορά, όπως για παράδειγμα στον τομέα των υπηρεσιών.
- Μέθοδοι με βάση τα αποτελέσματα που επιτυγχάνει ο εργαζόμενος. Βασίζονται σε ποσοτικά δεδομένα και είναι πιο αντικειμενικές. Μειονεκτήματά τους είναι ότι δεν λαμβάνουν υπόψη τους παράγοντες της κατάστασης που δεν ελέγχει ο ίδιος ο εργαζόμενος, δεν βοηθούν στον εντοπισμό των αιτιών της χαμηλής απόδοσης.

4.5.7 Πολιτική και Συστήματα Αμοιβής

Η αμοιβή μπορεί να είναι άμεση οικονομική παροχή (μισθοδοσία) ή έμμεση οικονομική παροχή (πρόσθετες παροχές όπως ιατροφαρμακευτική ασφάλεια).

Σύμφωνα με την πολιτική της εταιρίας πρέπει να επιτευχθούν τα παρακάτω με ένα αποτελεσματικό σύστημα αμοιβών:

- Απόκτηση Μηχανικών DevOps με προσόντα
- Διατήρηση των υφιστάμενων μηχανικών DevOps
- Εξασφάλιση ισότητας (μέσω των εσωτερικών βαθμίδων κατάταξης)
- Ανταμοιβή της εξέχουσας συμπεριφοράς
- Έλεγχος του κόστους
- Συμμόρφωση με κανονισμούς
- Διαμόρφωση εταιρικής κουλτούρας
- Καλές εργασιακές σχέσεις

Με δεδομένο τον ανταγωνισμό και την ανάγκη για αύξηση της ανταγωνιστικότητας, είναι απαραίτητο να υπάρχει συγκεκριμένη και σαφή πολιτική και συστήματα αμοιβών, ώστε, να επιτυγχάνει την καλύτερη δυνατή εργασιακή απόδοση και έλεγχο του κόστους.

Στην περίπτωση μας και βάση της έρευνας αγοράς έγινε επαναπροσδιορισμός των αμοιβών καθώς υπήρχε μεγάλος αριθμός υποψηφίων που απέρριπταν την προσφορά για κάποιο ανταγωνιστή με υψηλότερο πακέτο αποδοχών. Στην ανάλυση της εργασίας επαναπροσδιορίστηκαν οι αμοιβές για τις βαθμίδες μηχανικών Junior, Mid και Senior DevOps.

4.6 Μονοπάτια εκπαίδευσης DevOps

Στο συγκεκριμένο κεφάλαιο θα δοθούν ενδεικτικά τρία μονοπάτια εκπαίδευσης DevOps για Αρχάριους, Ενδιάμεσους αλλά και Προχωρημένους Μηχανικούς με ξεχωριστές ενότητες για το κάθε ένα από αυτά.

4.6.1 DevOps Beginner (Αρχάριος DevOps)

Οι ενότητες που περιλαμβάνει αυτό το μονοπάτι δίνονται παρακάτω με μια σύντομη περιγραφή για το περιεχόμενο της κάθε ενότητας:

4.6.1.1 DevOps Concepts

Σε αυτό το μάθημα, παρέχεται μια επισκόπηση του DevOps, εστιάζοντας στην βασική αξία του CAMS (κουλτούρα, αυτοματισμός, μέτρηση και κοινή χρήση). Καλύπτει τις διάφορες μεθοδολογίες και εργαλεία που ένας οργανισμός μπορεί να υιοθετήσει για μετάβαση σε DevOps, εξετάζοντας τόσο ευέλικτες όσο και αδύνατες αρχές διαχείρισης έργων και πώς οι παλιές αρχές όπως το ITIL, το ITSM και το SDLC ταιριάζουν στα DevOps. Το μάθημα ολοκληρώνεται με τους τρεις κύριους πυλώνες του DevOps - αυτοματισμός υποδομής, συνεχή παράδοση και μηχανικούς αξιοπιστίας - καθώς και ορισμένους πρόσθετους πόρους και μια σύντομη ματιά στο τι έχει το μέλλον καθώς οι οργανισμοί μεταβαίνουν από το cloud σε αρχιτεκτονικές χωρίς διακομιστές.

Στη συνέχεια αναλύονται οι βασικές αρχές της ευέλικτης μεθοδολογίας, καθώς και γιατί αυτή η προσέγγιση είναι μοναδική για τους μηχανικούς παραγωγής λογισμικού. Έπειτα, εξετάζεται το Scrum - ένα εξαιρετικά δημοφιλές ευέλικτο πλαίσιο – οι αρχές και οι ρόλοι. Διαχωρίζει αλλά και συμπληρώνει την εφαρμογή Scrum με το Kanban.

4.6.1.2 Έλεγχος Έκδοσης

Αυτό το μάθημα είναι μια πύλη για τον έλεγχο εκδόσεων λογισμικού εκμάθησης, τη διαχείριση διαδικασιών και τις τεχνικές συνεργασίας. Εξετάζει την ιστορία του ελέγχου έκδοσης και καταδεικνύει τις θεμελιώδεις έννοιες: check-in/ check-out, forking, συγχώνευση, δεσμεύσεις και διανομή. Η επιλογή ενός συστήματος ελέγχου έκδοσης είναι ζωτικής σημασίας για την αποτελεσματική διαχείριση και εκδοχή των στοιχείων σε ένα έργο ανάπτυξης λογισμικού (από πηγαίο κώδικα, εικόνες και μεταγλωττισμένα δυαδικά έως πακέτα εγκατάστασης), έτσι το μάθημα εξετάζει επίσης τις διαθέσιμες λύσεις. Εξετάζει συγκεκριμένα τους **Git**, **Perforce**, **Subversion**, **Mercurial** και **Microsoft Team Foundation Server (TFS)**, περιγράφοντας την κατάλληλη χρήση, τις δυνατότητες, τα οφέλη και το βέλτιστο μέγεθος ομάδας για το κάθε ένα.

4.6.1.3 CI/CD

Οι πρακτικές συνεχούς ενσωμάτωσης και συνεχούς παράδοσης (CI/CD) επιτρέπουν στους προγραμματιστές να παράγουν εφαρμογές με ταχύτερο ρυθμό και να ενισχύσουν την ομαδική συνεργασία στη διαδικασία. Καθώς το CI/CD μεγάλωσε για να γίνει βασικός τομέας εξάσκησης DevOps, νέα εργαλεία CI/CD άρχισαν να πολλαπλασιάζονται σε ολόκληρο τον κλάδο. Σε αυτό το μάθημα, δίνεται μια ποικιλία εργαλείων CI/CD σε διάφορες κατηγορίες, επισημαίνοντας τα

πλεονεκτήματα και τα μειονεκτήματα του καθενός: **Jenkins** και **Bamboo**, εργαλεία **SaaS** και επιλογές από παρόχους υπηρεσιών Cloud και αποθετήρια κώδικα.

4.6.1.4 Λειτουργικά Συστήματα

Η γνώση της γραμμής εντολών Linux είναι κρίσιμη για όποιον χρησιμοποιεί αυτό το λειτουργικό σύστημα ανοιχτού κώδικα και ιδιαίτερα για τους μηχανικούς DevOps. Για πολλές εργασίες, είναι πιο αποτελεσματικό και ευέλικτο από ένα γραφικό περιβάλλον. Για τους διαχειριστές, παίζει ζωτικό ρόλο στη διαμόρφωση των αδειών και στην εργασία με αρχεία. Σε αυτό το μάθημα, θα δοθούν τα βασικά στοιχεία της συνεργασίας με τη γραμμή εντολών Linux χρησιμοποιώντας το κέλυφος Bash, εστιάζοντας σε πρακτικές εντολές Linux με παραδείγματα που βοηθούν στην περιήγηση στη δομή αρχείων και φακέλων, επεξεργασία κείμενου και ορισμό δικαιωμάτων. Αναλύονται επίσης μερικά από τα κοινά εργαλεία γραμμής εντολών, όπως grep, awk και sed, και εφαρμογές γραμμής εντολών όπως οι συντάκτες κειμένου nano και Vim. Το μάθημα ολοκληρώνεται με μια ματιά στον τρόπο εγκατάστασης και ενημέρωσης λογισμικού με έναν διαχειριστή πακέτων.

4.6.1.5 Containers

Το **Docker** είναι το επόμενο βήμα πέρα από την εικονικοποίηση. Μια εικόνα Docker περιέχει όλα όσα χρειάζεται για να εκτελεστεί, ανεξάρτητα από τον διακομιστή Linux στον οποίο βρίσκονται ένα αντίγραφο του λειτουργικού συστήματος, μια βάση δεδομένων, κώδικας, αρχεία διαμόρφωσης, εξαρτήσεις και ούτω καθεξής. Οι εικόνες μπορούν επίσης να γίνουν πακέτα και να κοινοποιηθούν σε άλλους διαχειριστές του Docker. Το Docker χρησιμοποιείται για να τρέχει σύνθετα συστήματα με εκατομμύρια χρήστες και εκατοντάδες containers. Εισάγονται έννοιες για τα βασικά στοιχεία του Docker, συμπεριλαμβανομένων των containers, τα Dockerfiles και των δυνατοτήτων δικτύωσης, διαχείρισης δεδομένων και βελτιστοποίησης υποδομής.

Ο Κυβερνήτης (**Kubernetes**) είναι η πιο δημοφιλής μηχανή ενορχήστρωσης ανοιχτού κώδικα στον κόσμο. Προσφέρει τη δυνατότητα προγραμματισμού και διαχείρισης container Docker. Αυτό το μάθημα εισάγει προγραμματιστές, μηχανικούς DevOps και επαγγελματίες πληροφορικής στο Kubernetes. Αρχικά, μαθαίνουμε πώς μπορεί να ενεργοποιηθεί το περιβάλλον Kubernetes σε Mac ή Windows χρησιμοποιώντας το **Minikube** και κατανοούμε τα στοιχεία του Kubernetes. Στη συνέχεια, αναπτύσσεται ένα δείγμα εφαρμογής Kubernetes και διαχείριση με τον πίνακα ελέγχου Kubernetes. Επίσης μαθαίνουμε πώς να αναπτύξουμε μια πιο περίπλοκη εφαρμογή με μια βάση δεδομένων και API. Στη συνέχεια, μαθαίνουμε πώς να εκτελούμε εργασίες cron. Τέλος, εξερευνούμε πιο προηγμένα θέματα, όπως αναπτύξεις παραγωγής, χώρους ονομάτων, παρακολούθηση και καταγραφή, καθώς και έλεγχο ταυτότητας και εξουσιοδότηση.

4.6.1.6 Application Servers

Επιλέγουμε τον κατάλληλο διακομιστή για την εφαρμογή Java EE και μαθαίνουμε τα πλεονεκτήματα και τα μειονεκτήματα ορισμένων από τις πιο δημοφιλείς επιλογές. Σε αυτό το μάθημα, συγκρίνονται και αντιπαραβάλλονται τα τέσσερα κορυφαία: **Apache Tomcat**, **Oracle Weblogic**, **WildFly** και **Payara**, έναν διακομιστή εφαρμογών που προέρχεται από το **Glassfish**. Μαθαίνουμε πώς μπορούμε να εγκαταστήσουμε κάθε λύση, να αναπτύξουμε τα αρχεία WAR, να

προετοιμάσουμε και να προσθέσουμε προγράμματα οδήγησης βάσης δεδομένων και να ρυθμίσουμε τη δημιουργία συμπλέγματος για εφαρμογές Java EE που εκτελούνται σε κοντέινερ.

4.6.1.7 *Cloud DevOps*

Με την μετακίνηση του DevOps στο Cloud μπορούμε να γίνουμε πιο ευέλικτοι στην ανάπτυξη και τη λειτουργία λογισμικού. Αυτό το μάθημα εξηγεί πώς να ρυθμίσουμε μια διαδικασία DevOps που βασίζεται σε Cloud και να αξιοποιήσουμε υπηρεσίες που προσφέρονται από την **Amazon** και τη **Microsoft**. Εξετάζονται λύσεις DevOps που προσφέρονται στο **Amazon Web Services** και στο **Microsoft Azure**. Επιπλέον, ανατρέχει σε ορισμένες περιπτώσεις χρήσης που επιδεικνύουν πραγματικές εφαρμογές υλοποίησης υπηρεσιών DevOps στο Cloud.

4.6.2 *DevOps Intermediate (Ενδιάμεσος DevOps)*

Οι ενότητες που περιλαμβάνει αυτό το μονοπάτι δίνονται παρακάτω με μια σύντομη περιγραφή για το περιεχόμενο της κάθε ενότητας:

4.6.2.1 *Scripting*

Οι τακτικές εκφράσεις, ή Regexes, μπορούν να βοηθήσουν να επιταχύνουμε τις κουραστικές εργασίες και να βελτιώσει τη ροή εργασίας. Το **Regexes** μπορεί να χρησιμοποιηθεί στις περισσότερες σύγχρονες γλώσσες προγραμματισμού, όπως **Java**, **JavaScript**, **Perl**, **Python**, **ActionScript** και **C#**. Σε αυτό το μάθημα, μαθαίνουμε πώς να χρησιμοποιούμε Regexes για να ορίσουμε σύνθετα μοτίβα κειμένου που μπορούν να χρησιμοποιηθούν για ισχυρές τεχνικές αναζήτησης και αντικατάστασης. Καλύπτεται η βασική σύνταξη των Regexes, και ο τρόπος δημιουργίας ευέλικτων μοτίβων αντιστοίχισης και ο τρόπος με τον οποίο η μηχανή κανονικής έκφρασης αναλύει κείμενο για να βρει ομοιότητες. Μαθαίνουμε να γράφουμε Regexes χρησιμοποιώντας σύνολα χαρακτήρων, ομαδοποίηση, εναλλαγή, άγκυρες έναρξης και λήξης, όρια λέξεων και άλλα.

4.6.2.2 *Infrastructure and Configuration as Code*

Το **Terraform** είναι ένα εργαλείο DevOps για δηλωτική υποδομή, υποδομή ως κώδικα. Απλοποιεί και επιταχύνει τη διαμόρφωση περιβαλλόντων που βασίζονται σε Cloud. Σε αυτό το μάθημα, θα μάθουμε πώς να χρησιμοποιούμε το Terraform για τη διαμόρφωση της υποδομής και τη διαχείριση πόρων με το **Amazon Web Services** (AWS). Αφού δούμε πώς να ρυθμίσουμε το AWS για το Terraform θα καλύψουμε τον τρόπο με τον οποίο η Terraform διαχειρίζεται την υποδομή καθώς και τον τρόπο χρήσης των βασικών εντολών Terraform. Εξετάζονται επίσης πιο προηγμένα θέματα, συμπεριλαμβανομένου του τρόπου ρύθμισης του κώδικα Terraform ώστε να είναι πιο διατηρήσιμος και αρθρωτός. Με την ολοκλήρωση αυτού του μαθήματος, θα έχουμε τις γνώσεις που χρειαζόμαστε για να ορίσουμε και να διαχειριστούμε αποτελεσματικά την υποδομή με αυτό το ισχυρό εργαλείο.

Το **Ansible** είναι ένας εύχρηστος μηχανισμός αυτοματισμού πληροφορικής. Μπορεί να μεταμορφώσει και να βελτιστοποιήσει εργασίες όπως κυλιόμενες ενημερώσεις με λίγες γραμμές κώδικα. Ωστόσο, πολλοί οργανισμοί δεν χρησιμοποιούν λειτουργικά τμήματα και playbooks με δυνατότητα επαναχρησιμοποίησης. Αυτό το μάθημα εισάγει ενδιάμεσες έως προχωρημένες

περιπτώσεις χρήσης του εργαλείου σε κλίμακα. Θα δούμε πώς να επαναχρησιμοποιήσουμε τη λειτουργία Ansible με ρόλους, να κρυπτογραφήσετε δεδομένα με το **Ansible Vault**, να υποστηρίξουμε την αδυναμία και να διαχειριστούμε πολλές ομάδες συστημάτων με το **Ansible Tower**. Βλέπουμε πώς να συνδυάσουμε αυτές τις έννοιες για να χρησιμοποιήσουμε το Ansible με τον πιο αποτελεσματικό δυνατό τρόπο: μεταβλητή πόρων, σύλληψη των ιδεών σε ρόλους και επέκταση του Ansible από απλή διαχείριση διακομιστή σε διαχείριση δικτύου.

4.6.2.3 Cloud Native

Αν θέλουμε να επιταχύνουμε τη διαδικασία ρύθμισης και ανάπτυξης εφαρμογών σε ένα σύμπλεγμα Kubernetes, το **Helm** είναι το εργαλείο διαχείρισης πακέτων για το Kubernetes που μπορεί να βοηθήσει. Αυτό το εργαλείο προσφέρει στους προγραμματιστές έναν τρόπο γρήγορης ανάπτυξης και ενημέρωσης εφαρμογών σε ένα σύμπλεγμα, καθώς και επαναφορά σε προηγούμενη ανάπτυξη. Σε αυτό το μάθημα, εξηγείται πώς να εγκαταστήσουμε το Helm στον υπολογιστή μας και το πώς να εγκαταστήσουμε ένα **Helm Chart**, μια συλλογή αρχείων που περιγράφουν ένα σύνολο πόρων στο σύμπλεγμα Kubernetes, καθώς και πώς να επαληθεύσουμε εάν η εγκατάστασή λειτουργήσει, ενημέρωσε το γράφημα και επανέφερε τις αλλαγές. Εκτός από την παρουσίαση του τρόπου χρήσης ενός προκαταρκτικού γραφήματος Helm επιδεικνύεται πώς να δημιουργήσουμε ένα νέο γράφημα Helm από τη γραμμή εντολών, να αναπτύξουμε και να ενημερώσουμε ένα Kubernetes ConfigMap και secret μέσω Helm και να αξιοποιήσουμε προηγμένες λειτουργίες, όπως τον τρόπο απόδοσης τιμής ConfigMap δυναμικά με το πρότυπο Helm.

4.6.2.4 Continuous Testing

Το επόμενο βήμα πέρα από την ευέλικτη δοκιμή και τον αυτοματισμό δοκιμών είναι οι συνεχείς δοκιμές. Η συνεχής δοκιμή σημαίνει ότι η τακτική δοκιμαστική αυτοματοποίηση εκτελείται συχνά και δίνουμε πράσινο φως στις αλλαγές στον κώδικα πριν προχωρήσουμε στη διαδικασία ανάπτυξης. Ο κύκλος ζωής της ανάπτυξης επιταχύνεται και συμβάλει σε πιο σταθερές και αξιόπιστες παραδόσεις λογισμικού. Το DevOps παρέχει την υποδομή για να ξεκινήσουμε να εκτελούμε τον αυτοματισμό σε έναν αγωγό συνεχούς ολοκλήρωσης. Αυτό το μάθημα προσφέρει τις δεξιότητες και μαθαίνουμε τις βασικές έννοιες των συνεχών δοκιμών και τι σημαίνει το DevOps για το QA. Ανακαλύπτουμε μεθόδους για την εκτέλεση αυτοματοποιημένων δοκιμών σε έναν αγωγό συνεχούς ολοκλήρωσης και εξερευνούμε κοινά εργαλεία που θα συναντήσουμε κατά τη διαδικασία εφαρμογής συνεχών δοκιμών.

