ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη υπηρεσιοστρεφούς πλατφόρμας παρακολούθησης και ελέγχου υποδομής υπολογιστικού νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φώτιος Σ. Καλαθόπουλος

Επιβλέπων : Ιάκωβος Στ. Βενιέρης
Καθηγητής Ε.Μ.Π

Αθήνα, Ιανουάριος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη υπηρεσιοστρεφούς πλατφόρμας παρακολούθησης και ελέγχου υποδομής υπολογιστικού νέφους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φώτιος Σ. Καλαθόπουλος

Επιβλέπων: Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Δευτέρα, 19 Ιανουαρίου 2015

……..	……..	……..
Ιάκωβος Βενιέρης	Δήμητρα Θεοδώρα	Νικόλαος Ουζούνογλου
Καθηγητής Ε.Μ.Π.	Κακλαμάνη	Καθηγητής Ε.Μ.Π.
	Καθηγήτρια Ε.Μ.Π.	

Αθήνα, Ιανουάριος 2015

……

Φώτιος Σ. Καλαθόπουλος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Φώτιος Σ. Καλαθόπουλος 2015.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.
Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Η παρούσα διπλωματική έχει ως σκοπό τη διευρένηση της τεχνολογίας του Cloud Computing καθώς και τη σχεδίαση και ανάπτυξη υπηρεσιοστρεφούς πλατφόρμας με σκοπό το monitoring σε συστήματα υπολογιστικού νέφους. Αρχικά γίνεται μια παρουσίαση της προέλευσης και ιστορίας του Cloud Computing, των χαρακτηριστικών του καθώς και αναφορά στα μοντέλα υπηρεσιών του. Επίσης αναφέρονται διάφορα πρακτικά παραδείγματα τεχνολογιών στα οποία έχει άμεση εφαρμογή το «νέφος». Στη συνέχεια παρουσιάζεται η αρχιτεκτονική και υλοποίηση της client-server πλατφόρμας που υλοποίηθηκε με σκοπό το monitoring σε cloud σύστημα. Τέλος μέσω μιας πειραματικής διαδικασίας βασιζόμενης σε πραγματικό σύστημα υπολογιστικού νέφους συλλέγονται δεδομένα με σκοπό την εξέταση σεναρίων και την παρακολούθηση της συμπεριφοράς του συστήματος. Συμπληρωματικά παρατίθενται πληροφορίες σχετικά με τις τεχνολογίες που χρησιμοποίηθηκαν καθώς και ο κώδικας του project πάνω στο οποίο βασίστηκε η δημιουργία της πλατφόρμας.

Abstract

The scope of this thesis is to study and investigate the technology of Cloud Computing as well as the design and implementation of a service-oriented platform in cloud systems. Initially, we start with a presentation of the origin and history of Cloud Computing, a description of its functional characteristics follows and in the end a reference to its service models is made. Furthermore various practical examples of technologies in which cloud has direct application are presented. Next follows the architecture and implementation of the client-server platform that was implemented with the purpose of monitoring in a cloud system. Finally through an experimental procedure depending on a real cloud computing system, data is collected with the aim of testing various scenarios and monitoring the behavior of the system. There is an additional section with references to technologies that were used as well as the software code of the project on which the platform implementation was based.

Keywords:

cloud computing, monitoring, virtual machine, server, database, controller, service

Ευχαριστίες

Με την ευκαιρία της εκπόνησης της διπλωματικής μου εργασίας και της ολοκλήρωσης του κύκλου σπουδών μου στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχ. Η/Υ του Ε.Μ.Π. , νιώθω την ανάγκη να ευχαριστήσω όλους τους ανθρώπους που με βοήθησαν να φτάσω μέχρι αυτό το σημείο.

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή μου Ιάκωβο Στ. Βενιέρη, για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα αντικείμενο το οποίο είναι πολύ επίκαιρο τη σημερινή εποχή και το οποίο έπαιξε και θα συνεχίζει να παίζει σπουδαίο ρόλο στην επαγγελματική μου σταδιοδρομία. Τον ευχαριστώ και για το αίσθημα ικανοποίησης που ένιωσα αφού μου δόθηκε η δυνατότητα να επεκτείνω και να εμπλουτίσω τις γνώσεις μου στον προγραμματισμό και να νιώσω ο ίδιος πιο ανταγωνιστικός.

Ιδιαίτερες ευχαριστίες θα ήθελα να απευθύνω στον Υποψήφιο Διδάκτορα Ανδρέα Καψάλη, χωρίς την πολύτιμη και αστείρευτη βοήθεια του οποίου θα ήταν αδύνατη η διεκπαιρέωση της διπλωματικής μου εργασίας. Η συμβολή του στη συγγραφή της εργασίας καθώς και στην ανάπτυξη της εφαρμογής υπήρξε ζωτικής σημασίας.

Επίσης θα ήθελα να ευχαριστήσω όλους τους φίλους μου, στενούς και μη, συμφοιτητές και μη, που ήταν κοντά μου, με στήριζαν και με τους οποίους πέρασα 6 όμορφα φοιτητικά χρόνια. Τους ευχαριστώ για όσα περάσαμε μαζί και για όσα θα θυμόμαστε.

Τέλος, θέλω να ευχαριστήσω ιδιαίτερα τους γονείς μου, Ελένη και Στέλιο. Χωρίς τους δύο αυτούς ανθρώπους δε θα έφτανα εως εδώ. Αδιάκοπα τα τελευταία 24 χρόνια, στήριζαν τα όνειρα και τις φιλοδοξίες μου, μου έμαθαν να αγωνίζομαι και να γίνομαι καλύτερος. Πάντα ήταν δίπλα μου. Επίσης δε θα έλειπε από τις ευχαριστίες και η υπόλοιπη οικογένεια μου που πάντα ήταν δίπλα μου και με έσπρωχναν να κυνηγάω τα όνειρά μου.