4.6.2.5 Monitoring and Logging

Το **Elastic Stack** (παλαιότερα γνωστό ως **ELK Stack**) είναι ένα ισχυρό σύνολο εργαλείων ανοιχτού κώδικα για τη διαχείριση και κατανόηση σύνθετων δεδομένων. Χρησιμοποιώντας το Elastic Stack, αναζητούμε, οπτικοποιούμε και επεξεργαζόμαστε πιο εύκολα δεδομένα, καθώς και συγκεντρώνουμε δεδομένα καταγραφής για να λύσουμε γρήγορα τα προβλήματα. Σε αυτό το μάθημα, ξεκινάμε να τρέχουμε με το Elastic Stack, ώστε να μπορούμε να επωφεληθούμε από αυτά τα εργαλεία. Επιπλέον, μαθαίνουμε για διαφορετικά στοιχεία της στοίβας, όπως **Logstash**, **Beats** και **Kibana**.

Προκειμένου να αποφευχθούν οι διακοπές λειτουργίας, είναι σημαντικό να αξιοποιήσετε ένα εργαλείο παρακολούθησης και ειδοποίησης στο περιβάλλον Kubernetes. Το **Prometheus** είναι ένα κιτ εργαλείων παρακολούθησης και ειδοποίησης ανοιχτού κώδικα και συνδυάζεται ιδιαίτερα καλά με το Kubernetes. Σε αυτό το μάθημα, μαθαίνουμε πώς ενσωματώνεται αυτή η εργαλειοθήκη με το Kubernetes και λειτουργεί για την παρακολούθηση καταναμεμένων συστημάτων. Επίσης βλέπουμε την παρακολούθηση δεδομένων για συγκεκριμένες εφαρμογές, την προσθήκη κοντέινερ για δεδομένα εφαρμογών, το φιλτράρισμα και το συνδυασμό μετρήσεων και την εμφάνιση μετρήσεων στην κονσόλα ιστού. Τέλος θα δούμε πώς να δημιουργήσουμε μια απλή ειδοποίηση στο Prometheus και ειδοποιήσεις βάσει εφαρμογών.

4.6.2.6 Security

Τα κοντέινερ έχουν μεταμορφώσει τον τρόπο κατασκευής, ανάπτυξης και διαχείρισης των εφαρμογών. Αλλά είναι μόνο θέμα χρόνου προτού μια ευπάθεια ή εσφαλμένη διαμόρφωση οδηγήσει σε μια νέα αξιόλογη παραβίαση της ασφάλειας. Η καθιέρωση ελέγχων ασφαλείας είναι απαραίτητη για την προστασία εφαρμογών που αναπτύσσονται σε συστήματα ενορχήστρωσης όπως το Kubernetes. Αυτό το μάθημα εξηγεί πώς να ασφαλίσουμε το κοντέινερ και το οικοσύστημα Kubernetes χρησιμοποιώντας ένα απλό μοντέλο πέντε παραγόντων. Επανεξετάζονται η αρχιτεκτονική και η τεχνολογία Kubernetes, ο κύκλος ζωής και ανάπτυξης εφαρμογών. Στη συνέχεια, εισάγεται το μοντέλο ασφαλείας πέντε παραγόντων για την προστασία του Kubernetes και των συστατικών του, και βλέπουμε πώς μπορούν να εφαρμοστούν έλεγχοι ασφαλείας στο σχεδιασμό και τον κώδικα, τις εικόνες και τα μητρώα, τα κοντέινερ και τους κεντρικούς υπολογιστές, τις εφαρμογές και τα συμπλέγματα Kubernetes.

4.6.3 DevOps Advanced (Προχωρημένος DevOps)

Οι ενότητες που περιλαμβάνει αυτό το μονοπάτι δίνονται παρακάτω με μια σύντομη περιγραφή για το περιεχόμενο της κάθε ενότητας:

4.6.3.1 DevOps & SRE concepts

Site Reliability Engineering (**SRE**) είναι ένα αναδυόμενο παράδειγμα στο DevOps. Τα μεγαλύτερα ονόματα στην τεχνολογία και εταιρείες όπως **Google**, **Netflix**, **Microsoft** και **LinkedIn** χρησιμοποιούν το SRE. Στην πραγματικότητα, σε ολόκληρη τη βιομηχανία, ο "μηχανικός αξιοπιστίας ιστότοπου" αντικαθιστά τον "μηχανικό DevOps" σε θέσεις εργασίας. Με απλά λόγια, το SRE είναι εφαρμοσμένη μηχανική λογισμικού σε λειτουργίες για την εποχή του cloud. Αυτό το μάθημα εισάγει τα βασικά στοιχεία της μηχανικής αξιοπιστίας ιστότοπων, συμπεριλαμβανομένου του τρόπου με τον οποίο το SRE ταιριάζει στο DevOps και πώς μπορεί να ενσωματωθεί στο μοναδικό επιχειρηματικό περιβάλλον. Καλύπτονται οι κύριοι τομείς εμπειρογνωμοσύνης, όπως μηχανική απελευθέρωσης, διαχείριση αλλαγών, διαχείριση συμβάντων και αναδρομικές προοπτικές, αυτοματισμός αυτοεξυπηρέτησης, αντιμετώπιση προβλημάτων, επιδόσεις και εσκεμμένες δυσκολίες. Μαθαίνουμε πώς να ορίζουμε την αξιοπιστία μέσω SLA και SLO, να χειριζόμαστε κρίσεις, να σχεδιάζουμε καταναμεμένα συστήματα και να κλιμακώνουμε τα

συστήματά και την ομάδα. Επιπλέον, εξερευνούμε το χρόνο και τις στρατηγικές διαχείρισης έργων που επαναφέρουν την ανθρωπότητα στη δουλειά του SRE.

4.6.3.2 *Kubernetes*

Ανακαλύπτουμε πώς μπορούμε να βελτιώσουμε την ανάπτυξη του Cloud με το **Istio**, μια ισχυρή τεχνολογία που υποστηρίζει ένα μοντέλο συνδεσιμότητας βασισμένο σε πλέγμα υπηρεσιών για εφαρμογές μικροϋπηρεσιών σε ένα κατανεμημένο περιβάλλον διαχείρισης όπως το Kubernetes. Οι υπηρεσίες εμπλέκονται γενικά και το Istio συγκεκριμένα, απαντά ορισμένα από τα βασικά ζητήματα που αντιμετωπίζουν οι ομάδες ανάπτυξης κατά τη δημιουργία συστημάτων που βασίζονται σε μικροϋπηρεσίες. Σε αυτό το μάθημα βλέπουμε πώς ενεργοποιείται το Istio και ενσωματώνεται σε οποιοδήποτε περιβάλλον εφαρμογών που βασίζεται στο Kubernetes. Καλύπτεται ο τρόπος προσέγγισης της δρομολόγησης της κυκλοφορίας και της εξισορρόπησης με καθιέρωση διαπιστευτηρίων MTLs και σύνδεση σε υπηρεσίες non-MTLs.

4.6.3.3 *Business Continuity & Disaster Recovery*

Τα αντίγραφα ασφαλείας δημιουργούν ένα αντίγραφο δεδομένων που μπορούν να ανακτηθούν σε περίπτωση πρωταρχικής αποτυχίας δεδομένων. Αυτές οι αστοχίες μπορεί να οφείλονται σε αστοχία υλικού ή λογισμικού, καταστροφή δεδομένων ή σε ανθρώπινο συμβάν, όπως κακόβουλη επίθεση ή τυχαία διαγραφή. Ως εκ τούτου, η δημιουργία αντιγράφων ασφαλείας και η ανάκτηση συστήματος είναι μια κρίσιμη δεξιότητα τόσο για τους διαχειριστές συστήματος όσο και για τους οργανισμούς στους οποίους εργάζονται. Αυτό το μάθημα βοηθά τους διαχειριστές συστημάτων να διασφαλίσουν ότι προστατεύουν δεδομένα και συστήματα από απώλεια. Μαθαίνουμε πώς να δημιουργούμε αντίγραφα ασφαλείας των δύο πιο δημοφιλών λειτουργικών συστημάτων Windows και Linux χρησιμοποιώντας δωρεάν βοηθητικά προγράμματα και λύσεις τρίτων.

5

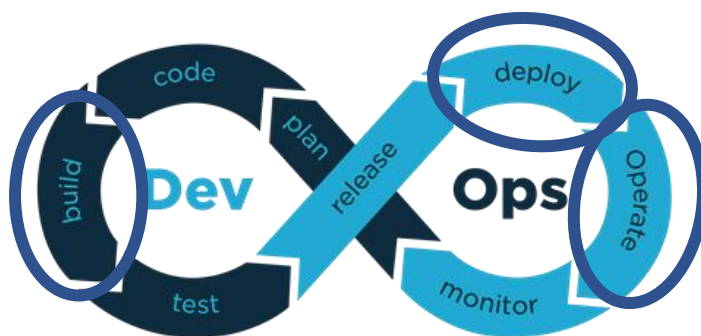
Υλοποίηση Αγωγού Ανάπτυξης CI/CD

Το κεφάλαιο αυτό είναι παράδειγμα υλοποίησης ορισμένων φάσεων μοντέλου DevOps ώστε να κατανοηθούν κάποιες από τις εργασίες που θα κληθεί μια ομάδα DevOps να υλοποιήσει σε ένα οργανισμό.

Ο σκοπός αυτού του κεφαλαίου είναι, εφόσον έχουν καλυφθεί σε προηγούμενα κεφάλαια όλα τα θεωρητικά κομμάτια και οργανωτικά θέματα DevOps να προχωρήσουμε σε ένα παράδειγμα εφαρμόζοντας την θεωρία σε πρακτικό κομμάτι.

5.1 Εισαγωγή και εργαλεία

Η υλοποίησή θα επικεντρωθεί στις φάσεις: Deploy, Operate και Build. Το κομμάτι του κώδικα (code) είναι έτοιμο σε αποθετήριο GitHub καθώς δεν αποτελεί κομμάτι εργασίας DevOps. Επίσης, στο παράδειγμα μας δεν χρησιμοποιήσαμε κάποια πολιτική κλάδων για το release, ούτε κάναμε κάποιες δοκιμές (test) και παρακολούθηση (monitor).



Εικόνα 5-1: Φάσεις DevOps για το παράδειγμα

Η υλοποίηση του αγωγού ανάπτυξης CI/CD έγινε με την χρήση εργαλείων όπως GitHub, Jenkins, Maven.

Προαπαιτούμενα:

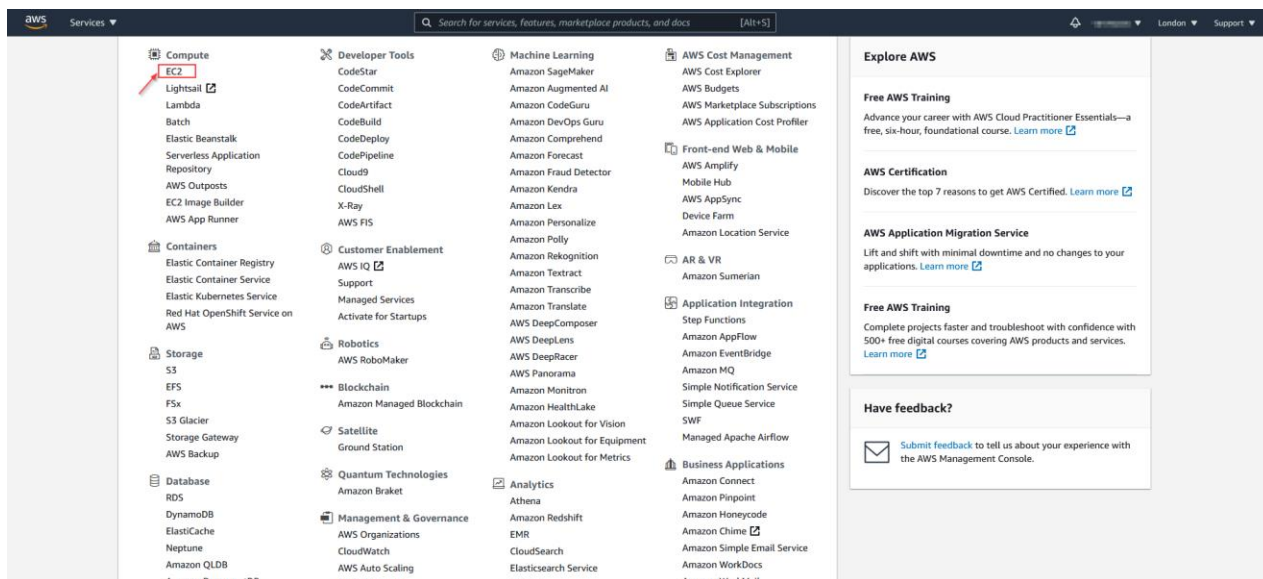
- **GitHub account:** Είναι το αποθετήριο που θα βρούμε το κομμάτι του κώδικα της εφαρμογής που θα κάνουμε deploy.
- **[AWS Free Tier account](#):** Η πλατφόρμα παροχής των διακομιστών για τις εγκαταστάσεις των εργαλείων.
- **[MobaXterm](#):** Το τερματικό για την ασφαλή σύνδεση μας στους διακομιστές.

5.2 Αγωγός Ανάπτυξης CI/CD με GIT, Jenkins και Maven

5.2.1 Εγκατάσταση Jenkins

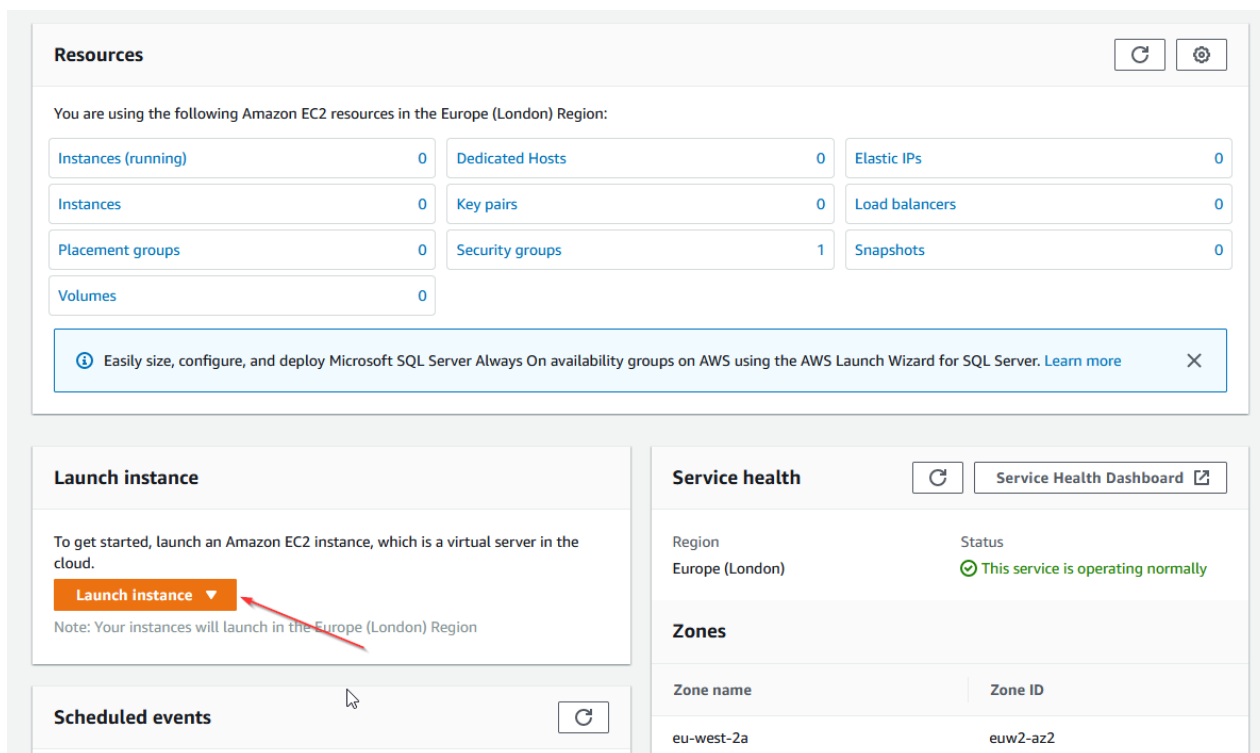
Προαπαιτούμενο για την εγκατάσταση Jenkins είναι ένα EC2 Instance το οποίο θα δημιουργήσουμε παρακάτω με τις οδηγίες.

- Κάνουμε login στο AWS account που δημιουργήσαμε από το παραπάνω [link](#) και επιλέγουμε EC2 όπως φαίνεται στην παρακάτω εικόνα:



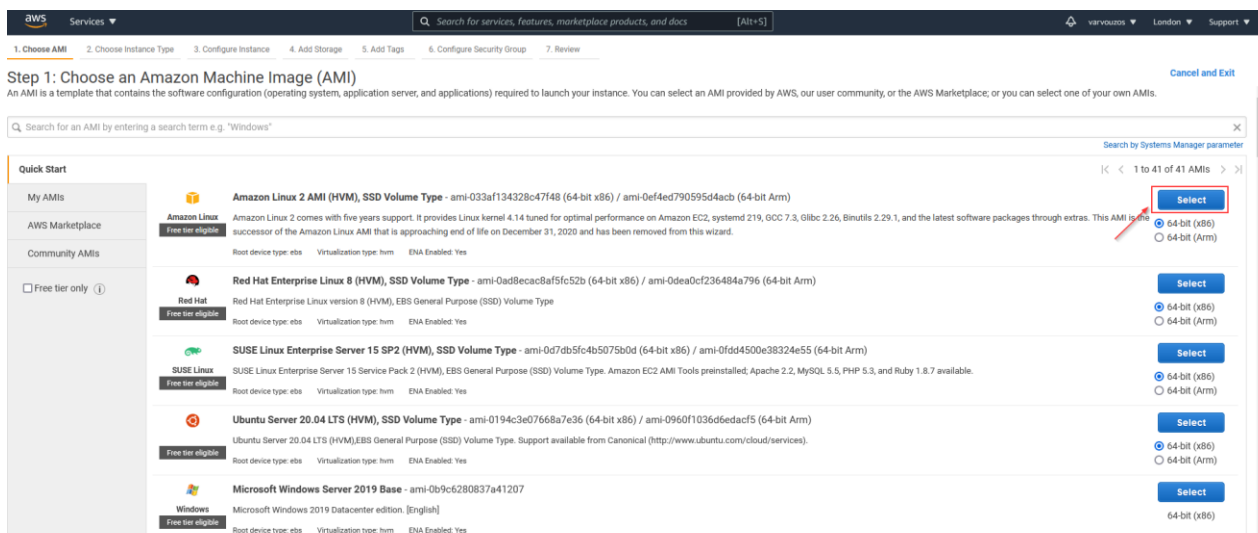
Εικόνα 5-2: EC2 Amazon Instance

- Από την επιλογή “Launch instance” μπαίνουμε στο διάλογο για την δημιουργία του instance:



Εικόνα 5-3: Εισαγωγή στο διάλογο για δημιουργία διακομιστή AWS

- Πατάμε “Select” στο “Amazon Linux”



Εικόνα 5-4: Επιλογή έκδοσης λειτουργικού

- Πηγαίνουμε στο βήμα “Add Tags” όπου προσθέτουμε “Key = Name” και “Value = Jenkins Server” και έπειτα πάμε στο “Configure Security Group” όπου φτιάχνουμε ένα καινούργιο με όνομα “DevOps_SG”. Προσθέτουμε και ένα Rule για να αφήνει την default TCP port 8080 του Jenkins.