Ιδιαίτερη αναφορά θα ήθελα να κάνω στον καθηγητή μου στο Λύκειο Μόδεστο Τριπολιτσιώτη, ο οποίος ουσιαστικά ήταν εκείνος που άναψε το «φιτίλι» για την σημερινή μου πορεία. Εκείνος με ενέπνευσε και πίστεψε στο τι μπορω να καταφέρω. Ήταν ο μέντοράς μου.

Table of Contents

Εισαγωγή

Σήμερα, το cloud computing έχει κατακτήσει τον επιχειρηματικό κι όχι μόνο κόσμο, ενώ χρησιμοποιείται από την πλειονότητα των χρηστών του διαδικτύου έχοντας για τα καλά μπει στην καθημερινότητα των ανθρώπων. Υποστηρίζεται ότι το cloud θα αποτελέσει το μέλλον της επιχειρηματικής τεχνολογίας, προσφέροντας εντυπωσιακά οικονομικά αποτελέσματα. Η ανάγκη για νέες τεχνολογίες που προσφέρουν ασφάλεια στην αποθήκευση και ευκολία στην πρόσβαση, είναι οι κυριότεροι λόγοι που αναπτύχθηκε ιδιαίτερα αυτός ο τομέας.

Ως εκ τούτου, οργανισμοί , ο ένας μετά τον άλλον, μεταφέρουν το software τους στο cloud. Πληθώρα εφαρμογών έχουν τη βάση τους στον νέφος. Σήμερα κατά κύριο λόγο υπάρχουν 3 μοντέλα υπηρεσιών, προσφέροντας διαφορετικές δυνατότητες το καθένα. Αυτά είναι: IaaS(Infrastructure-as-a-Service), PaaS(Platform-as-a-Service), SaaS (Software-as-a-Service).

Ένας cloud provider παρέχει υποδομές διαχείρισης πόρων σε πολλαπλούς χρήστες. Το φαινόμενο αυτό ονομάζεται multitenancy και αποτελεί την ουσία των cloud computing υπηρεσιών. Ως αποτέλεσμα προκύπτει ανάγκη για αποδοτική διαχείριση των συστημάτων υπολογιστικού νέφους. Οι αυξανόμενες ανάγκες σε πόρους καθώς και η εκθετική αύξηση των χρηστών που επιθυμούν να επωφεληθούν των πλεονεκτημάτων που προσφέρει το cloud έχει οδηγήσει τους παρόχους σε αναζήτηση μεθόδων για σωστό management.

Κάθε πάροχος cloud computing υπηρεσιών διαθέτει στους πελάτες του και σχετικό web interface για τη διαχείριση των εικονικών μηχανών, δίσκων και δικτύων που τους παρέχει, καθώς και κάποιο προγραμματιστικό API για την ανάπτυξη εναλλακτικών εφαρμογών διαχείρισης. Συνεπώς προκύπτει ανάγκη για monitoring των cloud συστημάτων. Υπάρχουν πολλαπλά εργαλεία monitoring στην αγορά όπως το Nagios με δυνατότητα σχολαστικής και συνεχούς παρακολούθησης της λειτουργίας των υποδομών. Το monitoring επιτρέπει την έγκαιρη επίλυση σημαντικών προβλημάτων που μπορούν να παρουσιαστούν προλαβαίνοντας τυχούσα αρνητική επίδραση στην παροχή

των υπηρεσιών στους τελικούς χρήστες. Στην πλευρά του παρόχου γίνεται έλεγχος του hardware & software που έχει στη διάθεσή του ενώ στην πλευρά του «καταναλωτή» παρέχεται πληροφορία για την απόδοση της πλατφόρμας και των υπηρεσιών. Μέσω του monitoring ο πάροχος εγγυάται το απαιτούμενο QoS (quality-of-service) που έχει συμφωνηθεί ενώ παράλληλα το κόστος εξαιτίας προβλημάτων μειώνεται δραματικά. Τέλος ο πάροχος πρέπει να χει λάβει υπόψη του "critical" καταστάσεις και να χει στην εργαλειοθήκη του άμεσες λύσεις και προτάσεις για την έγκαιρη αντιμετώπιση τους.

2.1 Τι είναι το cloud computing;

2.1.1 Ορισμός του Cloud Computing

Υπάρχουν αμέτρητοι ορισμοί και ερμηνείες για τον όρο “Cloud Computing” ή αλλιώς Υπολογιστικό Νέφος που μπορούν να ευρεθούν από ποικίλες πηγές. Ο όρος “υπολογιστικό νέφος” προέρχεται πιθανώς από διαγράμματα δικτύων όπου το σχήμα του συννέφου χρησιμοποίειται για να απεικονίσει συγκεκριμένα είδη δικτύων όπως το Internet ή εσωτερικά δίκτυα.

Μια απλοποιημένη ερμηνεία για το Cloud Computing θα μπορούσε να είναι η εξής: «Με τον όρο υπολογιστικό νέφος αναφερόμαστε στο σύνολο των εφαρμογών που παρέχονται δια μέσου Διαδικτύου σε συνδυασμό με τον εξοπλισμό και το λογισμικό των κέντρων βάσεων δεδομένων που παρέχουν αυτές τις εφαρμογές.»

Figure 1 To Cloud Computing προσφέρει ευρεία γκάμα εφαρμογών σε πολλαπλούς χρήστες [1]

Σύμφωνα με τον ορισμό του US NIST [1] ως Cloud Computing ορίζουμε το μοντέλο που επιτρέπει την εύκολη και on-demand (κατ' απαίτηση) πρόσβαση μέσω δικτύου σε ένα διαμοιραζόμενο σύνολο από παραμετροποιήσιμους υπολογιστικούς πόρους (πχ. δίκτυα, servers, αποθηκευτικός χώρος, υπηρεσίες, εφαρμογές) οι οποίοι μπορούν εύκολα και γρήγορα να παρακολουθηθούν και διατεθούν με ελάχιστη διαχειριστική παρέμβαση ή αλληλεπίδραση από τον πάροχο των υπηρεσιών.