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name:

Description:

Type (1)	Protocol (1)	Port Range (1)	Source (1)	Description (1)
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP	TCP	8080	Custom 0.0.0.0/0, ::/0	e.g. SSH for Admin Desktop

Warning

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Εικόνα 5-5: Δημιουργία Security Group για την ασφαλή πρόσβαση

- Πατάμε “Review and Launch” και στη συνέχεια “Launch” για να δημιουργήσουμε ένα καινούργιο key pair:

Select an existing key pair or create a new key pair ✕

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair ▾

Key pair name

DevOps

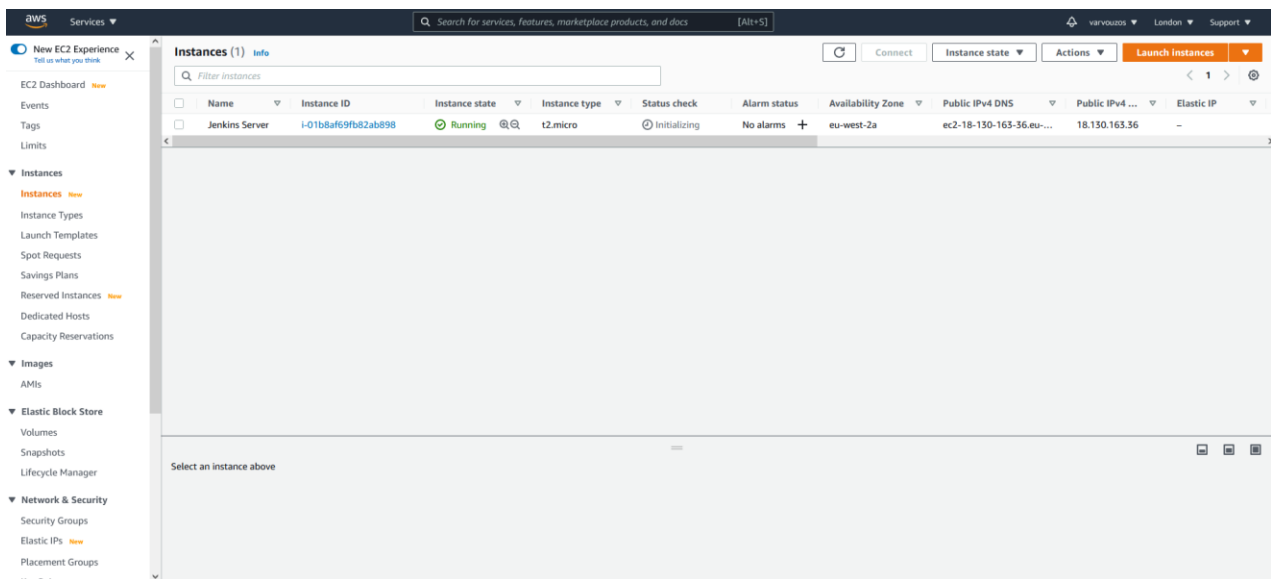
Download Key Pair

⋮ You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

Cancel Launch Instances

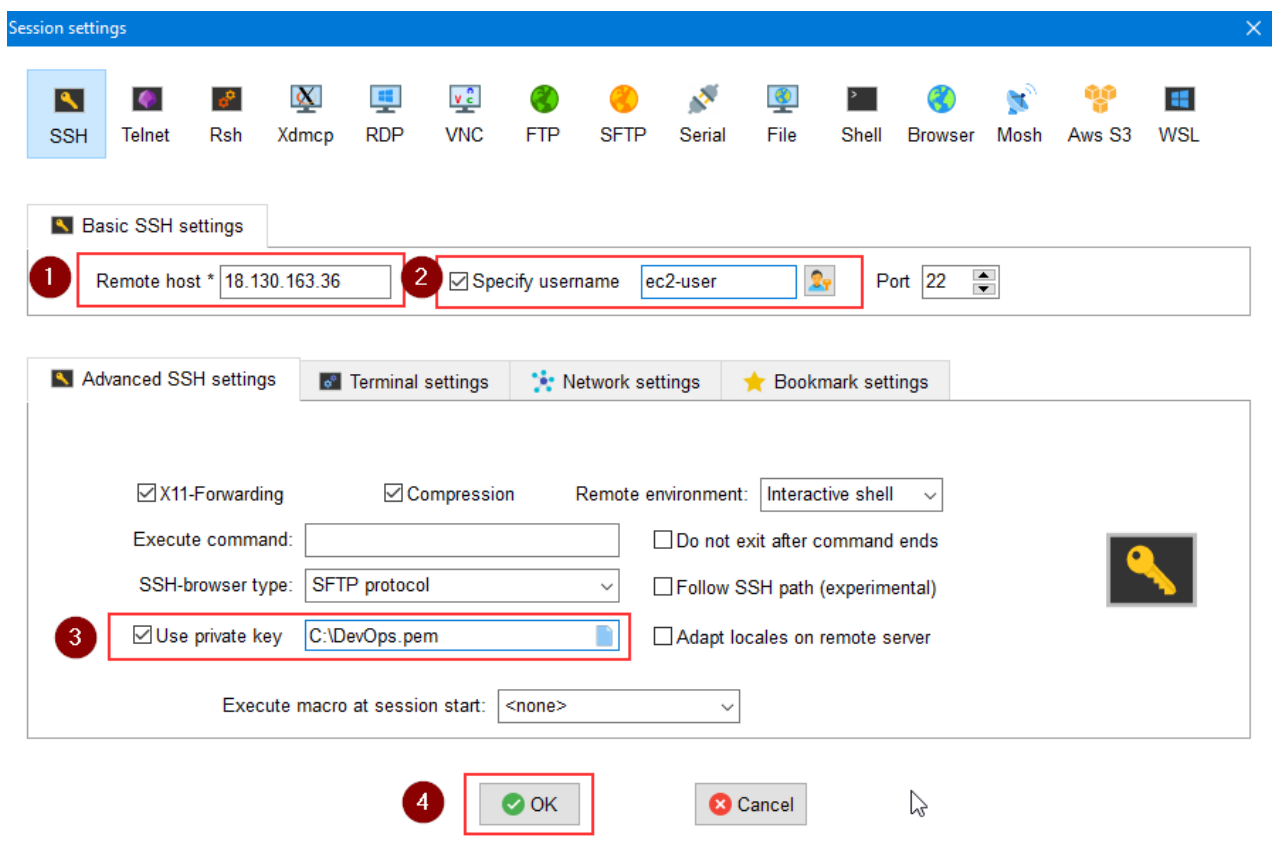
Εικόνα 5-6: Δημιουργία keypair

- Κατεβάζουμε το Key Pair και το αποθηκεύουμε για να το χρησιμοποιήσουμε για την σύνδεση με το instance.

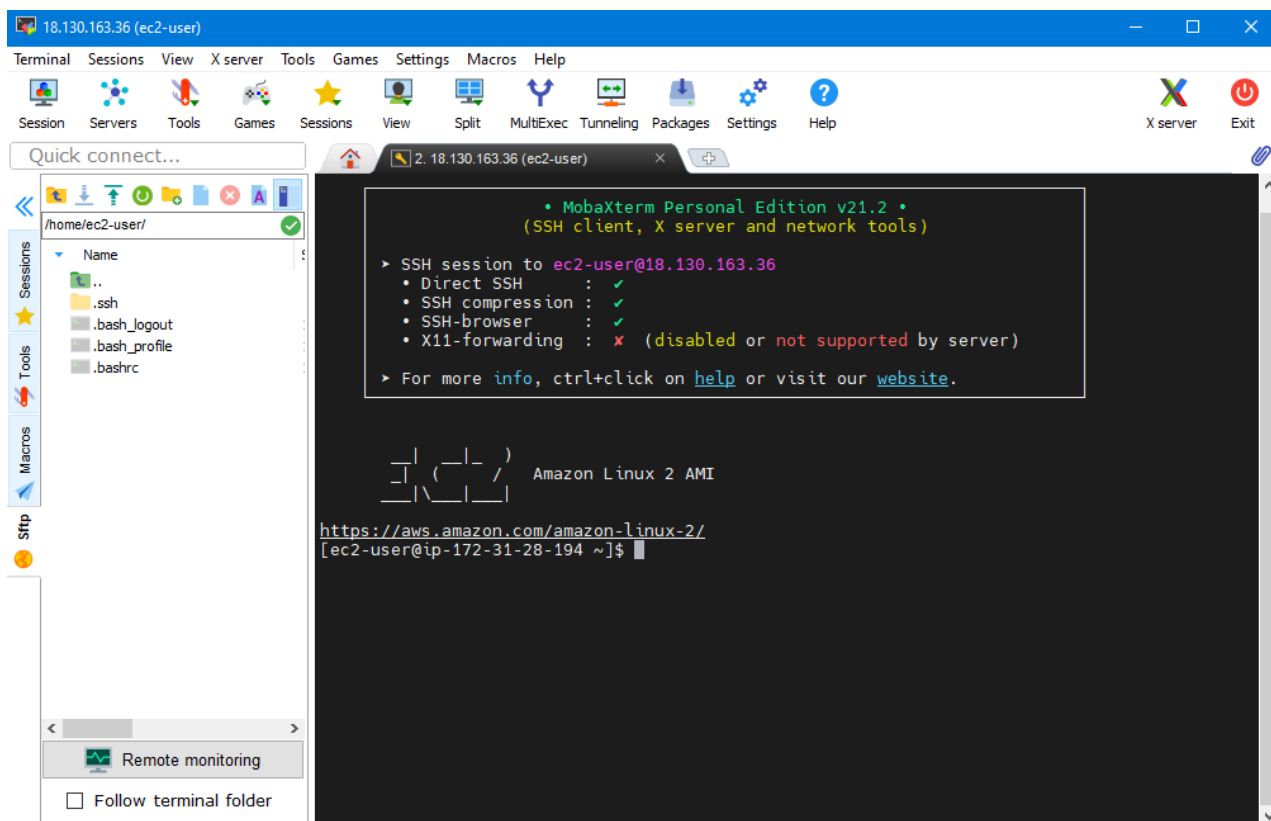


Εικόνα 5-8: Ο διακομιστής έπειτα την δημιουργία

- Με την χρήση του MobaXterm κάνουμε SSH login στο Instance που δημιουργήσαμε για να προχωρήσουμε την εγκατάσταση χρησιμοποιώντας την Public IP του μηχανήματος που δημιουργήσαμε (Remote host):



Εικόνα 5-7: Ρυθμίσεις τερματικού MobaXterm



Εικόνα 5-9: Είσοδος στον διακομιστή για την εγκατάσταση Jenkins

- Πληκτρολογούμε `sudo su -` για να γίνουμε root χρήστες.
- Για την εγκατάσταση Java θα χρησιμοποιήσουμε `open java` από <http://openjdk.java.net/install/>
`yum install java-1.8*`
- Προσθέτουμε το Java home path μέσα στο `bash_profile`:
`vi ~/.bash_profile`


```
[root@ip-172-31-28-194 ~]# sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.i
o.key
[root@ip-172-31-28-194 ~]# yum install jenkins
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
jenkins | 2.9 kB 00:00:00
jenkins/primary_db | 37 kB 00:00:00
Resolving Dependencies
--> Running transaction check
--> Package jenkins.noarch 0:2.289.1-1.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository Size
=====
Installing:
jenkins noarch 2.289.1-1.1 jenkins 71 M
Transaction Summary
=====
Install 1 Package

Total download size: 71 M
Installed size: 71 M
Is this ok [y/d/N]: y
Downloading packages:
jenkins-2.289.1-1.1.noarch.rpm | 71 MB 00:00:02
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : jenkins-2.289.1-1.1.noarch 1/1
Verifying : jenkins-2.289.1-1.1.noarch 1/1

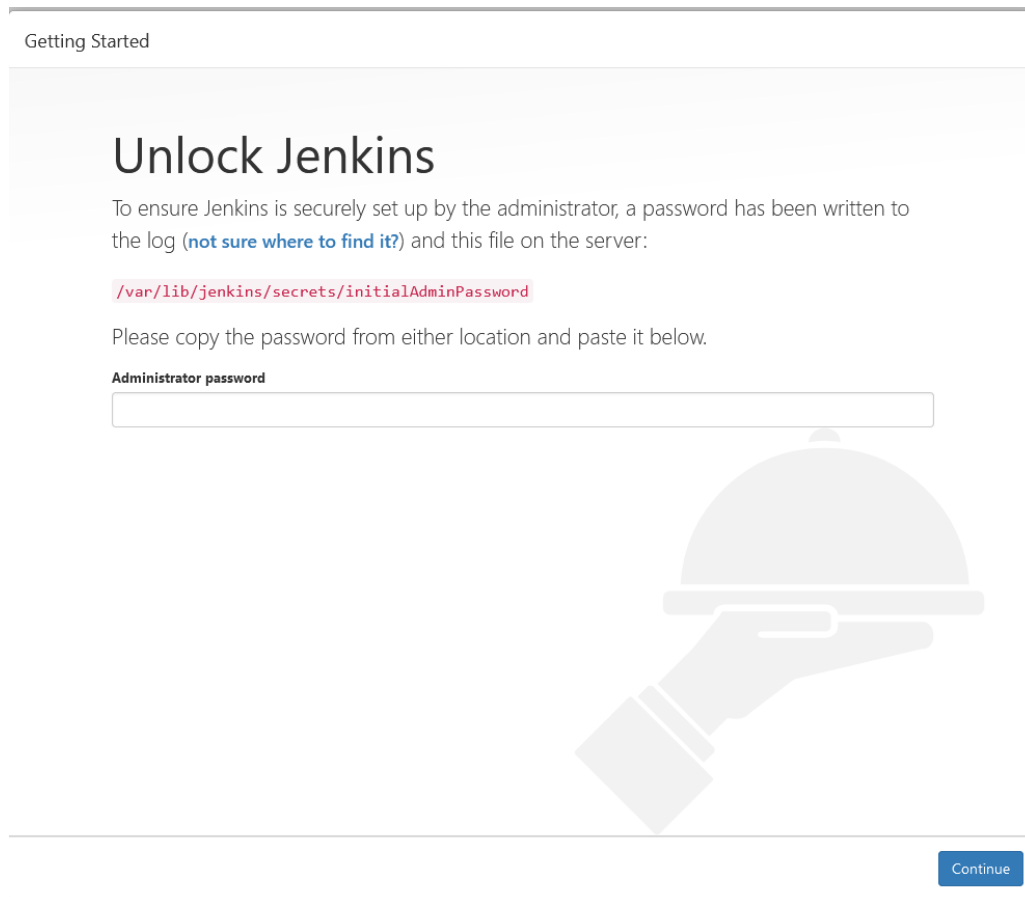
Installed:
jenkins.noarch 0:2.289.1-1.1

Complete!
[root@ip-172-31-28-194 ~]# service jenkins start
service jenkins start
Starting jenkins (via systemctl): [ OK ]
```

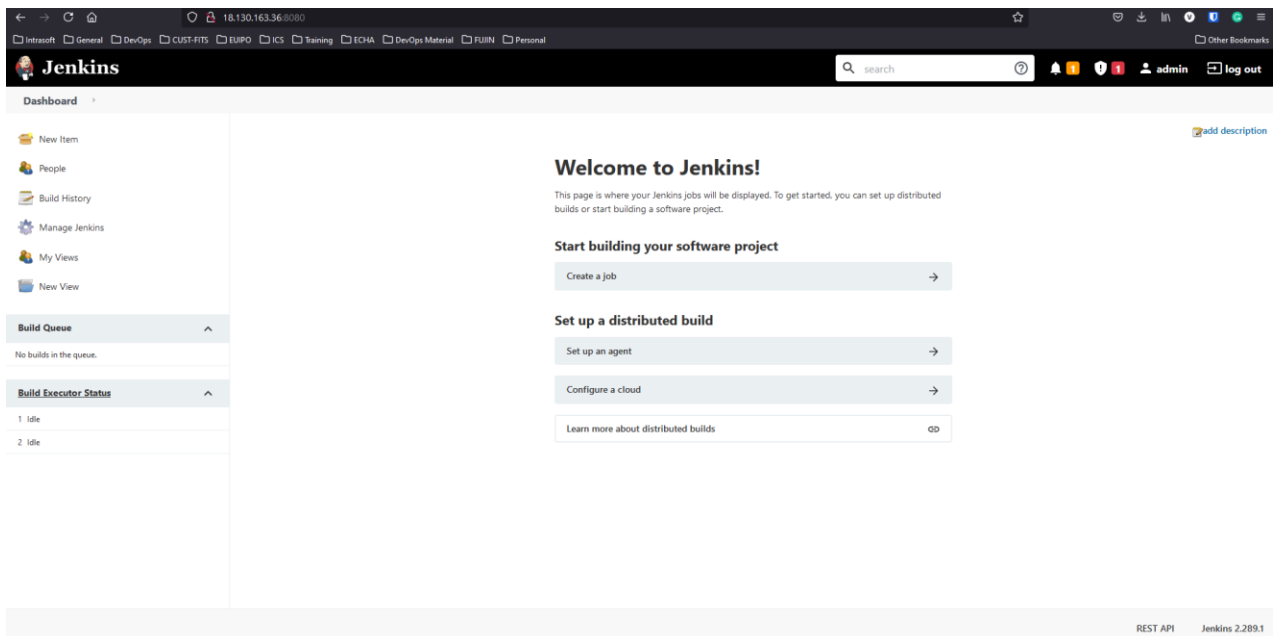
Εικόνα 5-11: Εκκίνηση Jenkins

- Για να κάνουμε Login στο Jenkins ανοίγουμε τον Browser και χρησιμοποιούμε την Public IP του AWS Jenkins instance στην πόρτα 8080. Έπειτα για να κάνουμε unlock τον Jenkins πρέπει να εντοπίσουμε το κλειδί μέσω SSH:

```
cat /var/lib/jenkins/secrets/initialAdminPassword
```



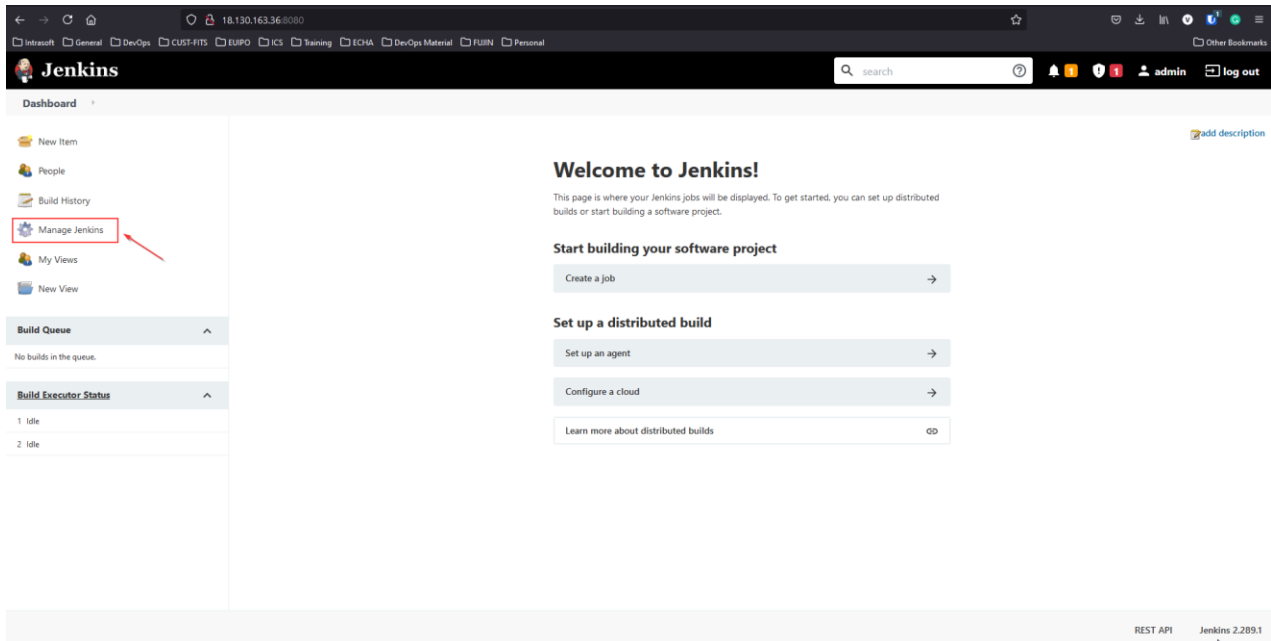
Εικόνα 5-12: Ξεκλείδωμα Jenkins στον browser



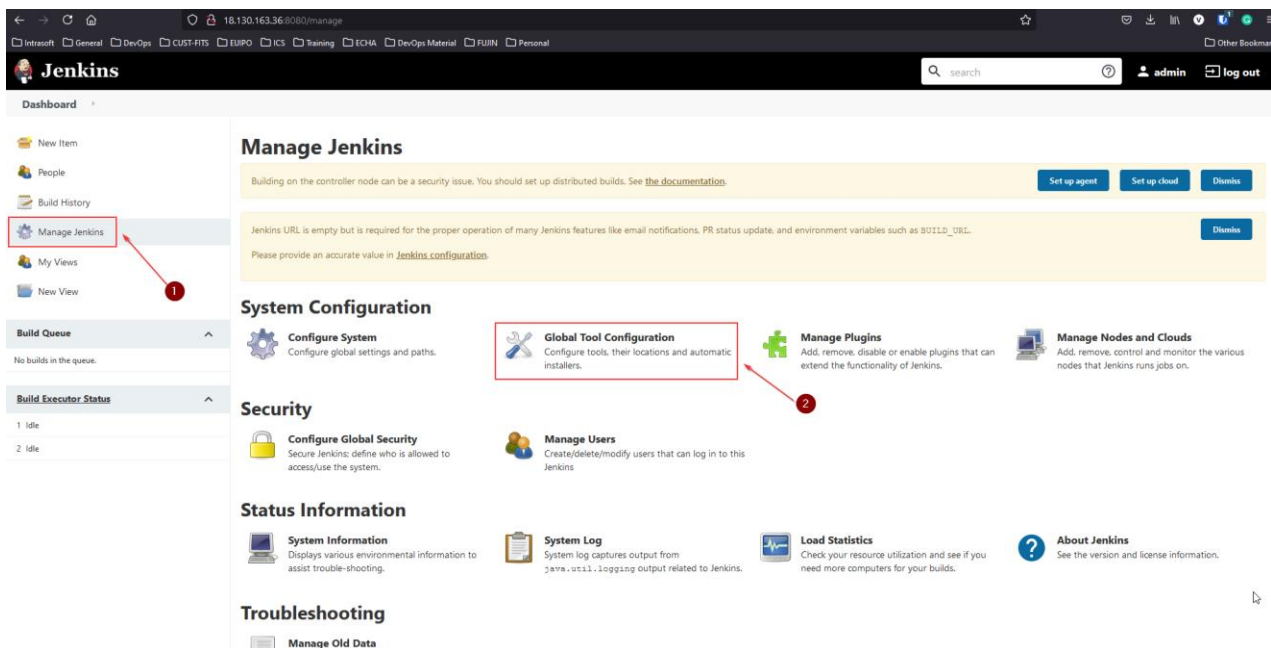
Εικόνα 5-13: Είσοδος στο γραφικό περιβάλλον Jenkins

- Απαραίτητη προϋπόθεση μετά το Login να κάνουμε αλλαγή password πηγαίνοντας admin → Configure → [Αλλαγή password] → Save

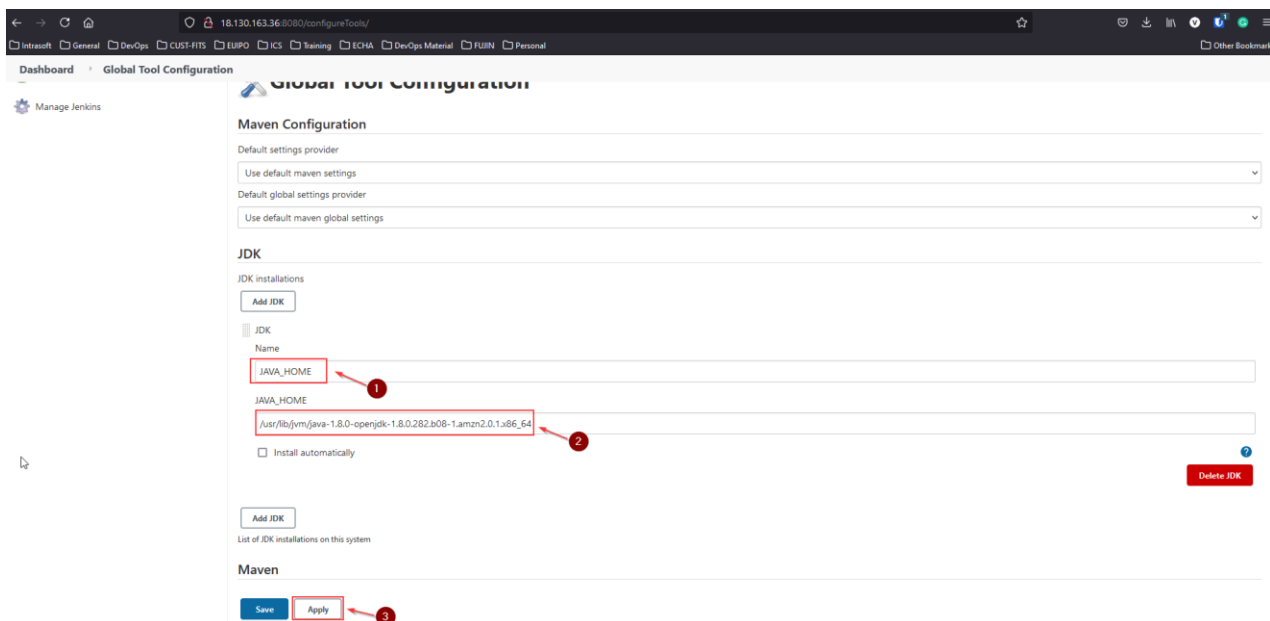
- Στη συνέχεια δηλώνουμε το path που έχουμε κάνει την εγκατάσταση της Java: Manage Jenkins → Global Tool Configuration → Add JDK → Name = JAVA_HOME και βρίσκουμε με την εντολή `echo $JAVA_HOME` στο MobaXterm και το κάνουμε αντιγραφή – επικόλληση στο πεδίο JAVA_HOME και πατάμε “Apply”



Εικόνα 5-14: Ρυθμίσεις Jenkins

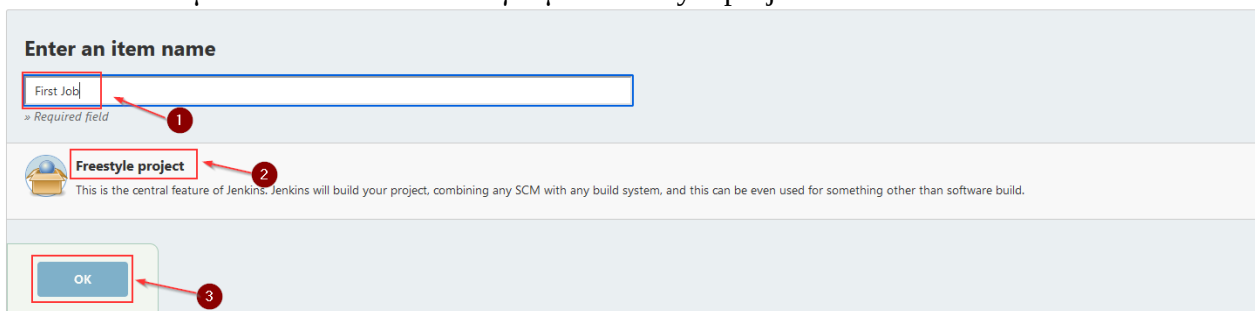


Εικόνα 5-15: Διαμόρφωση Jenkins



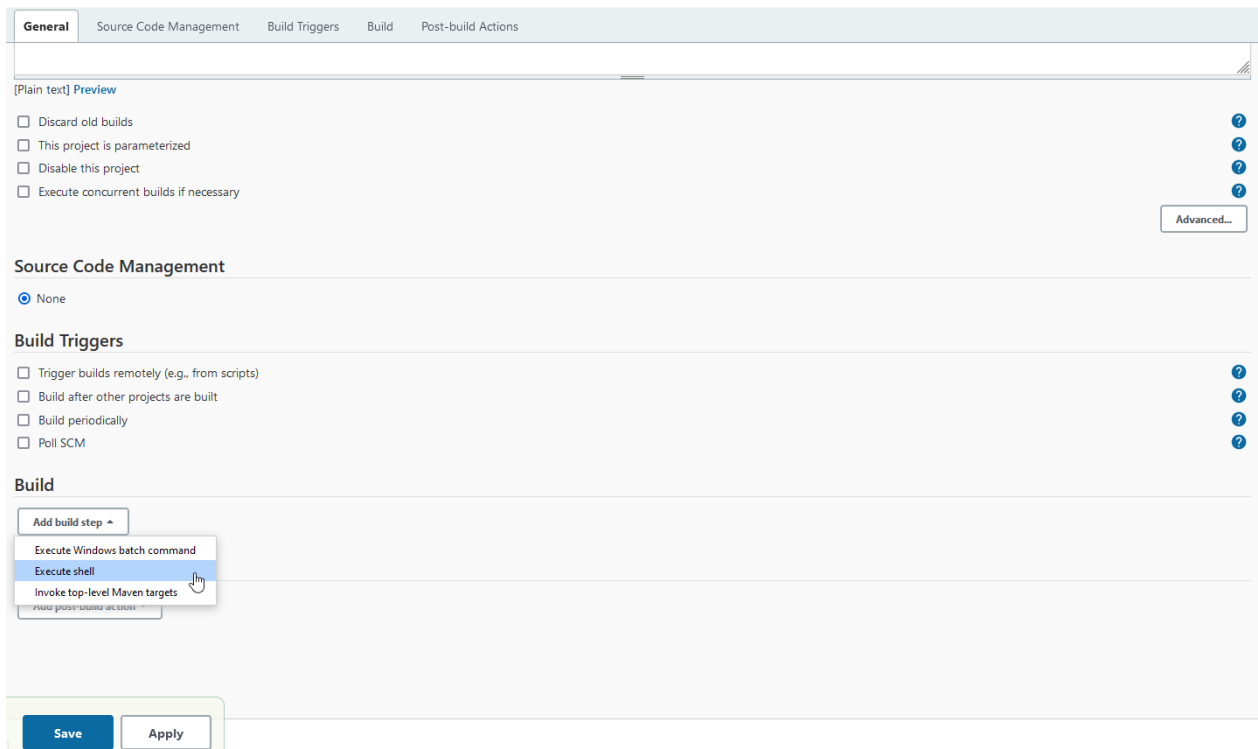
Εικόνα 5-16: Δήλωση τοποθεσίας Java Home

- Για την δημιουργία ενός Job πατάμε “New Item” από την αρχική οθόνη και έπειτα δίνουμε ένα όνομα “First Job” και επιλέγουμε “Freestyle project”:



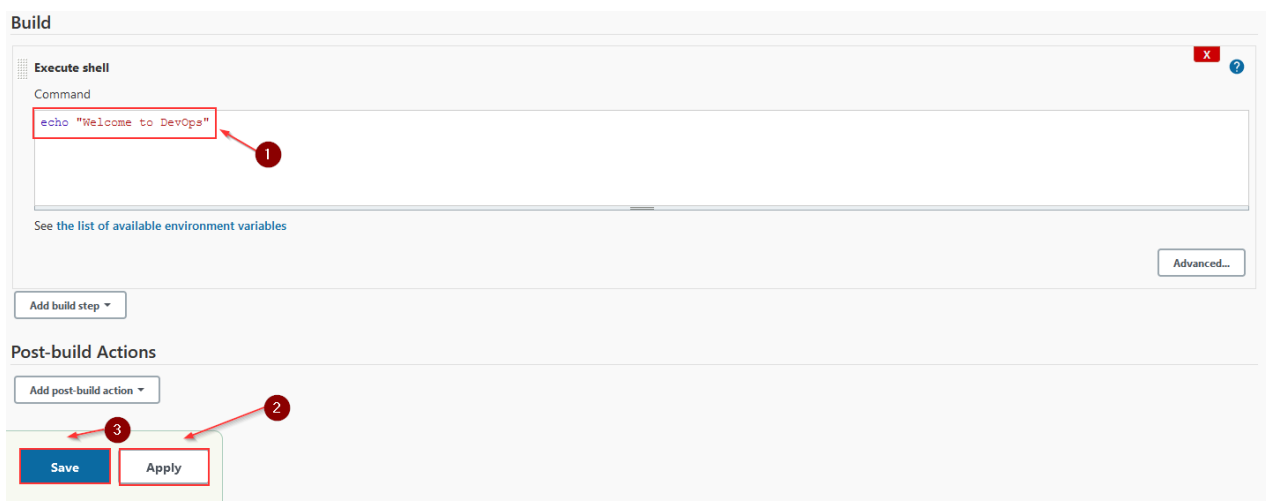
Εικόνα 5-17: Δημιουργία Jenkins Job

- Στην επιλογή Build επιλέγουμε “Execute shell” από την στιγμή που χρησιμοποιούμε Linux



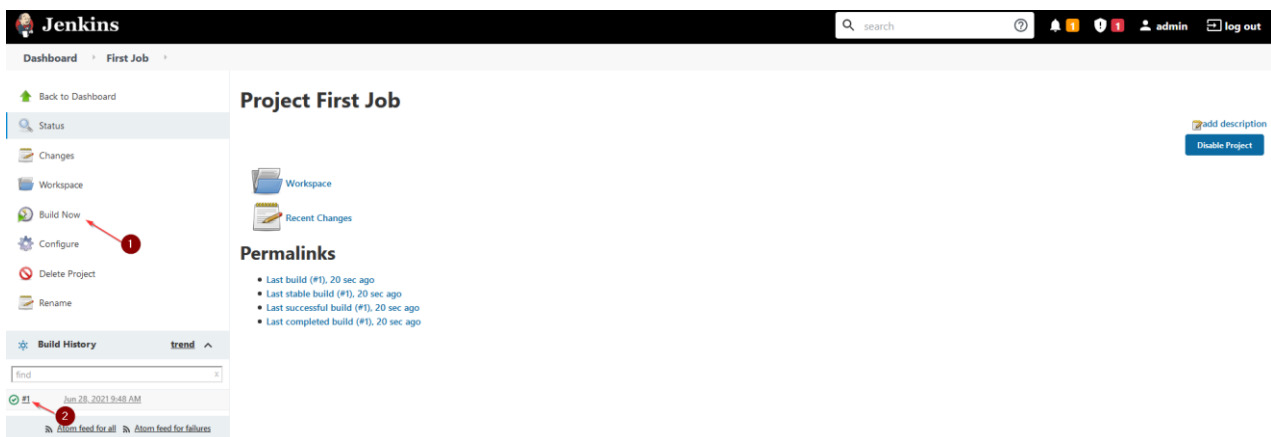
Εικόνα 5-18: Ρυθμίσεις για το Build

- Εισάγουμε στο Command: `echo "Welcome to DevOps"` → Apply → Save

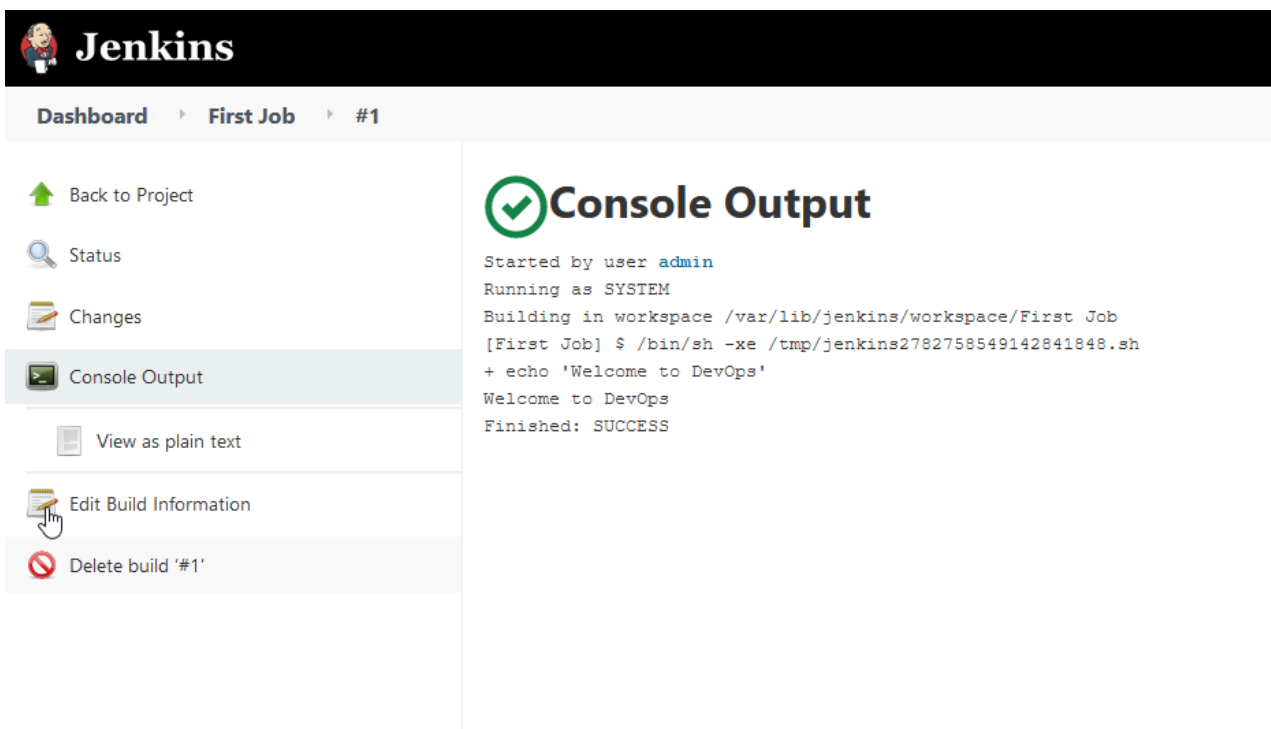


Εικόνα 5-19: Εντολή δοκιμής echo

- Πατάμε “Build Now” και μετά πάνω στην αρίθμηση για να ανοίξει η κονσόλα:



Εικόνα 5-21: Πρώτο Build



Εικόνα 5-20: Αποτελέσματα κονσόλας πρώτου Build

5.2.2 Εγκατάσταση Git

- Στο MobaXterm στο Jenkins instance πληκτρολογούμε

```
yum install git -y
```

```
[root@ip-172-31-28-194 ~]# yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core | 3.7 kB 00:00:00
amzn2extra-docker | 3.0 kB 00:00:00
jenkins | 2.9 kB 00:00:00
Resolving Dependencies
--> Running transaction check
--> Package git.x86_64 0:2.23.4-1.amzn2.0.1 will be installed
--> Processing Dependency: perl-Git = 2.23.4-1.amzn2.0.1 for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Processing Dependency: git-core-doc = 2.23.4-1.amzn2.0.1 for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Processing Dependency: git-core = 2.23.4-1.amzn2.0.1 for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Processing Dependency: emacs-filesystem >= 25.3 for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Processing Dependency: perl(Git::I18N) for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Processing Dependency: perl(Git) for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Processing Dependency: libsecret-1.so.0()(64bit) for package: git-2.23.4-1.amzn2.0.1.x86_64
--> Running transaction check
--> Package emacs-filesystem.noarch 1:27.1-1.amzn2.0.1 will be installed
--> Package git-core.x86_64 0:2.23.4-1.amzn2.0.1 will be installed
--> Package git-core-doc.noarch 0:2.23.4-1.amzn2.0.1 will be installed
--> Package libsecret.x86_64 0:0.18.5-2.amzn2.0.2 will be installed
--> Package perl-Git.noarch 0:2.23.4-1.amzn2.0.1 will be installed
--> Processing Dependency: perl(Error) for package: perl-Git-2.23.4-1.amzn2.0.1.noarch
--> Package perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2 will be installed
--> Running transaction check
--> Package perl-Error.noarch 1:0.17020-2.amzn2 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository Size
=====
Installing:
git x86_64 2.23.4-1.amzn2.0.1 I amzn2-core 135 k
Installing for dependencies:
emacs-filesystem noarch 1:27.1-1.amzn2.0.1 amzn2-core 64 k
git-core x86_64 2.23.4-1.amzn2.0.1 amzn2-core 5.0 M
git-core-doc noarch 2.23.4-1.amzn2.0.1 amzn2-core 2.4 M
libsecret x86_64 0.18.5-2.amzn2.0.2 amzn2-core 153 k
perl-Error noarch 1:0.17020-2.amzn2 amzn2-core 32 k
perl-Git noarch 2.23.4-1.amzn2.0.1 amzn2-core 47 k
perl-TermReadKey x86_64 2.30-20.amzn2.0.2 amzn2-core 31 k
Transaction Summary
=====
Install 1 Package (+7 Dependent packages)

Total download size: 7.9 M
Installed size: 41 M
Downloading packages:
(1/8): emacs-filesystem-27.1-1.amzn2.0.1.noarch.rpm | 64 kB 00:00:00
(2/8): git-2.23.4-1.amzn2.0.1.x86_64.rpm | 135 kB 00:00:00
(3/8): git-core-doc-2.23.4-1.amzn2.0.1.noarch.rpm | 2.4 MB 00:00:00
```

Εικόνα 5-22: Εγκατάσταση Git από το τερματικό (1/2)

```

(4/8): libsecret-0.18.5-2.amzn2.0.2.x86_64.rpm | 153 kB 00:00:00
(5/8): perl-Error-0.17020-2.amzn2.noarch.rpm | 32 kB 00:00:00
(6/8): perl-Git-2.23.4-1.amzn2.0.1.noarch.rpm | 47 kB 00:00:00
(7/8): git-core-2.23.4-1.amzn2.0.1.x86_64.rpm | 5.0 MB 00:00:00
(8/8): perl-TermReadKey-2.30-20.amzn2.0.2.x86_64.rpm | 31 kB 00:00:00
-----
Total | 24 MB/s | 7.9 MB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : git-core-2.23.4-1.amzn2.0.1.x86_64 1/8
Installing : git-core-doc-2.23.4-1.amzn2.0.1.noarch 2/8
Installing : 1:emacs-filesystem-27.1-1.amzn2.0.1.noarch 3/8
Installing : 1:perl-Error-0.17020-2.amzn2.noarch 4/8
Installing : perl-TermReadKey-2.30-20.amzn2.0.2.x86_64 5/8
Installing : libsecret-0.18.5-2.amzn2.0.2.x86_64 6/8
Installing : perl-Git-2.23.4-1.amzn2.0.1.noarch 7/8
Installing : git-2.23.4-1.amzn2.0.1.x86_64 8/8
Verifying : libsecret-0.18.5-2.amzn2.0.2.x86_64 1/8
Verifying : perl-TermReadKey-2.30-20.amzn2.0.2.x86_64 2/8
Verifying : git-2.23.4-1.amzn2.0.1.x86_64 3/8
Verifying : 1:perl-Error-0.17020-2.amzn2.noarch 4/8
Verifying : git-core-doc-2.23.4-1.amzn2.0.1.noarch 5/8
Verifying : perl-Git-2.23.4-1.amzn2.0.1.noarch 6/8
Verifying : git-core-2.23.4-1.amzn2.0.1.x86_64 7/8
Verifying : 1:emacs-filesystem-27.1-1.amzn2.0.1.noarch 8/8

Installed:
git.x86_64 0:2.23.4-1.amzn2.0.1

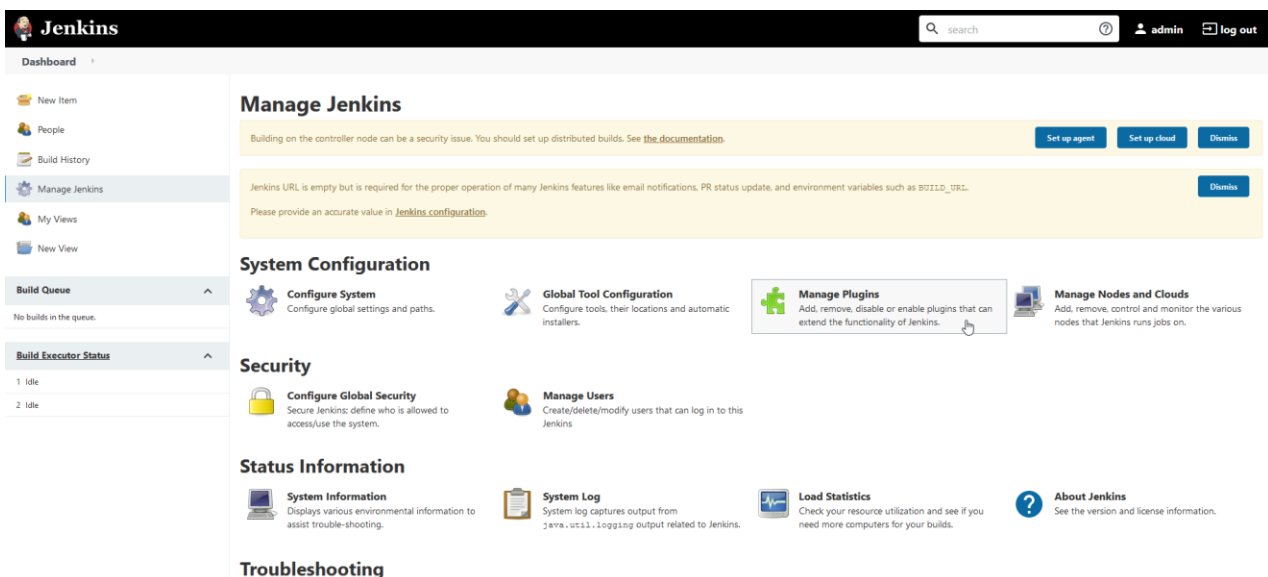
Dependency Installed:
emacs-filesystem.noarch 1:27.1-1.amzn2.0.1 git-core.x86_64 0:2.23.4-1.amzn2.0.1
git-core-doc.noarch 0:2.23.4-1.amzn2.0.1 libsecret.x86_64 0:0.18.5-2.amzn2.0.2
perl-Error.noarch 1:0.17020-2.amzn2 perl-Git.noarch 0:2.23.4-1.amzn2.0.1
perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2

Complete!

```

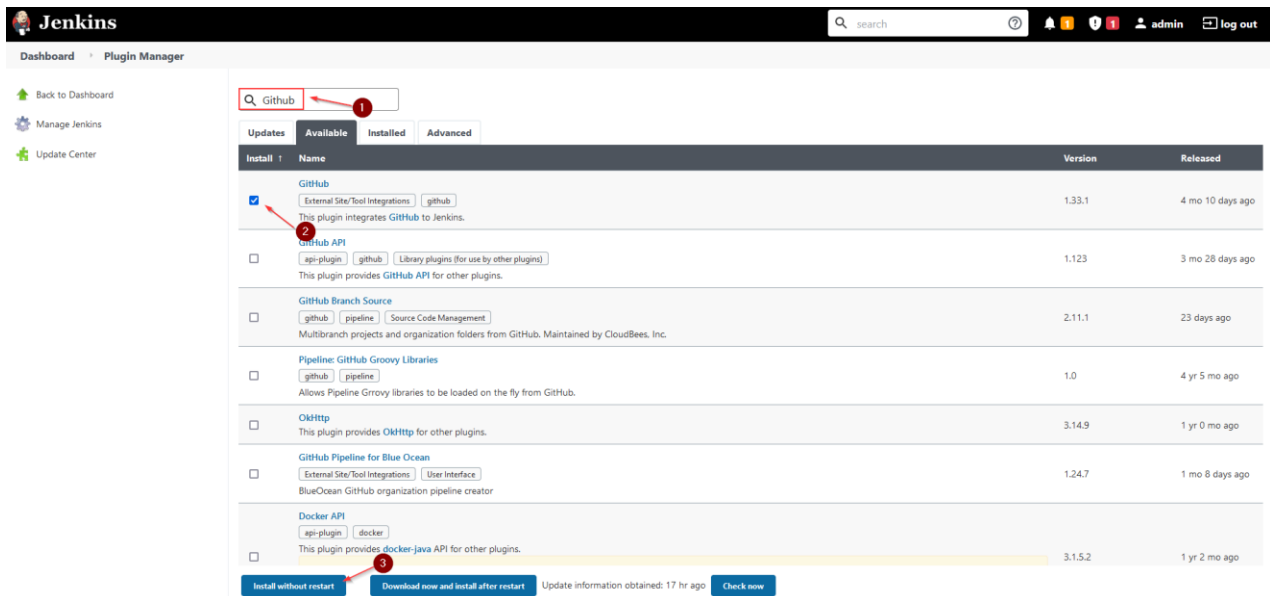
Εικόνα 5-23: Εγκατάσταση Git από το τερματικό(2/2)

- Έπειτα στο Jenkins γραφικό περιβάλλον (UI) κάνουμε εγκατάσταση τα plugins Manage Jenkins → Manage Plugins



Εικόνα 5-24: Διαχείριση Plugins

➤ GitHub → Install without restart



Εικόνα 5-25: Εγκατάσταση plugin GitHub

➤ Προσδιορίζουμε στο Global Tool Configuration το path: /usr/bin/git

🔧 Global Tool Configuration

Maven Configuration

Default settings provider

Use default maven settings

Default global settings provider

Use default maven global settings

JDK

JDK installations...

Git

Git installations

Git

Name

github

Path to Git executable ?

/usr/bin/git

☐ Install automatically ?

Delete Git

Save

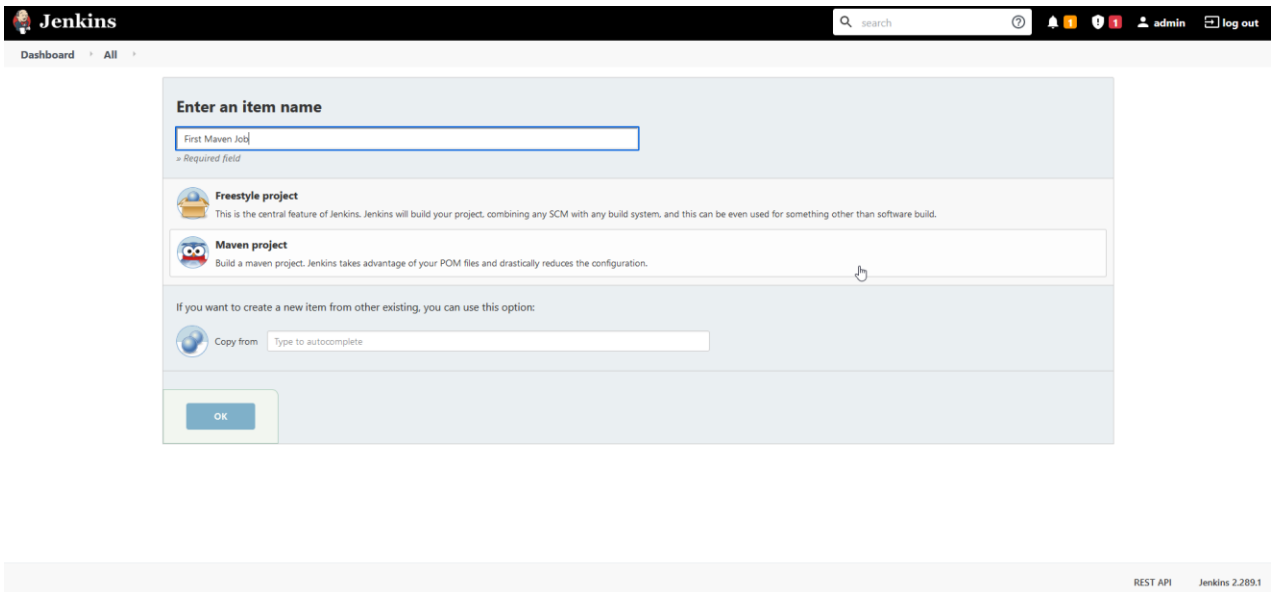
Apply

Εικόνα 5-26: Global Tool Configuration path

5.2.3 Εγκατάσταση Maven

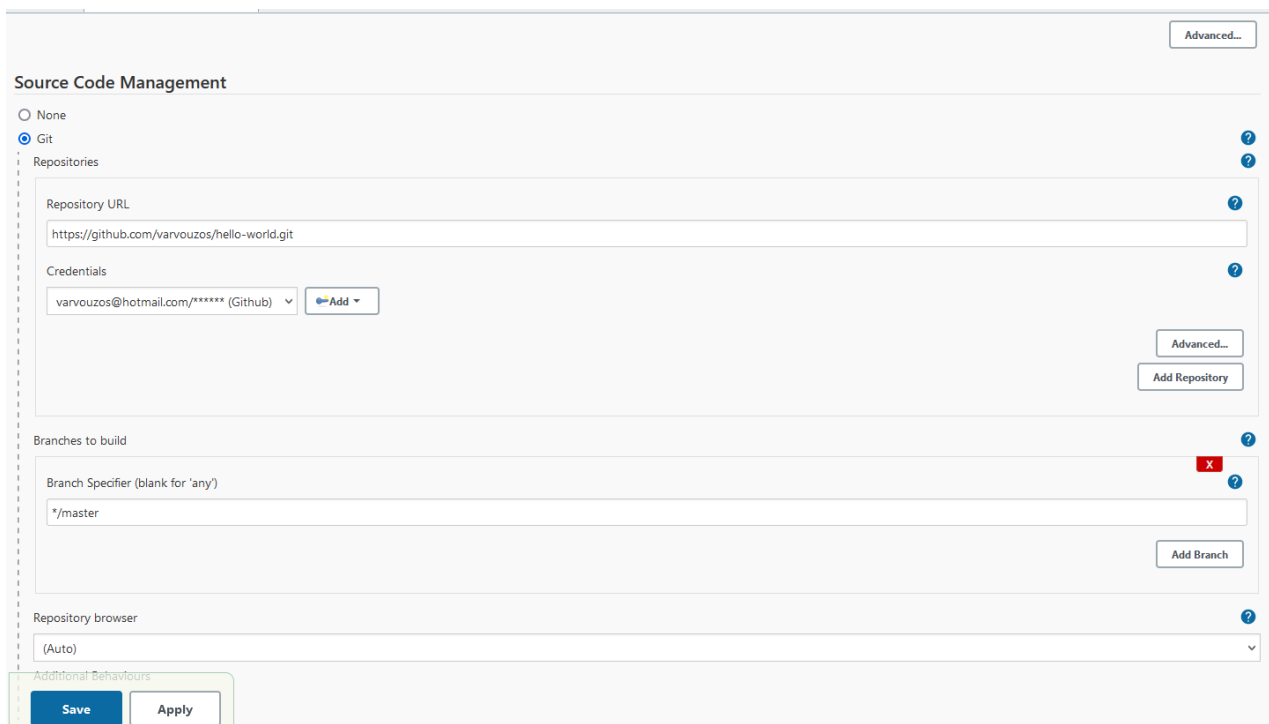
- Δημιουργία maven directory μέσω MobaXterm στο file path /opt
`mkdir /opt/maven`

- Για την δημιουργία ενός Maven Job πατάμε “New Item” από την αρχική οθόνη και έπειτα δίνουμε ένα όνομα “First Maven Job” και επιλέγουμε “Maven project”:



Εικόνα 5-28: Δημιουργία ενός Maven Job

- Εισάγουμε το GitHub repository URL και σε περίπτωση που δεν είναι public βάζουμε και credentials:



Εικόνα 5-29: GitHub repository URL

Εικόνα 5-30: Ρυθμίσεις Build

5.3 Tomcat server στον αγωγό ανάπτυξης

Προαπαιτούμενο για την εγκατάσταση Tomcat, είναι ένα EC2 Instance με Java το οποίο θα δημιουργήσουμε με τις ίδιες οδηγίες που δημιουργήσαμε το Jenkins instance στο κεφάλαιο 5.2.1 χρησιμοποιώντας το ίδιο security group.