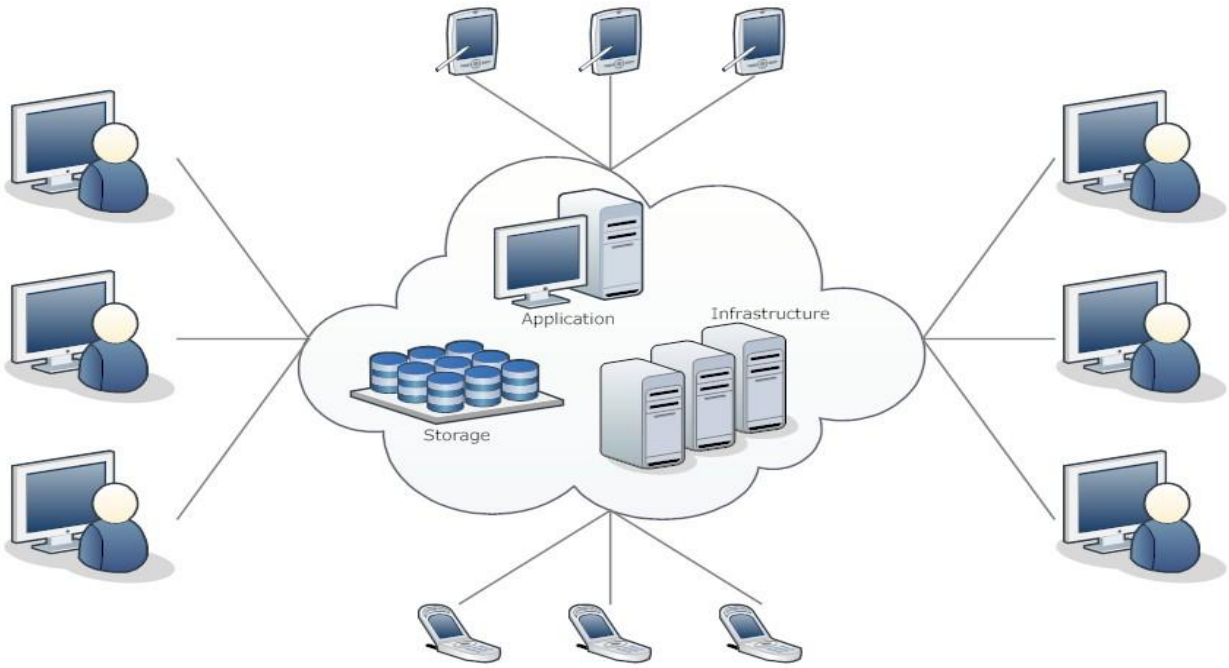


Figure 2 To Cloud = εφαρμογές+υποδομή+αποθηκευτικός χώρος+πρόσβαση από το σπίτι [2]

2.1.2 Βασικά Λειτουργικά Χαρακτηριστικά του Cloud Computing

Το μοντέλο του Cloud με βάση τον ορισμό του NIST απαρτίζεται απο 5 βασικά λειτουργικα χαρακτηριστικά [1] :

- On-demand self-service

Ο καταναλωτής μπορεί να ζητήσει μονομερώς υπολογιστικές δυνατότητες, όπως ο χρόνος χρησιμοποίησης του server και το μέγεθος του αποθηκευτικού χώρου που θα χρησιμοποιηθεί αυτόματα μέσω δικτύου, χωρίς να απαιτείται καμία ανθρώπινη αλληλεπίδραση με τον πάροχο της εκάστοτε υπηρεσίας.

- Broad network access

Οι παραπάνω δυνατότητες είναι προσβάσιμες από παντού δια μέσω δικτύου και διαδεδομένων μηχανισμών, πράγμα που επιτρέπει την χρήση και πρόσβαση τους από πολλές και ετερογενείς πλατφόρμες χρήστη (π.χ. κινητά τηλέφωνα, φορητούς υπολογιστές, PDA).

- Resource pooling

Οι υπολογιστικοί πόροι του παρόχου συγκεντρώνονται σε κοινό σύνολο χρησιμοποιώντας ένα μοντέλο πολλών ενοικιαστών, με ποικίλους φυσικούς και εικονικούς πόρους οι οποίοι αποδίδονται δυναμικά κατ' απαίτηση χρήστη. Ο καταναλωτής γενικά δεν έχει κανένα έλεγχο ή γνώση της ακριβής τοποθεσίας των παρεχόμενων πόρων, αλλά μπορεί να δύναται να προσδιορίσει σε ένα πιο αφηρημένο επίπεδο την τοποθεσία όπως τη χώρα, την πόλη ή το data-center. Παραδείγματα τέτοιων πόρων είναι ο αποθηκευτικός χώρος, η επεξεργασία, η μνήμη, το εύρος ζώνης δικτύου, τα Virtual Machines.

- Rapid elasticity

Οι δυνατότητες αυτές μπορούν να παρακολουθηθούν και διατεθούν με ελαστικό τρόπο, μερικές φορές αυτόματα, αυξάνοντας ή μειώνοντας το μέγεθός τους κατ απαίτηση. Στον καταναλωτή – τελικό χρήστη οι δυνατότητες αυτές που είναι διαθέσιμες, συχνά μοιάζουν να είναι απεριοριστες και μπορούν να αγοραστούν – αποκτηθούν σε οποιαδήποτε ποσότητα, οποιαδήποτε στιγμή.