- Εγκατάσταση του Tomcat μέσω MobaXterm:

```
cd /opt
wget https://ftp.cc.uoc.gr/mirrors/apache/tomcat/tomcat-
8/v8.5.68/bin/apache-tomcat-8.5.68.tar.gz
tar -xvzf apache-tomcat-8.5.68.tar.gz
```

- Αδειοδότηση executing permissions για το startup.sh and shutdown.sh που είναι κάτω από το bin μέσω MobaXterm:

```
chmod +x /opt/apache-tomcat-8.5.68/bin/startup.sh
chmod +x /opt/apache-tomcat-8.5.68/bin/shutdown.sh
```

- Εισαγωγή σχολίου στο αρχείο context.xml για απενεργοποίηση του localhost:

```
find / -name context.xml

<!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"
      allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->
```

- Εισαγωγή ρόλων και credentials για login στο αρχείο tomcat-users.xml

```
cd conf
vi tomcat-users.xml

<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>
<user username="admin" password="admin" roles="manager-gui, manager-
script, manager-jmx, manager-status"/>
<user username="deployer" password="deployer" roles="manager-script"/>
<user username="tomcat" password="s3cret" roles="manager-gui"/>
```




- Με την Δημόσια IP του Tomcat στην πόρτα 8080 κάνουμε login με τον χρήστη “tomcat”:



Tomcat Web Application Manager

Message: OK

Manager

[List Applications](#) [HTML Manager Help](#) [Manager Help](#) [Server Status](#)

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file path:

WAR or Directory path:

WAR file to deploy

Εικόνα 5-31: Είσοδος στον Tomcat

- Εγκατάσταση του “Deploy to Container” plugin στον Jenkins:

Jenkins [admin](#) [log out](#)

Dashboard **Plugin Manager**

[Back to Dashboard](#) [Manage Jenkins](#) [Update Center](#)

Updates **Available** **Installed** **Advanced**

Install	Name	Version	Released
☒	Deploy to container Artifact Uploader This plugin allows you to deploy a war to a container after a successful build. Glassfish 3.x remote deployment	1.16	7 mo 27 days ago

Εικόνα 5-32: Εγκατάσταση του Deploy to Container

- Δημιουργία του Maven Jenkins Job:

Jenkins [admin](#) [log out](#)

Dashboard **All**

Enter an item name

Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

If you want to create a new item from other existing, you can use this option:

Εικόνα 5-33: Δημιουργία του Maven Jenkins Job

➤ Configuration του Jenkins job:

Build

Root POM ?
pom.xml

Goals and options ?
clean install package

Post Steps

Aggregate downstream test results
Archive the artifacts
Build other projects
Deploy artifacts to Maven repository
Maven Invoker Plugin Results
Record fingerprints of files to track usage
Git Publisher
Deploy war/ear to a container
Set GitHub commit status (universal)
Set build status on GitHub commit (deprecated)
Add post-build action

Run regardless of build result
Run regardless of build result
Run regardless of build result
Run regardless of build result
Run regardless of build result
Run regardless of build result
Run regardless of build result
Run regardless of build result

Save Apply

Εικόνα 5-34: Configuration του Jenkins job (1/2)

Deploy war/ear to a container ?

WAR/EAR files ?
**/*.war

Context path ?

Containers

Tomcat 8.x Remote ?

Credentials
deployer/* (user to deploy on tomcat vm) Add

Tomcat URL ?
http://3.10.199.1:8080

Add Container

☐ Deploy on failure

Add post-build action

Save Apply

Εικόνα 5-35: Configuration του Jenkins job (2/2)

- Μετά από επιτυχημένο build μπορούμε να κάνουμε έλεγχο στην Public IP του Tomcat πόρτα 8080/webapp:

Hello, Welcome to DevOps!!

This is my first deployment

It works

Εικόνα 5-36: Σελίδα του webapp που κάναμε Deploy

6

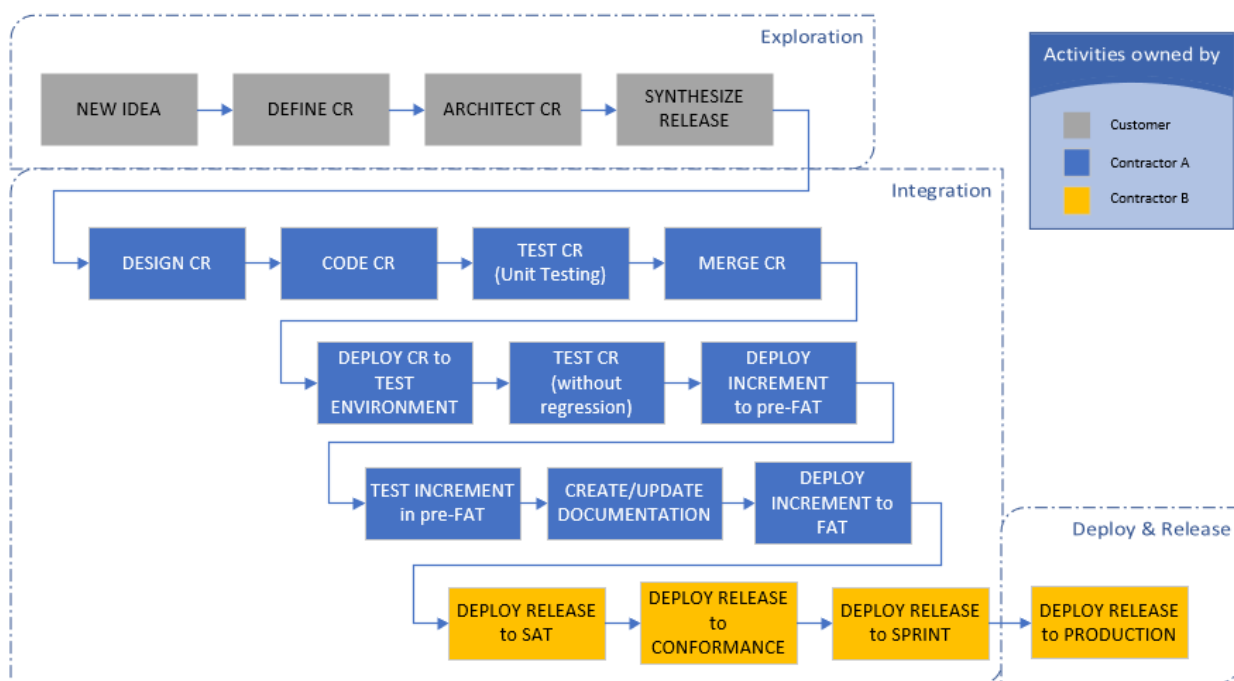
Μελέτη Περίπτωσης με SAFe

Το κεφάλαιο της μελέτης περίπτωσης εφαρμόζει τα εργαλεία που αναφέρονται στο κεφάλαιο 3.5 σε πραγματικές συνθήκες. Η καταγραφή έγινε σε εταιρία ανάπτυξης λογισμικού για λογαριασμό τρίτων μέσω κοινοπραξίας.

Ο σκοπός της μελέτης περίπτωσης είναι η εφαρμογή των εργαλείων σε πραγματικές συνθήκες ώστε να εξάγουμε αποτελέσματα αναφορικά με το DevOps στην παράδοση λογισμικού καθώς και να γίνει η σύνδεση με τα θεωρητικά κομμάτια και να προταθούν και βελτιώσεις βάση αυτών.

6.1 Χαρτογράφηση Αγωγού Ανάπτυξης

Το πρώτο βήμα για την βελτίωση των αξιών της ροής είναι η χαρτογράφηση του τρέχοντος αγωγού. Στην μελέτη περίπτωσης υλοποιήθηκε χαρτογράφηση των σταδίων και αποτύπωση με διαφορετικό χρώμα των υπεύθυνων ομάδων προς παράδοση του κάθε σταδίου (Εικόνα 4-1). Στο παράδειγμα μας έχουμε τρεις βασικούς εμπλεκόμενους για την παράδοση των σταδίων και είναι ο πλάτης και δυο ανάδοχοι έργου μέσω κοινοπραξίας. Για την μελέτη περίπτωσης και την χαρτογράφηση έχουμε εστιάσει σε δραστηριότητες του ανάδοχου έργου Α, χωρίς αυτό να σημαίνει ότι δεν θα προταθούν κάποιες γενικότερες βελτιώσεις που αφορούν ολόκληρο τον αγωγό ανάπτυξης.



Εικόνα 6-1: Χαρτογράφηση για την μελέτη περίπτωσης

6.2 Παραδοχές

Εικόνα 6-2: Μετρήσεις αγωγού ανάπτυξης ανάδοχου έργου A

και

Αποτελέσματα

Μετρήσεων

Έναρξη για τις μετρήσεις του ανάδοχου έργου A (Contractor A), με μπλε χρώμα στη εικόνα 4-1, είναι το στάδιο της σχεδίασης με έναυσμα τον καθορισμό των απαιτήσεων από τον πελάτη και τελικό στάδιο την επιτυχή παράδοση στο FAT περιβάλλον. Όλα τα στάδια τα οποία υπάρχει ανάμειξη από τον ανάδοχο έργου A είναι στην πτυχή της συνεχούς ενσωμάτωσης. Εμπλοκή υπάρχει από αναλυτές και στην πτυχή της εξερεύνησης με τον πελάτη αλλά δεν υπήρχαν στοιχεία προς μέτρηση.

Παραδοχές:

- Το μικρότερο στοιχείο που περνά από τον αγωγό ανάπτυξης είναι ένα αίτημα αλλαγής (CR).
- Ένα χαρακτηριστικό (feature) αποτελείται από ένα ή περισσότερα CR.
- Όλα τα αιτήματα αλλαγής (CR) δημιουργούν μια νέα έκδοση.
- Τα αιτήματα αλλαγής ορίζονται από τον πελάτη.
- Τα αιτήματα αλλαγής υλοποιούνται ανά διαστήματα **δυσεβδομάδων** (80 ώρες).
- Οι τελικές παραδόσεις γίνονται ανά **έξι μήνες** (940 ώρες).

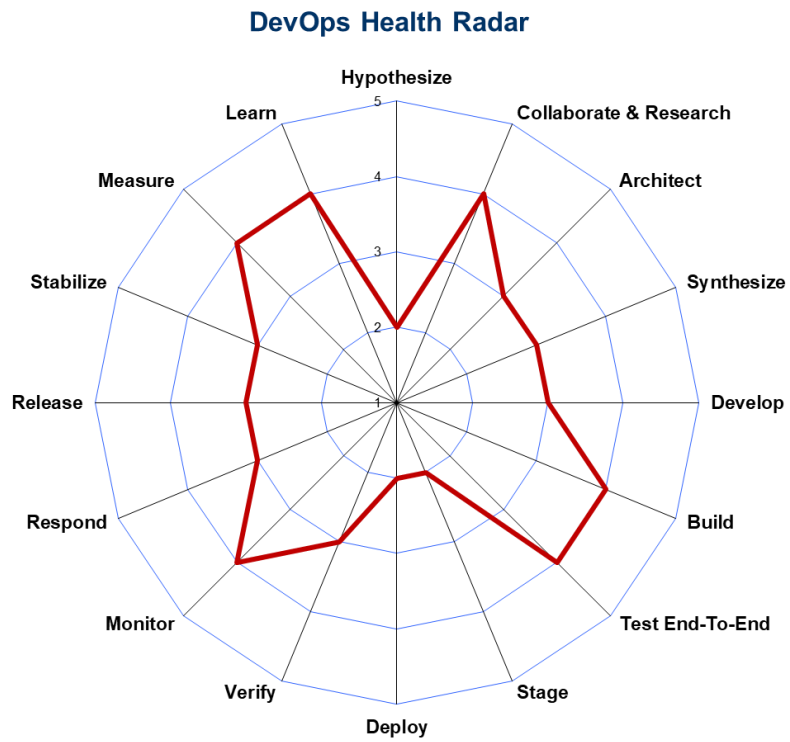
6.3 Μετρήσεις Health Radar

Το επόμενο βήμα μετά την χαρτογράφηση είναι η εκτέλεση της αξιολόγησης του DevOps Health Radar η οποία βρίσκεται με αναλυτική βαθμολογία και επεξηγήσεις στο [Παράρτημα](#) τμήμα i. Με αυτό το εργαλείο γίνεται η αξιολόγηση της κάθε δραστηριότητας από τις τέσσερις πτυχές. Η βαθμολογία κυμαίνεται από 1 έως 10 δεδομένης της επεξήγησης στην κάθε δραστηριότητα ξεχωριστά:

Step	Design CR	Code CR	Test CR (Unit Testing)	Merge CR	Deploy CR to test environment	Test CR (without regression)	Deploy increment to pre-FAT	Test increment in pre-FAT	Docs	Deploy to FAT	Total
Process Time (in hours)	24	75	2	4	8	4	8	8	16	8	157
Lead Time (in hours)	12	12	0	4	12	8	72	0	320	0	440
%Complete and Accurate	70%	70%	95%	95%	90%	97%	93%	97%	99%	99%	34%

Βαθμολογία	Τιμές
Sit (1-2)	1
Crawl (3-4)	2
Walk (5-6)	3
Run (7-8)	4
Fly (9-10)	5

Στην εικόνα 4-1 βλέπουμε τις τιμές που προέκυψαν έπειτα από την βαθμολογία της αξιολόγησης στο παράρτημα. Σε αυτή την αξιολόγηση συμπεριλήφθηκαν και βαθμολογήθηκαν όλες οι πτυχές.



Εικόνα 6-3: Τιμές Health Radar μετά την βαθμολόγηση

7 *Πρόγραμμα Προπτυχιακών Σπουδών (ΠΠΣ)*

Το κεφάλαιο με το πρόγραμμα προπτυχιακών σπουδών είναι ένα ενδεικτικό πρόγραμμα ενός κύκλου σπουδών που αποτελείται από θεωρητικό αλλά και πρακτικό μέρος.

Ο σκοπός του κεφαλαίου είναι να περιγράψει ένα πρόγραμμα βάση της ανωτέρω διπλωματικής εργασίας και να προετοιμάσει απόφοιτους πληροφορικής για πιθανές θέσεις μηχανικών DevOps.

7.1 Συνοπτική περιγραφή προγράμματος

7.1.1 Σε ποιους απευθύνεται

Το πρόγραμμα απευθύνεται σε προπτυχιακούς φοιτητές τμημάτων Πληροφορικής

7.1.2 Σκοπός και Προσδοκώμενα Αποτελέσματα

Σκοπός του προγράμματος είναι:

- να επιμορφώσει τους φοιτητές με τις τεχνολογικές εξελίξεις σχετικά με την ανάπτυξη λογισμικού.
- να καταρτίσει τους φοιτητές με τις σύγχρονες μεθοδολογίες (π.χ. Ευέλικτη), τα εργαλεία και τεχνικές για σχεδίαση και υλοποίηση τεχνολογιών συνεχούς ενσωμάτωσης/συνεχούς παράδοσης.
- να προετοιμάσει τους φοιτητές για πιθανές θέσεις στην αγορά εργασίας μηχανικών DevOps.

7.1.3 Προαπαιτούμενα

Προαπαιτούμενες γνώσεις:

- Βασικές γνώσεις προγραμματισμού
- Βασικές γνώσεις Linux
- Βασικές γνώσεις βάσεων δεδομένων

7.2 Μαθήματα

7.2.1 Εισαγωγή στα DevOps (Θεωρητικό Κομμάτι)

7.2.1.1 Θεματική Ενότητα: Αρχές DevOps

Διάλεξη 1

Στα πλαίσια της θεματικής ενότητας για τις Αρχές DevOps, κατά την πρώτη διάλεξη θα γίνει ιστορική αναδρομή και θα παρουσιαστεί η έννοια της Ευέλικτης Ανάπτυξης Λογισμικού (Agile Software Development) και οι μεθοδολογίες που στηρίχτηκε το DevOps. Συγκεκριμένα θα αναφερθούν το Lean Movement, το Agile Manifest, το Agile Movement, το Continuous Delivery Movement, το Toyota Kata.

Διάλεξη 2

Στην δεύτερη διάλεξη θα γίνει αναφορά στα τρία μονοπάτια: Αρχές Ροής, Αρχές Ανατροφοδότησης και Συνεχής Μάθηση και Πειραματισμός. Αναλυτικότερα θα δοθούν τα συστατικά στοιχεία του πρώτου μονοπατιού: Ορατότητα εργασίας, Περιορισμός εργασίας σε εξέλιξη και μείωση των εξαρτήσεων μεταξύ των ομάδων.

Διάλεξη 3

Σε συνέχεια από την προηγούμενη διάλεξη θα αναλυθεί το δεύτερο μονοπάτι DevOps και συγκεκριμένα: η ασφαλής εργασία σε πολύπλοκα συστήματα, ο έγκαιρος εντοπισμός προβλημάτων, ο διαμοιρασμός της γνώσης μέσω εντοπισμού προβλημάτων και η εισαγωγή ποιοτικών ελέγχων σε περισσότερα τμήματα.

Διάλεξη 4

Σε αυτή την διάλεξη ολοκληρώνεται η θεματική ενότητα με το τρίτο μονοπάτι που περιλαμβάνει: την υιοθέτηση κουλτούρας για συνεχή μάθηση, τη θεσμοθέτηση βελτίωσης της καθημερινής εργασίας, τον μετασχηματισμό τοπικών ευρημάτων σε γενικότερες βελτιώσεις και την εισαγωγή μεθόδων ανθεκτικότητας.

7.2.1.2 Θεματική Ενότητα: Βέλτιστες Πρακτικές DevOps

Διάλεξη 1

Θα γίνει ανάλυση SWOT σχετικά με DevOps αφού επεξηγηθεί τι είναι το SWOT. Η συμπλήρωση των δυνατών και αδύνατων σημείων, οι ευκαιρίες και οι απειλές θα συζητηθούν και θα συμπληρωθούν κατά την διάρκεια της διάλεξης. Θα συζητηθούν θεμελιώδεις πρακτικές DevOps: ο τρόπος επικοινωνίας στην παραδοσιακή οργανωτική δομή και τα προβλήματα της, η διαχείριση διαμόρφωσης, η συνεχής ενσωμάτωση και η εφαρμογή στρατηγικής δοκιμών.

Διάλεξη 2

Κατά την δεύτερη διάλεξη θα αναλυθεί ο αγωγός ανάπτυξης και συγκεκριμένα: η ανατομία των αγωγών ανάπτυξης, η δόμηση και δέσμη ενεργειών ανάπτυξης, το στάδιο της δέσμευσης, η αυτοματοποιημένη αποδοχή δοκιμών και τέλος η ανάπτυξη και παράδοση της εφαρμογής.

Διάλεξη 3

Κατά την διάρκεια της τρίτης διάλεξης θα γίνει αναφορά στο οικοσύστημα συνεχούς παράδοσης: την διαχείριση υποδομών και περιβαλλόντων, το υπολογιστικό νέφος, την μονολιθική αρχιτεκτονική και την αρχιτεκτονική μικροϋπηρεσιών.

Διάλεξη 4

Στην συνέχεια θα αναπτυχθεί εκτενώς η μεθοδολογία SAFe DevOps: τις επτά ικανότητες, την χαρτογράφηση αξιών ροής με επεξήγηση των όρων και το DevOps Health Radar.

7.2.1.3 Θεματική Ενότητα: Σύσταση τμήματος DevOps

Διάλεξη 1

Στην πρώτη διάλεξη θα αναλυθούν οι δομές τμημάτων DevOps: οι εξωτερικές ομάδες, οι ενοποιημένες ομάδες ανάπτυξης και λειτουργιών, συνεργασία ομάδας ανάπτυξης και λειτουργιών, η υποδομή ως υπηρεσία, η εσωτερική ομάδα και οι μηχανικοί αξιοπιστίας ιστότοπου. Επίσης στη ίδια διάλεξη θα δοθούν τα κριτήρια επιλογής δομής DevOps: αποφυγή Big Bang, επίτευξη μικρών στόχων και γρήγορες νίκες, καθορισμός προϊόντος και υπηρεσίες Greenfield ή Brownfield, DevOps για μεγάλους οργανισμούς με μονολιθικά συστήματα και καθορισμός κατάλληλου συστήματος εγγραφών.

Διάλεξη 2

Στην δεύτερη διάλεξη θα αναλυθούν οι πιθανοί ρόλοι DevOps καθώς και η Δόμηση DevOps και Αρχιτεκτονική: Δόμηση που καθορίζει την παράδοση λογισμικού, οργανισμοί Matrix και Λειτουργικές ομάδες, ομάδες προσανατολισμένες σε προϊόντα και υπηρεσίες, δημιουργία ομάδας DevOps και Αρχιτεκτονικής λογισμικού και τέλος το μέγεθος μιας ομάδας DevOps.