- Measured Service

Τα συστήματα Cloud αυτόματα ελέγχουν και βελτιστοποιούν τη χρήση των υπολογιστικών πόρων χρησιμοποιώντας κάποια μετρητικά συστήματα σε κάποιο από τα επίπεδα της αφαίρεσης που εισάγουν, κατάλληλο για την εκάστοτε παρεχόμενη υπηρεσία (αποθηκευτικού χώρου, υπολογιστικής ισχύος, εύρους ζώνης, ενεργού αριθμού χρηστών κλπ.). Η χρήση των πόρων μπορεί να παρακολουθηθεί, ελεγχθεί και να αναφερθεί, παρέχονας διαφάνεια και για τις δύο πλευρές, τελικού χρήστη – καταναλωτή και παρόχου της χρησιμοποιούμενης υπηρεσίας.

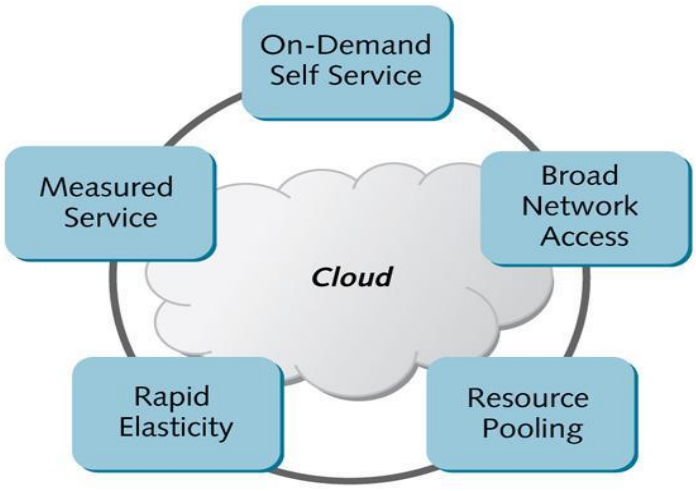


Figure 3 Τα βασικά χαρακτηριστικά του Cloud Computing [3]

2.1.3 Ιστορική εξέλιξη του Cloud Computing

Η ιδέα να παρέχεται μια κεντροποιημένη υπολογιστική υπηρεσία χρονολογείται πίσω στη δεκαετία του '60 , όπου οι υπολογιστικές υπηρεσίες παρέχονταν μέσω δικτύου με τη χρήση μηχανισμού κεντρικής time-sharing (χρονομεριστικής) τεχνολογίας. Ο μηχανισμός αυτός χρησιμοποιήθηκε με σκοπό την αποδοτικότερη χρήση των κοστοβόρων κεντρικών μονάδων. Ως αποτέλεσμα, πολλαπλοί χρήστες μοιράζονταν τόσο φυσική πρόσβαση όσο και CPU-time (χρόνο πυρήνα) μέσω της χρήσης τερματικών που δεν είχαν δικές τους επεξεργαστικές δυνατότητες. [2]

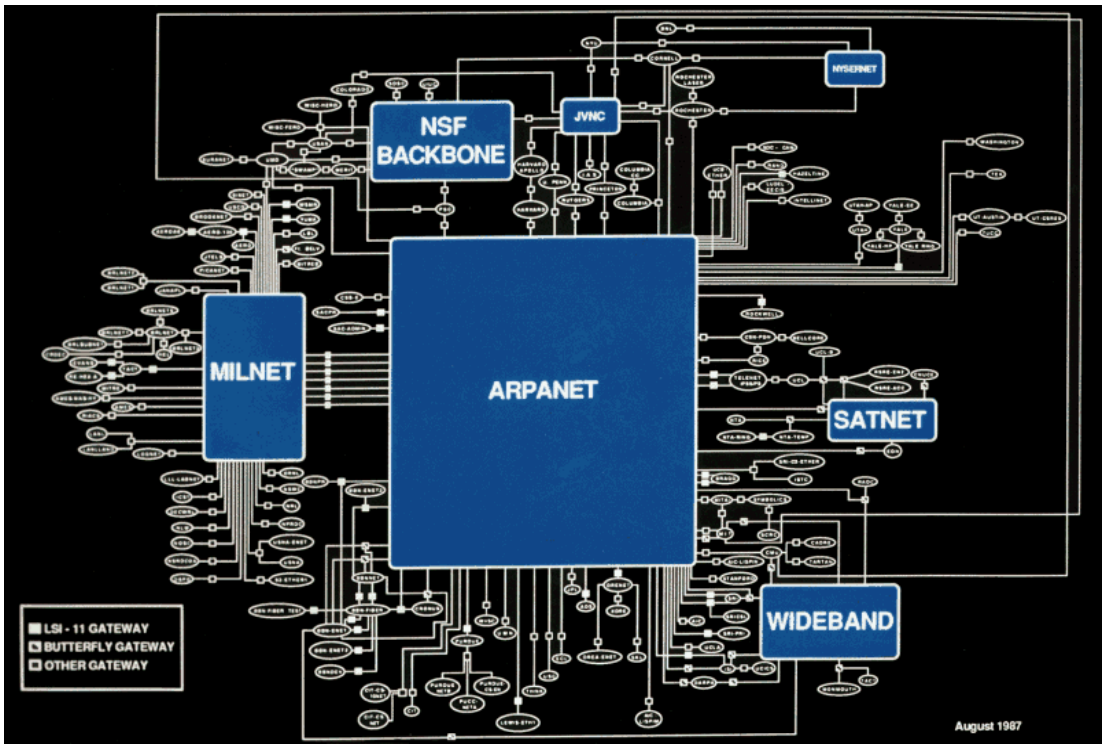


Figure 4 ARPANET architecture – core of the Internet in August 1987 [4]

Το επόμενο μεγάλο βήμα στην ιστορία του Cloud Computing έγινε το 1969. Ο Αμερικανός επιστήμονας υπολογιστών J.C.R Licklider συνέβαλε στη δημιουργία του ARPANET (Advanced Research Projects Agency Network) , του προγόνου του

σημερινού Internet. Όραμά του ήταν όλοι να μπορούσαν να συνδεθούν και να χουν πρόσβαση σε πληροφορία (data) και προγράμματα (programs) σε οποιοδήποτε σημείο από οπουδήποτε. Από πολλούς θεωρείται ο "πατέρας" του Cloud Computing. [3]

Figure 5 VM logo [5]

Τη δεκαετία του '70, η IBM κυκλοφόρησε ένα λειτουργικό σύστημα με την ονομασία VM που επέτρεπε διαχειριστές να χουν πολλαπλά εικονικά συστήματα / μηχανήματα ή αλλιώς Virtual Machines (Vms) σε ένα μοναδικό φυσικό κόμβο. Το συγκεκριμένο λειτουργικό σύστημα υιοθέτησε το μηχανισμό του time-sharing της δεκαετίας του '50 και το εξέλιξε σε ένα ολοκαίνουριο επίπεδο, με αποτέλεσμα κάποιες λειτουργίες που χρησιμοποιούνται στα σημερινα συστήματα υπολογιστικού νέφους να οφείλουν την ύπαρξή τους στο VM της IBM.

Τη δεκαετία του '90, οι εταιρείες τηλεπικοινωνιών οι οποίες προηγουμένως προσέφεραν δίκτυα μεταγωγής κυκλώματα (point-to-point), στράφηκαν σε εικονικά ιδιωτικά δίκτυα/virtual private networks (VPNs) , με καλή ποιότητα υπηρεσίας (quality of service - QoS) και ικανοποιητικό κόστος.

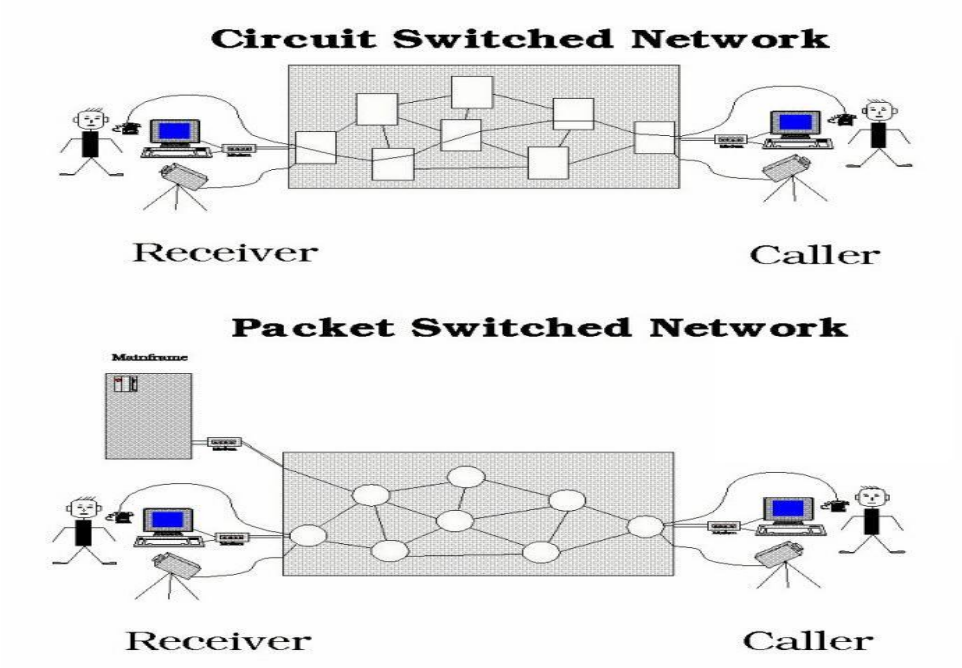


Figure 6 Circuit switched vs Packet switched network [6]

Η μεταγωγή κυκλώματος χρησιμοποιείται ευρέως στα κλασικά τηλεφωνικά δίκτυα και βασίζεται στην εγκατάσταση ενός κυκλώματος για όλη τη διάρκεια της τηλεφωνικής σύνδεσης. Η πληροφορία μεταφέρεται με μια συγκεκριμένη συχνότητα επανάληψης, δηλαδή μόνο ένα κανάλι που χει ορισθεί ως χρονική θυρίδα είναι διαθέσιμο με αποτέλεσμα τη μη αποδοτική χρήση του εύρους ζώνης. [4]

Στα συστήματα μεταγωγής πακέτου (packet switching) η πληροφορία του χρήστη καθώς και η επιπρόσθετη πληροφορία της επικεφαλίδας που χρησιμοποιείται εσωτερικά στο δίκτυο για δρομολόγηση, διόρθωση λαθών, έλεγχο ροής καθώς και άλλες λειτουργίες ενθυλακώνονται σε πακέτα. Σημαντικό πλεονέκτημα θεωρείται η αξιοπιστία του συγκεκριμένου συστήματος μεταγωγής καθώς και η αξιοποίηση του εύρους ζώνης. [4]

Το σύμβολο του συννέφου άρχισε να χρησιμοποιείται ως σημείο οριοθέτησης μεταξύ του παρόχου και του καταναλωτή. Το cloud computing εκτείνει αυτό το όριο περιλαμβάνοντας όλους τους εξυπηρετητές (servers) καθώς και όλη την υποδομή του δικτύου. Το επόμενο βήμα για τους επιστήμονες ήταν η εύρεση τρόπων για τη μετάβαση από τα εικονικά υπολογιστικά συστήματα στην εικονοποιήση (virtualisation) σε επίπεδο δικτύου, με στόχο την αποδοτικότερη αξιοποίηση της υποδομής, της πλατφόρμας και των εφαρμογών μέσω αλγορίθμων και τεχνικών.