Διάλεξη 3

Σε αυτή τη διάλεξη θα παρουσιαστεί η στελέχωση και το ανθρώπινο δυναμικό για DevOps με παραδείγματα από την σύσταση τμήματος σε πολυεθνική: ο στρατηγικός προγραμματισμός ανθρώπινων πόρων, η ανάλυση εργασίας, η προσέλκυση υποψηφίων, η επιλογή υποψηφίων, η εκπαίδευση και ανάπτυξη ανθρώπινων πόρων, η αξιολόγηση απόδοσης και η πολιτική και τα συστήματα αμοιβής.

7.2.2 Υλοποίηση παραδείγματος Αγωγού Ανάπτυξης (Πρακτικό κομμάτι)

7.2.2.1 Θεματική Ενότητα: Αγωγός ανάπτυξης CI/CD με GIT, Jenkins, Maven, Ansible

Διάλεξη 1

Στην πρώτη διάλεξη θα πραγματοποιηθεί το αρχικό setup της υποδομής, εγκατάσταση του MobaXterm και έπειτα θα γίνει η εγκατάσταση του Jenkins.

Διάλεξη 2

Στην δεύτερη διάλεξη θα γίνει η εγκατάσταση Git, Maven και των plugins στον Jenkins.

Διάλεξη 3

Σε αυτή τη διάλεξη θα γίνει η εγκατάσταση του Tomcat και οι περαιτέρω ρυθμίσεις.

Διάλεξη 4

Στην τέταρτη διάλεξη θα γίνει το setup της Ansible, θα δημιουργηθεί το πρώτο playbook το οποίο θα τρέξει μέσω Jenkins.

7.2.2.2 Θεματική ενότητα: Docker & Kubernetes

Διάλεξη 1

Στην πρώτη διάλεξη θα πραγματοποιηθεί το αρχικό setup του Docker και το integration με Jenkins και δημιουργία Jenkins job για να αντιγράφει τα artifacts στο DockerHost. Έπειτα θα δημιουργηθεί ένα Dockerfile.

Διάλεξη 2

Κατά την δεύτερη διάλεξη θα γίνει το setup του Ubuntu server για το Kubernetes καθώς και το setup του Kubernetes Cluster,

Διάλεξη 3

Σε αυτή τη διάλεξη θα γίνει deployment με χρήση kubectl εντολών. Θα δημιουργηθεί YAML file και θα γίνει και το integration του Kubernetes με Ansible.

Διάλεξη 4

Στην τελευταία διάλεξη θα δημιουργηθούν Jenkins jobs για τα deployments σε Kubernetes αλλά και για την δημιουργία των Docker images. Θα υλοποιηθεί το integration στο Jenkins με CI/CD jobs για το deployment στο Kubernetes.

8

Συμπεράσματα

Το τελευταίο κεφάλαιο της διπλωματικής περιλαμβάνει γενικές διαπιστώσεις, βελτιώσεις και σύνοψη. Στις προτεινόμενες βελτιώσεις γίνεται αναφορά των αποτελεσμάτων της μελέτης περίπτωσης και βάση των θεωρητικών κομματιών δίνονται οι πιθανές αλλαγές ώστε να βελτιωθεί ο κύκλος παράδοσης λογισμικού με την συνεισφορά των τεχνικών DevOps.

Ο σκοπός αυτού του κεφαλαίου είναι να συνοψίσει και να αναδείξει τα γενικότερα συμπεράσματα σχετικά με την εφαρμογή τεχνικών και πρακτικών DevOps σε πραγματικές συνθήκες και να κάνει την σύνδεση με τα θεωρητικά κομμάτια.

8.1 Γενικές Διαπιστώσεις

Ο αγωγός συνεχούς ανάπτυξης μέσω των φάσεων μοντέλου παράδοσης λογισμικού DevOps περιλαμβάνει ροές εργασιών, δραστηριότητες και αυτοματισμούς που απαιτούνται ώστε να καλυφθεί ένα κομμάτι λειτουργικότητας από την λήψη μίας ιδέας έως την παράδοση στον τελικό χρήστη. Ανεξαρτήτως οργανισμού και μεθοδολογίας που ακολουθείται οι πτυχές παραμένουν οι ίδιες. Για να επιτύχουμε όμως την συνεχή ανάπτυξη πρέπει να αναλυθεί το επίπεδο ωριμότητας όλων των πτυχών και των επιμέρους δραστηριοτήτων.

Το κενό που υπάρχει μεταξύ της ομάδας ανάπτυξης και της ομάδας λειτουργιών έρχεται να τα καλύψει η έννοια των DevOps, η οποία δεν είναι αποκλειστικά μια ομάδα μηχανικών αλλά η αλλαγή τρόπου σκέψης και η υιοθέτηση κουλτούρας που μας βγάζει από την εποχή των αυστηρών κανόνων και των άκαμπτων γραφειοκρατικών διαδικασιών.

8.2 Βελτιώσεις με Βάση τα Αποτελέσματα της Μελέτης

Σύμφωνα με την χαρτογράφηση που υλοποιήθηκε και τα αποτελέσματα (Εικόνα 4-2) της στο κεφάλαιο 4 βλέπουμε με μια πρώτη ματιά ότι υπάρχουν εμφανή προβλήματα και καθυστερήσεις που πρέπει να ληφθούν υπόψιν. Για παράδειγμα βλέπουμε ότι υπάρχει υψηλός χρόνος 320 ωρών για κάθε αίτημα αλλαγής επειδή η ολοκλήρωση των εγγράφων γίνεται έπειτα από την υλοποίηση όλων των αιτημάτων αλλαγής που έχουν προγραμματιστεί και διαρκεί δυο μήνες μέχρι να ξεκινήσει το επόμενο στάδιο που είναι η εκτέλεση των δοκιμών.

Step	Design CR	Code CR	Test CR (Unit Testing)	Merge CR	Deploy CR to test environment	Test CR (without regression)	Deploy increment to pre-FAT	Test increment in pre-FAT	Docs	Deploy to FAT	Total
Process Time (in hours)	24	75	2	4	8	4	8	8	16	8	157
Lead Time (in hours)	12	12	0	4	12	8	72	0	320	0	440
%Complete and Accurate	70%	70%	95%	95%	90%	97%	93%	97%	99%	99%	34%

Εικόνα 4-2: Μετρήσεις αγωγού ανάπτυξης ανάδοχου έργου Α

Επίσης με μια ματιά στο Health Radar καταλαβαίνουμε ότι πρέπει να εστιάσουμε αρχικά στην βελτίωση των δραστηριοτήτων με τις χαμηλότερες τιμές (Deploy, Stage, Hypothesize) και σταδιακά να επικεντρωθούμε και να βελτιώσουμε και τις υπόλοιπες.

Μερικές από τις διορθωτικές κινήσεις βάση των μετρήσεων είναι:

8.2.1 Αυτοματοποίηση της Διαδικασίας Ανάπτυξης

Για την υλοποίηση της αυτοματοποίησης είναι αναγκαία η χρήση κατάλληλων εργαλείων. Σχετική αναφορά έχει γίνει στο [\(κεφάλαιο 3.2.3\)](#) και αφορά το εργαλείο Jenkins το οποίο είναι συμβατό με τα περισσότερα εργαλεία ελέγχου εκδόσεων και μπορεί να μειώσει τον χρόνο της ανάπτυξη σε μόλις δυο ώρες.

Επιπλέον στην μέχρι τώρα διαδικασία δεν υπάρχει ο ρόλος του μηχανικού DevOps ο οποίος θα αναλάβει την υλοποίηση της αυτοματοποίησης και της συντήρησης μέσω των κατάλληλων εργαλείων. Η υλοποίηση μπορεί με εύκολο τρόπο να εφαρμοστεί σε όλα τα περιβάλλοντα και δεν δημιουργεί κάποια επιπλέον δουλειά για κανέναν από τους εμπλεκόμενους.

8.2.2 Εισαγωγή Μικροϋπηρεσιών

Εκτός από την βελτίωση που θα επιτευχθεί με την αυτοματοποίηση της διαδικασίας ανάπτυξης θα μπορούσε να επιτευχθεί και περαιτέρω βελτίωση με την εισαγωγή μικροϋπηρεσιών [\(κεφάλαιο 3.4.4\)](#). Με την υιοθέτησή της πρακτικής των μικροϋπηρεσιών στο στάδιο της αρχιτεκτονικής (Architecture) και σε συνδυασμό με μια αρχιτεκτονική βασισμένη σε υποδοχείς (Containers) όπως το Kubernetes ή το OpenShift, ο χρόνος που θα απαιτείται για την ανάπτυξη

εργασιών από το ένα περιβάλλον στο άλλο θα μειωθεί περαιτέρω σε λίγα δευτερόλεπτα χωρίς διακοπή λειτουργίας της υπηρεσίας.

8.2.3 Αυτοματοποίηση των Δοκιμών

Παρόλο που οι δοκιμές είναι ήδη αυτοματοποιημένες, ενδέχεται να υπάρξουν περαιτέρω βελτιώσεις σε αυτόν τον τομέα ([κεφάλαιο 3.3.4](#)). Για παράδειγμα, η εισαγωγή του Zalenium (<https://opensource.zalando.com/zalenium/>) επιτρέπει την ανάπτυξη ενός συμπλέγματος Selenium σε περιβάλλον Kubernetes που μπορεί να εκτελεστεί μετά από κάθε ανάπτυξη για να ελέγξει την ανάπτυξη και τις ενσωματώσεις της. Επιπλέον, η τεχνολογία Test Containers μπορεί να εισαχθεί στο Unit Testing για να επιτρέψει την προσθήκη υποδοχέων για τη δοκιμή των ενσωματώσεων των στοιχείων με άλλα στοιχεία χωρίς τη χρήση πλαστών υπηρεσιών, αλλά με τη χρήση των πραγματικών υπηρεσιών που εκτελούνται σε ένα συμπεριλαμβανόμενο υποδοχέα.

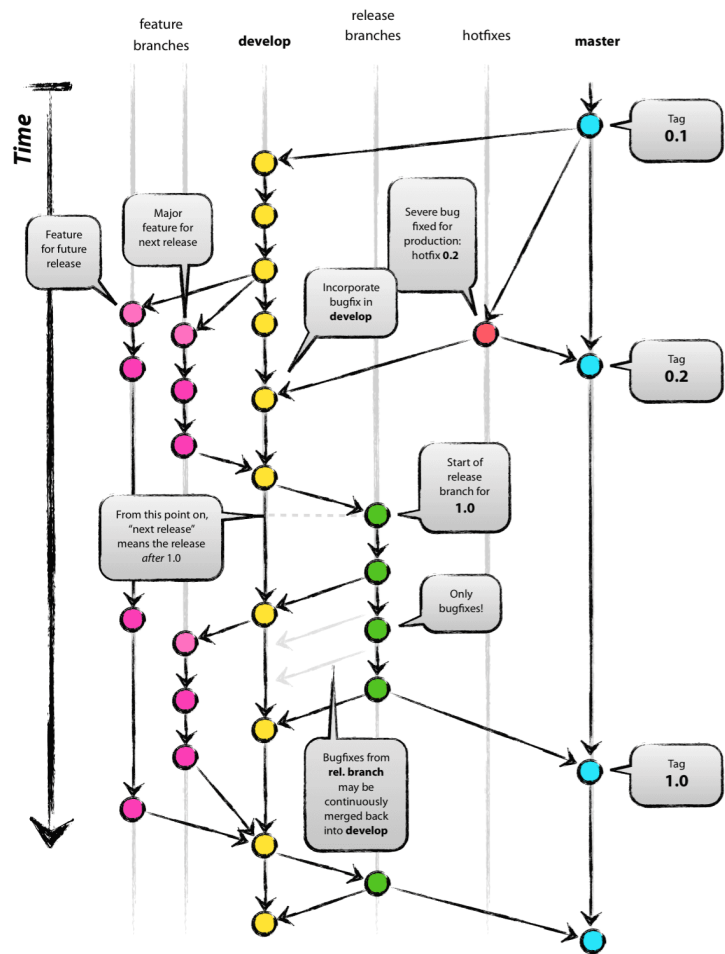
8.2.4 Πολλαπλές Παραδόσεις στο FAT Περιβάλλον

Με την αξιοποίηση της ευέλικτης παράδοσης και τις αρχές της ροής η μείωση των μεγεθών των παραδοτέων ([κεφάλαιο 2.3.3](#)) θα βελτιώσει την ροή από άκρη σε άκρη στον αγωγό παράδοσης. Για να επιτευχθεί αυτό πρέπει να αναλυθούν τα αιτήματα αλλαγών περαιτέρω σε νέα μικρότερα χαρακτηριστικά (features) που εξακολουθούν να εφαρμόζουν πολύτιμη λειτουργικότητα, αλλά έχουν μέγεθος ώστε να υλοποιηθούν εντός δύο εβδομάδων. Με αυτόν τον τρόπο, τα πρόσφατα καθορισμένα νέα χαρακτηριστικά μπορούν να περάσουν από τον αγωγό πολύ πιο γρήγορα, ενσωματώνονται κάθε δύο εβδομάδες και είναι διαθέσιμα στο περιβάλλον pre-FAT για ολοκληρωμένες λειτουργικές δοκιμές.

Οι προαναφερθείσες βελτιώσεις θα επέτρεπαν την κυκλοφορία σε περιβάλλον FAT πλήρως δοκιμασμένης, ολοκληρωμένης και τεκμηριωμένης λειτουργίας. Ένα πρόσθετο πλεονέκτημα είναι ότι τα παραδοτέα είναι διαθέσιμα σε όλους τους ενδιαφερόμενους για σκοπούς δοκιμών, συλλογής σχολίων και εκπαίδευσης.

8.2.5 Πολιτική Κλάδων (Branching Policy)

Στο [κεφάλαιο 3.2.2](#) έγινε αναφορά για τα εργαλεία GitLab και GitHub. Εκτός από τον έλεγχο έκδοσης πολύ σημαντικό είναι να ακολουθηθεί και μια πολιτική κλάδων που εκτός από τον κεντρικό κλάδο (master branch) παρέχονται και επιπλέον υποστηρικτικοί κλάδοι όπως αυτός της ανάπτυξης ή της επείγουσας επιδιόρθωσης (hotfix). Με αυτό τον τρόπο μπορούν να δουλεύουν ταυτόχρονα για το ίδιο μεγάλο χαρακτηριστικό και να έχουν διαθέσιμες τις τελευταίες αλλαγές που έχουν ανέβει μέχρι να ολοκληρωθεί η δουλειά όλων, να συγχωνευθεί και τελικά να συμπεριληφθεί με ετικέτα (tag) στον κύριο κλάδο.



Εικόνα 8-1: Πολιτική κλάδων στο GIT^[23]

8.2.6 Πιο Συχνή Παράδοση στο Παραγωγικό Περιβάλλον

Δεδομένου ότι η ολοκληρωμένη, πλήρως δοκιμασμένη και αποτυπωμένη σε έγγραφα λειτουργία θα είναι διαθέσιμη σε κάθε πιθανή επανάληψη (βελτίωση [κεφαλαίου 6.2.4](#)), θα δίνεται η δυνατότητα στον πελάτη να μπορεί να παρακολουθεί την πρόοδο κάθε αρχικά ζητούμενου αιτήματος αλλαγής (CR). Αυτό θα δίνει την δυνατότητα στον πελάτη να διαθέτει κατ' απαίτηση την λειτουργία καθώς η αλλαγή θα παραδίδεται σε όλα τα περιβάλλοντα μέσα σε λίγα λεπτά.

8.2.7 Αναβάθμιση των Παραδοτέων σε Μονολιθικά Συστήματα

Σε αντίθεση με τη νέα ανάπτυξη, τα Μονολιθικά ([κεφάλαιο 3.4.3](#)) συστήματα ακολουθούν την προσέγγιση καταρράκτη, όπου τα αιτήματα αλλαγών αντιμετωπίζονται ως μονολιθικό μπλοκ και οι ξεχωριστές φάσεις όπως ο σχεδιασμός, η συγγραφή του κώδικα, η δόμηση, η δοκιμή και η ανάπτυξη είναι καθαρά διαδοχικές. Αυτό δημιουργεί σημαντικές καθυστερήσεις σε κάθε μία από τις φάσεις και κληρονομεί τους κινδύνους των ελαττωμάτων που εντοπίζονται πολύ αργά στη διαδικασία, οι οποίες συνεπάγονται υψηλό κόστος για την αποκατάσταση και περαιτέρω καθυστερήσεις στην τελική απελευθέρωση της προγραμματισμένης εργασίας.

Από τη φύση τους, τέτοιες αναπτύξεις μπορούν να ωφεληθούν περισσότερο υιοθετώντας έναν πιο ευέλικτο κύκλο ζωής ανάπτυξης που θα επιτρέψει την επιτάχυνση των αξιών της ροής στον αγωγό παράδοσης. Η εισαγωγή περισσότερης ευελιξίας στις τελικές παραδόσεις (που ορίζεται επί του παρόντος σε δύο φορές το χρόνο) δεν είναι κάτι που ο πελάτης θα μπορούσε να αποδεχτεί εύκολα δεδομένου του κόστους υλοποίησης. Μακροπρόθεσμα όμως αυτό το κόστος είναι κέρδος καθώς δεν υπάρχουν καθυστερήσεις, απαιτούνται λιγότεροι άνθρωποι για τις δραστηριότητες και υπάρχει προστιθέμενη αξία στα παραδοτέα.

8.3 Σύνοψη

Συνοψίζοντας, εάν οι παραπάνω βελτιώσεις είχαν υιοθετηθεί και εξακολουθούσαν να χρησιμοποιούν τον αγωγό που παρουσιάστηκε προηγουμένως (Εικόνα 4-1), με βάση τα νέα δεδομένα ένα νέο χαρακτηριστικό:

- Θα απαιτούσε περίπου 5 ημέρες για να υλοποιηθεί και να δοκιμαστεί χωρίς παλινδρόμηση (χρόνος διαδικασίας)
- Θα χρειαζόταν κατά μέσο όρο συνολικά 8 ημέρες έως ότου το χαρακτηριστικό να δοκιμαστεί πραγματικά χωρίς παλινδρόμηση (χρόνος παράδοσης)
- Θα χρειαζόταν 10 ημέρες (μία επανάληψη/δύο εβδομάδες) για να φτάσει στο περιβάλλον pre-FAT για να ξεκινήσει η ολοκληρωμένη λειτουργική δοκιμή (πλήρης λειτουργική δοκιμή, δοκιμή παλινδρόμησης και δοκιμή απόδοσης).
- Θα χρειαζόταν 17 ημέρες (3,5 εβδομάδες) για να φτάσει στο FAT περιβάλλον.