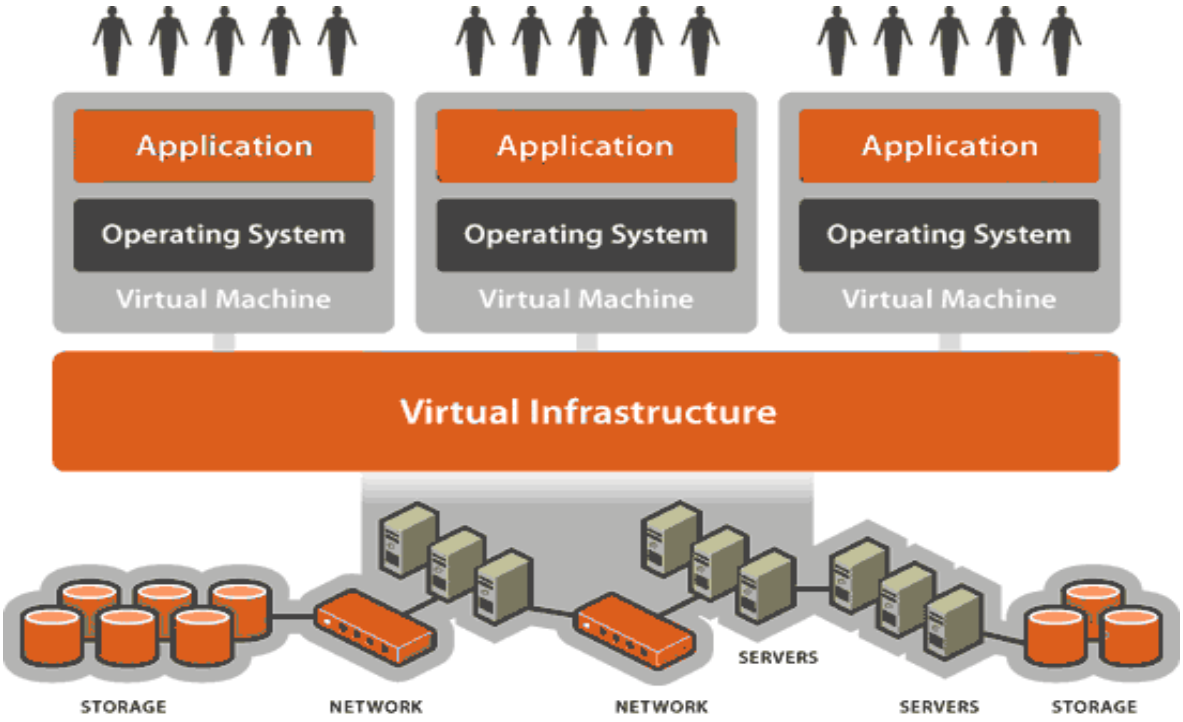
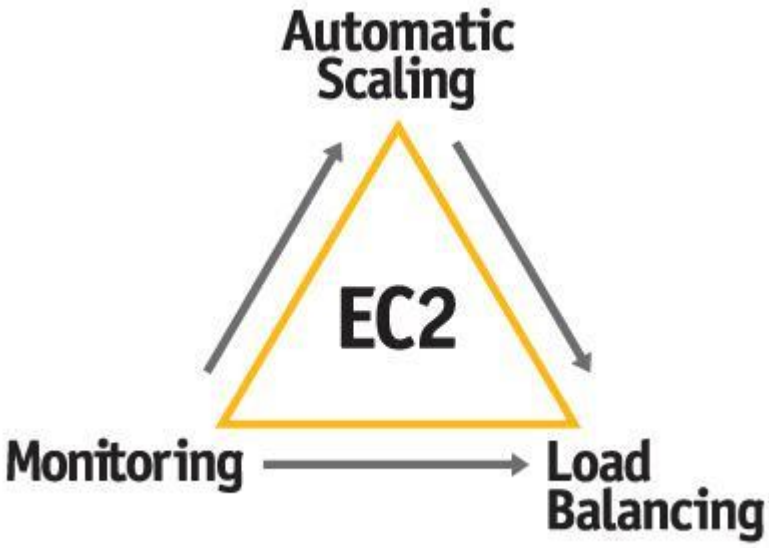


Figure 7 Virtualization of infrastructure [7]

Η προέλευση του όρου Cloud Computing δεν είναι ξεκάθαρη. Η πρώτη χρήση του λογίζεται είτε σε ειδικούς στην Compaq Computer το 1996 είτε στον καθηγητή του Πανεπιστημίου του Τέξας , Ramnath Chellappa , που αναφέρθηκε σε αυτόν τον όρο σε μια ακαδημαική ομιλία του το 1997 σχετικά με ένα νέουπολογιστικόπρότυπο (computing paradigm). [5]



Το 2002, η Amazon δημιουργεί το Amazon Web Services (AWS) , παρέχοντας ένα ανεπτυγμένο σύστημα υπηρεσιών cloud. Το 2006, εισάγει το Elastic Compute Cloud (EC2) ως εμπορική υπηρεσία δικτύου (web service). Ως αποτέλεσμα, επιχειρήσεις είχαν τη δυνατότητα να νοικιάζουν υπολογιστές για να τρέχουν τις δικές τους εφαρμογές. [5]

Figure 8 EC2 χαρακτηριστικά [8]

Από το 2008 και μετά παρατηρείται η εισαγωγή στην αγορά λογισμικού και open-source πλατφόρμων για την ανάπτυξη διαφόρων ειδών συστημάτων Cloud σε ιδιώτες ή οργανισμούς με υψηλό QoS και μια ευρεία γκάμα IT υπηρεσιών. Μερικά παραδείγματα είναι τα Eucalyptus, OpenNebula, OpenStack, Google App Engine, Windows Azure και Oracle Cloud.