Δεδομένου ότι για μια νέα έκδοση που απαιτεί 8 εβδομάδες ανάπτυξης, ο μέσος όρος CR έχει μέγεθος ώστε να ταιριάζει με την εφαρμογή του εντός τεσσάρων εβδομάδων, η πλήρης λειτουργικότητα CR θα ήταν διαθέσιμη στο FAT εντός 27 ημερών (αντί για 47 ημέρες) και θα ήταν εφικτή η ανάπτυξη στην παραγωγή μέσα σε λίγα λεπτά. Αυτά τα στοιχεία αφορούν μια νέα έκδοση 2,5 μηνών (μία έκδοση που απαιτεί δύο μήνες ανάπτυξης) και θα ήταν σημαντικά μεγαλύτερη για μια κυκλοφορία έξι μηνών. Σε σύγκριση με τον αρχικό αγωγό παράδοσης (Εικόνα 4-1), τα κέρδη απόδοσης είναι ήδη σημαντικά.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] Kim, G., Debois, P., Willis, J., Humble, J. and Allspaw, J., 2016. The Devops Handbook.
- [2] Vincent, R., Lively, K., Wang, A., & Harrison, D. 2019. Achieving DevOps: A Novel About Delivering the Best of Agile, DevOps, and Microservices.
- [3] Puppet.com. 2021. [online] Available at: <<https://puppet.com/resources/report/2020-state-of-devops-report/>> [Accessed 24 January 2021].
- [4] Skelton, M. and Pais, M., 2019. *Team topologies*.
- [5] Devopsdays.org. 2021. *About Devopsdays*. [online] Available at: <<https://devopsdays.org/about/>> [Accessed 1 February 2021].
- [6] Devops-certification.org. 2021. [online] Available at: <https://www.devops-certification.org/contents/DevOps_Revealed_by_International_DevOps_Certification_Academy.pdf> [Accessed 1 February 2021].
- [7] Kim, G., 2021. *The Three Ways: The Principles Underpinning DevOps - IT Revolution*. [online] IT Revolution. Available at: <<https://itrevolution.com/the-three-ways-principles-underpinning-devops/>> [Accessed 1 February 2021].
- [8] Plutora. 2021. *Understanding the Andon Cord as the Key to Lean Operations - Plutora*. [online] Available at: <<https://www.plutora.com/blog/andon-cord>> [Accessed 6 February 2021].
- [9] E. E. (Chip) Ponsford, D., 2021. *Andon Cords and Practice Environmentalism*. [online] Leanvets.com. Available at: <<https://www.leanvets.com/2018/05/andon-cords-and-practice.html>> [Accessed 6 February 2021].
- [10] Medium. 2021. *Getting Started With Kanban*. [online] Available at: <<https://neiltamplin.me/getting-started-with-kanban-eabdc3d0dc1>> [Accessed 1 February 2021].
- [11] Westrum, R., 2004. A typology of organisational cultures. *Quality and Safety in Health Care*, 13(suppl_2), pp.ii22-ii27.
- [12] Web.archive.org. 2021. *What is Kaizen | Definition of KAIZEN | Kaizen Meaning*. [online] Available at: <<https://web.archive.org/web/20131029204446/http://www.kaizen.com/about-us/definition-of-kaizen.html>> [Accessed 10 February 2021].
- [13] Wa.aws.amazon.com. 2021. *Game day - AWS Well-Architected Framework*. [online] Available at: <<https://wa.aws.amazon.com/wat.concept.gameday.en.html>> [Accessed 20 February 2021].
- [14] Netflix.github.io. 2021. *Home - Chaos Monkey*. [online] Available at: <<https://netflix.github.io/chaosmonkey/>> [Accessed 20 February 2021].
- [15] DevOps Institute. 2021. *8 Key Challenges in Adopting DevOps: Part 1 — DevOps Institute*. [online] Available at: <<https://devopsinstitute.com/8-key-challenges-in-adopting-devops-part-1/>> [Accessed 21 February 2021].
- [16] DevOps Institute. 2021. *8 Key Challenges in Adopting DevOps: Part 2 - Solutions — DevOps Institute*. [online] Available at: <<https://devopsinstitute.com/8-key-challenges-in-adopting-devops-part-2-solutions/>> [Accessed 21 February 2021].

- [17] Jez. and Farley., 2010. Continuous Delivery: Reliable Software Releases Through Build, Test, And Deployment Automation.
- [18] Jenkins. 2021. *Jenkins*. [online] Available at: <<https://www.jenkins.io/>> [Accessed 27 February 2021].
- [19] Tutorialspoint.com. 2021. *Agile Testing - Quadrants - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/agile_testing/agile_testing_quadrants.htm> [Accessed 27 February 2021].
- [20] Scaled Agile Framework. 2021. *DevOps - Scaled Agile Framework*. [online] Available at: <<https://www.scaledagileframework.com/devops/>> [Accessed 28 February 2021].
- [21] Kim, G., Behr, K. and Spafford, G., 2018. The Phoenix Project, 5Th Anniversary Edition. Portland, OR: IT Revolution Press.
- [22] Knaster, R., 2021. *Development Value Streams - Scaled Agile Framework*. [online] Scaled Agile Framework. Available at: <<https://www.scaledagileframework.com/development-value-streams/>> [Accessed 13 March 2021].
- [23] nvie.com. 2021. *A successful Git branching model*. [online] Available at: <<https://nvie.com/posts/a-successful-git-branching-model/>> [Accessed 14 March 2021].
- [24] Χυτήρης, Λ., (2001). Διοίκηση Ανθρώπινων Πόρων
- [25] J. Shah, D. Dubaria and J. Widhalm, "A Survey of DevOps tools for Networking," *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2018, pp. 185-188, doi: 10.1109/UEMCON.2018.8796814.

ΠΑΡΑΡΤΗΜΑ

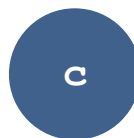
- i. Health Radar Score^[20]

Health Radar Score^[20]

		 © 2017 Scaled Agile, Inc. All rights reserved. DevOps Health Radar Agile Project Delivery Line: Customer's new development following agile methodology Date: 15/01/2021 Scoring: Sit (1-2), Crawl (3-4), Walk (5-6), Run (7-8), Fly (9-10)			
Dimension	Sub-dimension	Dimension / Question	Score	Rating	Comments
Continuous Exploration	Hypothesize	Hypothesizing entails expressing a business idea (epic) in terms of the business value it is expected to deliver. This hypothesis is then implemented as a Minimum Viable Product (MVP) through the Continuous Delivery Pipeline and evaluated for accuracy when released. Please rate your team's ability to translate business ideas into clear and measurable epic hypothesis statements. Sit (1-2): Ideas are vague or not defined. Crawl (3-4): Ideas are defined (e.g., as epics) but do not include hypothesis statements. Walk (5-6): Some ideas are expressed as hypothesis statements with measurable outcomes. Run (7-8): Most ideas are expressed as hypothesis statements with measurable outcomes and include an MVPs. Fly (9-10): All ideas are expressed as hypothesis statements with measurable outcomes and include an MVPs.	4.0	2	
Continuous Exploration	Collaborate & Research	Collaborate and Research involves Product Managers working directly with end-users, stakeholders and subject matter experts to understand customers' needs and identify specific business outcomes and associated metrics to guide solution delivery. Please rate your team's ability to collaborate with customer-side and IT-side experts to define Minimum Marketable Features (MMF) in support of the hypothesis. Sit (1-2): Product Management roles and responsibilities are not defined or followed. Crawl (3-4): Product Management creates requirements in large batches with little customer or development collaboration. Walk (5-6): Product Management collaborates with business-side or development-side experts, but not both, when defining requirements. Run (7-8): Product Management regularly collaborates with business-side, development-side, and operation-side experts but does not define Minimum Marketable Features. Fly (9-10): Product Management always collaborates with business-side, development-side, and operation-side experts and defines Minimum Marketable Features.	7.0	4	Continuous collaboration with End users, developers but no MMF defined
Continuous Exploration	Architect	Architecting for continuous delivery involves applying ""just enough"" intentional architecture to assure policy compliance without sacrificing product development flow, to ensure solutions are loosely coupled and to continuously pay down technical debt. Please rate your team's effectiveness at architecting for continuous delivery. Sit (1-2): Architecture is monolithic and fragile; it is difficult to change and involves managing complex dependencies across many components and systems. Crawl (3-4): Architecture is predominantly monolithic but some applications/systems are loosely coupled. Walk (5-6): Architectures is most decoupled but doesn't allow Release on Demand. Run (7-8): Architecture is aligned around value delivery and with few dependencies across components and systems. Fly (9-10): Architecture is built for Release on Demand and operability.	6.0	3	Have advanced from monolithic to loosely coupled components and applications, have the ability to release on demand (but do not do so due to the processes) with some dependencies

Continuous Exploration	Synthesize	Synthesizing involves combining the outputs of Hypothesize, Collaborate & Research and Architect to produce well-formed, prioritized features. These features then become the primary vehicle of value delivery through the remainder of the Continuous Delivery Pipeline. Please rate your team's ability to synthesize the results of Continuous Exploration activities into a well-crafted, prioritized, actionable feature backlog. Sit (1-2): The program backlog does not exist or is not shared. Crawl (3-4): The program backlog exists but the features are incomplete and prioritization is an afterthought. Walk (5-6): The program backlog contains fully defined features but are not prioritized using WSJF. Run (7-8): Features in the program backlog are complete, prioritized using WSJF and calibrated to the delivery capacity of the ART. Fly (9-10): The program backlog is a collection of Minimum Marketable Features created using BDD and prioritized using WSJF.	5.0	3	
Continuous Integration	Develop	Developing in the Continuous Delivery Pipeline involves splitting features into stories, implementing stories in vertical slices using Test-Driven Development (TDD), and committing changes to version control as they are made. Please rate your team's ability to quickly and reliably define and implement stories. Sit (1-2): The team backlog does not exist or is not used to manage daily work. Crawl (3-4): Stories are either incomplete or too verbose; unit tests are generally not written; peer reviews are not conducted. Walk (5-6): Stories are complete; most changes have unit tests; peer reviews are usually conducted. Run (7-8): Code is checked in daily; unit test coverage is 80%+; peer reviews are always conducted. Fly (9-10): Code is checked in multiple times per day; tests are written before code (TDD); pair work and other Built-in quality practices are the norm.	6.0	3	
Continuous Integration	Build	Build is triggered at the moment of check-in and involves compiling, unit testing (and other forms of component-level validation), successfully merging to trunk/main, committing to the repository and producing deploy-able artifacts. Please rate your team's effectiveness at building and integrating continuously. Sit (1-2): Builds are run fewer than once per iteration and/or are completely manual. Crawl (3-4): Builds are run once per iteration and are partially automated; dev branches are open for a month or more; builds break often. Walk (5-6): Automated builds run once a day; broken builds are corrected in 2-4 hours; manual unit tests are run against each build; dev branches are open for 2-4 weeks. Run (7-8): Builds run automatically upon code commit; broken builds are corrected within 1 hour; automated unit tests are run against each build; dev branches are merged to trunk every iteration. Fly (9-10): Builds run on every commit; builds include static code analysis and security testing; gated commits prevent defects from entering the version control; dev branches are merged to trunk on every commit."	8.0	4	Legacy systems are different. For the projects of the past couple of years 8,0 reflects the maturity including static code analysis. The dev branches are merged to the release branch (but only when the release branch is completed it is merged to the trunk)
Continuous Integration	Test End-To-End	Testing involves validating feature-level functionality in production-like environments. This typically includes functional testing, integration testing, regression testing, performance testing and exploratory testing. Please rate your team's effectiveness at testing continuously, end-to-end in production-like environments. Sit (1-2): Testing is performed manually in environments that do not mimic production; testing occurs in large batches during a scheduled "testing" phase Crawl (3-4): Testing is mostly manual in non-production-like environments; stories are implemented and tested independently within a single PI Walk (5-6): Half the testing is automated and performed in production-like, or production-simulated, environments every PI Run (7-8): The majority of tests are automated and run in production-like environments; stories are implemented and fully tested every iteration Fly (9-10): Successful builds trigger automatic deployment to production-like test environments; all tests are automated; tests run in parallel and changes are fully validated after every commit	7.0	4	Legacy systems are at 1,0 and new development are at 7,0 (Please note here that customer's terminology for this step is INTEGRATED FUNCTIONAL TESTING and not End-to-end testing as in the methodology)
Continuous Integration	Stage	Staging involves deploying features to a full copy of the production environment, from where they can be demonstrated to stakeholders, user acceptance tested and hosted for training purposes prior to production launch. Please rate your team's ability to stage features in full production-like (non-test) environments for final validation prior to production deployment. Sit (1-2): No staging environment exists or we use a test environment for staging. Crawl (3-4): Features are deployed manually to a staging environment once every PI. Walk (5-6): Features are deployed to a staging environment once per month and demonstrated to Product Management. Run (7-8): Features and infrastructure are auto-deployed to a staging environment every iteration and accepted by Product Management. Fly (9-10): Stories, changes and infrastructure are auto-deployed to a staging environment, validated, and immediately proceed to deployment.	4.0	2	It stops at 4,0 because only full releases reach the CONFORMANCE environment

Continuous Deployment	Deploy	Deployment is the actual migration of features into the production environment. Because the Continuous Delivery Pipeline separates deployment from release, deployed features are not assumed to be live to end users. Please rate your team's ability to continuously deploy features to production as well as the ability to control their visibility using feature toggles and/or other means. Sit (1-2): Features are deployed to production every 3+ months; deployments are manual and painful; "deployed" implies "released". Crawl (3-4): Features are deployed to production at PI boundaries; deployments are mostly manual; "deployed" implies "released". Walk (5-6): Features are deployed to production every iteration; deployments are mostly automated; some features can be deployed without being released. Run (7-8): Features are deployed to production every iteration and fully automated through the pipeline; dark releases are common. Fly (9-10): Features are deployed continuously throughout each iteration; Dev teams initiate deployments directly via pipeline tools; release is completely decoupled from deployment; dark releases are the norm.	3.0	2	PI consists with a release.
Continuous Deployment	Verify	Deployments must be verified for completeness and integrity before releasing to end users. Please rate your team's ability to accurately determine deployment success or failure and ability to roll back or fix forward as appropriate to correct deployment issues. Sit (1-2): Deployments are not verified in production before being released to end users. Crawl (3-4): Deployments are verified with manual smoke tests and/or UAT; we address deployment issues within a stated grace/triage/warranty period; we often correct issues directly in production. Walk (5-6): Deployments are verified with manual tests prior to releasing to end users; rolling back is painful or impossible; we do not make changes directly in production. Run (7-8): Deployments are verified using automated smoke tests, synthetic transactions and penetration tests prior to release; we can easily roll back or fix forward to recover from failed deployments. Fly (9-10): Automated production tests run on an ongoing basis and feed monitoring systems; failed deployments can be rolled back instantly or fixed forward through the entire pipeline.	5.0	3	
Continuous Deployment	Monitor	Monitoring implies that full-stack telemetry is active for all features deployed through the Continuous Delivery Pipeline so that system performance, end-user behavior, incidents and business value can be determined quickly and accurately in production. Please rate your team's effectiveness at monitoring the full solution stack (front end, middle tier, back end, infrastructure, etc.) and ability to analyze feature value based on these events. Sit (1-2): No feature level production monitoring exists; only infrastructure monitoring is in place. Crawl (3-4): Features only log faults and exceptions; analyzing events involves manually correlating logs from multiple systems. Walk (5-6): Features log faults, user activity and other events; data is analyzed manually to investigate incidents and measure business value of features. Run (7-8): Full-stack monitoring is in place; events can be correlated throughout the architecture; data is presented through system-specific dashboards. Fly (9-10): Federated monitoring platform provides one-stop access to full-stack insights; data is used to gauge system performance and business value.	8.0	4	For new applications is at 8.0. For older applications there is monitoring capability but not that mature
Continuous Deployment	Respond	The ability to detect and recover from unforeseen production incidents is critical to the Continuous Delivery Pipeline. Please rate your team's effectiveness at proactively detecting high severity production issues, identifying root causes using monitoring systems and quickly resolving issues by building, testing and deploying fixes through the pipeline (versus applying changes directly in production). Sit (1-2): Customers find issues before we do; resolving high priority issues is time consuming and reactive; customers have low confidence in our ability to recover from production issues. Crawl (3-4): Operations owns production issues; development involvement requires significant escalation; teams blame each other in times of crisis. Walk (5-6): Development and Operations collectively own the incident resolution process; recovering from major incidents is reactive but a team effort. Run (7-8): Our monitoring systems detect most issues before our customers do; Dev and Ops work proactively to recover from major incidents. Fly (9-10): Our monitoring systems alert us to dangerous conditions based on carefully-designed tolerance thresholds; Developers are responsible for supporting their own code and proactively issue fixes through the pipeline before users are affected.	5.0	3	Cooperation among Dev and Ops is good (no blame game)



Release on Demand	Release	Releasing involves making deployed features available to end users. Please rate your team's ability to release features to users on demand using feature toggles, blue/green environments, canary releases, etc. Sit (1-2): Releases are tightly coupled to deployments and customers are extremely dissatisfied with the frequency of releases. Crawl (3-4): Releases are tightly coupled to deployments but customers are somewhat dissatisfied with the frequency of releases. Walk (5-6): Release and deployment are coupled but both occur continuously or on demand. Run (7-8): Release is decoupled from deployment; deployed features are released to the end user population based on business readiness. Fly (9-10): Deployed features can be released to individual segments of the user population; feature toggles are refactored when no longer used.	5.0	3	End users are satisfied with the frequency
Release on Demand	Stabilize	The Continuous Delivery Pipeline requires the production environment to be continually stable, reliable, available, supportable and secure. Please rate your team's effectiveness at maintaining stable solutions that avoid unplanned down time and security breaches. Sit (1-2): We experience frequent unplanned outages and/or security breaches with long recovery times. Crawl (3-4): We experience occasional unplanned outages but recover within our service level agreements. Walk (5-6): We have very few unplanned outages; availability, security, and disaster recovery measures are effective. Run (7-8): We have no unplanned outages; We plan and rehearse failure and recovery. Fly (9-10): We maximize resiliency by deliberately injecting faults into our production environment and rehearsing recovery procedures.	6.0	3	Very few and with good coping mechanisms
Release on Demand	Measure	Measurement involves collecting factual information about the value of a deployed feature and evaluating it against the original hypothesis statement. Please rate your team's ability to collect objective information about the actual value realized by deployed features so that it can inform strategic financial decisions. Sit (1-2): We don't define or measure the value of features. Crawl (3-4): We've defined what 'value' is but don't know how to measure it. Walk (5-6): We capture qualitative feedback from the business about the value of our features. Run (7-8): We capture qualitative and quantitative feedback from the business and our monitoring systems about the value of our features. Fly (9-10): We aggregate the quantitative and qualitative feedback to objectively validate the original hypothesis and inform pivot-or-persevere decisions.	7.0	4	Data is collected, end users raise issues. This is performed in the closing phase
Release on Demand	Learn	Learning entails making a judgment call to validate or invalidate the original hypothesis based on objective measures of business value, system performance and customer feedback. Please rate your team's ability to make strategic, "pivot-or-persevere" decisions based on empirical performance data and commitment to actively applying those insights to continuously improve the pipeline. Sit (1-2): Features are never evaluated post-release. Crawl (3-4): Features are sometimes evaluated using subjective information and/or unilateral opinions. Walk (5-6): Hypotheses are evaluated using objective measures but actions are heavily influenced by corporate politics. Run (7-8): Hypotheses are objectively evaluated; pivot or persevere decisions are made without mercy or guilt. Fly (9-10): Continuous learning and experimentation are ingrained in the DNA of the organization.	7.0	4	