2.1.4 Πλεονεκτήματα και μειονεκτήματα Cloud Computing

2.1.4.1 Πλεονεκτήματα

Τα συστήματα Cloud computing εμφανίζουν τα παρακάτω χαρακτηριστικά-κλειδιά που μπορούν να παρουσιαστούν ως πλεονεκτήματα: [2] [6]

- Μειωμένο κόστος

  Αποτελεί το σημαντικότερο θετικό στοιχείο του cloud computing που επιτυγχάνεται με την εξάλειψη επενδύσεων σε αυτοτελή λογισμικό και εξυπηρετητές. Μέσω του cloud, οι επιχειρήσεις εξοικονομούν χρήματα λόγω της έλλειψης συνδρομών για licences και της απουσίας λειτουργικών δαπανών όπως το κόστος για την αποθήκευση data , τη διαχείρηση και την αναβάθμιση λογισμικού. Επιπρόσθετα νέα μοντέλα χρέωσης έχουν εμφανιστεί όπως το one-time-payment και το pay-as-you-go κάνοντας το cloud computing πιο ελκυστικό ως υλοποίηση. Σημαντικό στοιχείο θεωρείται και το μειωμένο κόστος αρχικής επένδυσης (Low Barrier to Entry).

- Διεπαφή εφαρμογής (API - Application programming interface)

  Οι μηχανές έχουν τη δυνατότητα αλληλεπίδρασης με cloud λογισμικό με τον ίδιο τρόπο που μια διεπαφή χρήστη επιτυγχάνει αλληλεπίδραση μεταξύ ανθρώπου και υπολογιστή.
  Τα συστήματα υπολογιστικού νέφους χρησιμοποιούν Representational State Transfer (REST)-based APIs.

- Συντήρηση

  Η συντήρηση (maintenance) εφαρμογών cloud computing είναι εύκολη, επειδή δε χρειάζεται η εγκατάσταση τους στους υπολογιστές κάθε χρήστη χωριστά και η πρόσβαση είναι εφικτή από οποιοδήποτε σημείο.

- Προσβασιμότητα

  Η πρόσβαση για τους χρήστες μπορεί να επιτευχθεί από οποιοδήποτε σημείο και μέσω πληθώρας ηλεκτρονικών συσκευών όπως ηλεκτρονικούς υπολογιστές, smartphones, tablets κτλ. με απαραίτητη προυπόθεση απλά την ύπαρξη υποδομής δικτύου.

- Απόδοση και κλιμάκωση

  Οι cloud διεργασιές αναπτύσσονται αυτόματα μόνο όταν χρειάζονται με άμεση συνέπεια την πληρωμή μόνο για εφαρμογές και αποθηκευτικό χώρο που είναι αναγκαία.
  Τα συστήματα cloud χρησιμοποιούν διανεμημένες αρχιτεκτονικές που προσφέρουν αξιόλογη υπολογιστική ισχύ.

- Αξιοπιστία

  Ένα σύστημα υπολογιστικού νέφους χτίζεται πάνω σε μια σθεναρή αρχιτεκτονική παρέχοντας ελαστικότητα(resiliency) και πλεονασμό(redundancy) στους χρήστες. Παρέχονται συστήματα αυτόματης ανακατεύθυνσης μεταξύ πλατφόρμων hardware παράλληλα με υπηρεσίες ανάκαμψης(recovery). Η δυνατότητα υλοποίησης εξισορρόπησης φόρτου (load-balancing) και εφεδρική λειτουργίας (back-up) καθιστά το Cloud ιδιαίτερα αξιόπιστο.

- Παραγωγικότητα

  Αύξηση της παραγωγικότητας (productivity) μπορεί να επιτευχθεί με την επεξεργασία πληροφορίας από πολλαπλούς χρήστες παράλληλα, όπως και με την απουσία αναγκαιότητας εγκατάστασης αναβαθμίσεων λογισμικού από τους ίδιους τους χρήστες.

- Μεγάλος αποθηκευτικός χώρος

  Το cloud computing εξαλείφει ανησυχίες έλλειψης αποθηκευτικού χώρου σε τοπικά συστήματα υπολογιστών παρέχοντας ουσιαστικά σχεδόν απεριόριστη αποθηκευτική χωρητικότητα, επαρκή για όλες τις ανάγκες.

- Φιλικότητα στο περιβάλλον

  Ένα cloud σύστημα είναι γενικά πιο αποδοτικό από μια τυπική IT υποδομή χρησιμοποιώντας λιγότερους υπολογιστικούς πόρους με συνέπεια την εξοικονόμηση ενέργειας. Ανά πάσα στιγμή, μόνο οι πόροι που πραγματικά είναι αναγκαίοι καταναλώνονται από το σύστημα.

2.1.4.2 Μειονεκτήματα

Αν και τα συστήματα Cloud Computing παρουσιάζουν σωρεία θετικών χαρακτηριστικών, εν τέλει υπάρχουν και κάποια μειονεκτήματα, που αναλόγως μπορούν να αντιμετωπιστούν επιτυχώς: [7]

- Ασφάλεια/Ιδιωτικότητα

  Αξιοποιώντας μια απομακρυσμένη cloud υποδομή, τα δεδομένα μιας επιχείρησης τα οποία μπορεί να ναι ευαίσθητα και εμπιστευτικα εκτείθενται και είναι επιρρεπή σε κακόβουλες επιθέσεις. Αυτό με τη σειρά του θέτει σε κίνδυνο την ύπαρξη της ίδιας επιχείρησης. Είναι καθήκον του παρόχου cloud υπηρεσιών η διαχείριση και προστασία των δεδομένων , άρα η αξιοπιστία του παρόχου είναι εξέχουσας σημασίας.

- Συμβατότητα

  Όταν μια επιχείρηση αποφασίσει να ενταχθεί στο Cloud , η πιθανότητα να αναγκαστεί να αντικαταστήσει το μεγαλύτερο μέρος των υποδομών της λόγω ασυμβατότητας είναι αρκετά υψηλή. Μια λύση στην παραπάνω κοστοβόρα κίνηση είναι τα υβριδικά νέφη.

- Εξάρτηση προμηθευτή (vendor lock-in)

  Ένα βασικό μειονέκτημα στα συστήματα υπολογιστικού νέφους αποτελεί η εξάρτηση από τον πάροχο. Αν ένας χρήστης αποφασίσει να αλλάξει πάροχο, τότε η μεταφορά μεγάλης ποσότητας δεδομένων από τον ένα πάροχο στον άλλο μπορεί να προβεί ιδιαίτερα δύσκολη και κοπιαστική.

- Τεχνικές δυσκολίες

  Η πιθανότητα μιας σοβαρής δυσλειτουργίας ακόμα και στα cloud συστήματα των καλύτερων παρόχων δε μπορεί να αγνοηθεί. Η τεχνολογία cloud μπορεί να προσβληθεί από διακοπές λειτουργίας, προβλήματα ηλεκτροδότησης αλλά και από προβλήματα συνδεσιμότητας του δικτύου.

2.1.5 Υλοποιήσεις Cloud Computing

Ένα μοντέλο διάθεσης καθορίζει το σκοπό του νέφους και τη φύση της τοποθεσίας αυτού. Οι ορισμοί του NIST για τα τέσσερα μοντέλα διάθεσης είναι οι ακόλουθοι: [1] [2]

- Δημόσιο νέφος (Public cloud):

  Η υποδομή δημοσίου νέφους είναι διαθέσιμη για δημόσια χρήση, και ανήκει σε έναν οργανισμό που πωλεί υπηρεσίες νέφους. Ο πάροχός επιτρέπει στους καταναλωτές να έχουν υπό τον έλεγχο τους πόρους που έχουν ζητήσει, και αυτό γίνεται συνήθως μέσω μιας διαδικτυακής διεπαφής. Το ενοίκιο του καταναλωτή είναι σύμφωνα με την ανάγκη του μέσω του μοντέλου "pay-as-you-go" (πληρώνεις όσο χρησιμοποιείς). Τα public Clouds προσφέρουν πρόσβαση σε μεγάλα "ταμεία" υπολογιστικών πόρων οι οποίοι μάλιστα είναι αρκετά επεκτάσιμοι και σε προσωρινό ακόμα επίπεδο χωρίς να απαιτούν κάποια επένδυση κεφαλαίου για την ανάπτυξη των υποδομών του data center. Τα πιο διάσημα δημόσια νέφη περιλαμβάνουν τα Amazon Web Services, Google AppEngine, and Microsoft Azure.

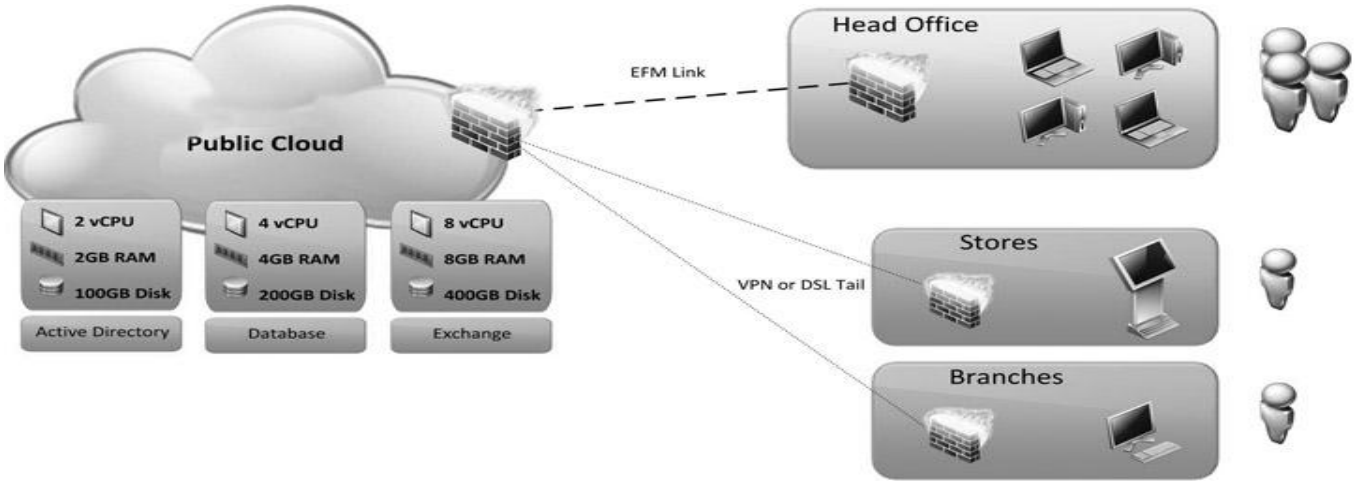


Figure 9 Public cloud outline example [9]

- Ιδιωτικό νέφος (Private cloud):

Η υποδομή ιδιωτικού νέφους αποτελεί ένα σύνολο απο υπολογιστικούς πόρους που προσφέρονται ως ένα προτυποποιημένο σύνολο υπηρεσιών οι οποίες καθορίζονται, σχεδιάζονται και χρησιμοποιούνται αποκλειστικά από έναν οργανισμό. Η επιλογή ανάπτυξης ενός private cloud συνήθως καθοδηγείται από την ανάγκη για διατήρηση του πλήρους ελέγχου ενός παραγωγικού περιβάλλοντος εξαιτίας ιδιαίτερων απαιτήσεων των εφαρμογών σε θέματα είτε απόδοσης είτε νομικού πλαισίου λειτουργίας. Σημαντικό χαρακτηριστικό αλλά συνάμα μειονέκτημα του αποτελεί το υψηλό κόστος απόκτησης και λειτουργίας του. Επειδή έχει εφαρμοστεί στα πλαίσιο ενός ήδη υπάρχοντος data center κάποιου οργανισμού (και προφανώς πίσω από ένα ήδη υπάρχων firewall) το private Cloud υπόκειται στους περιορισμούς ασφάλειας του οργανισμού και επομένως παρέχει μεγαλύτερη ασφάλεια σε ευαίσθητα δεδομένα.

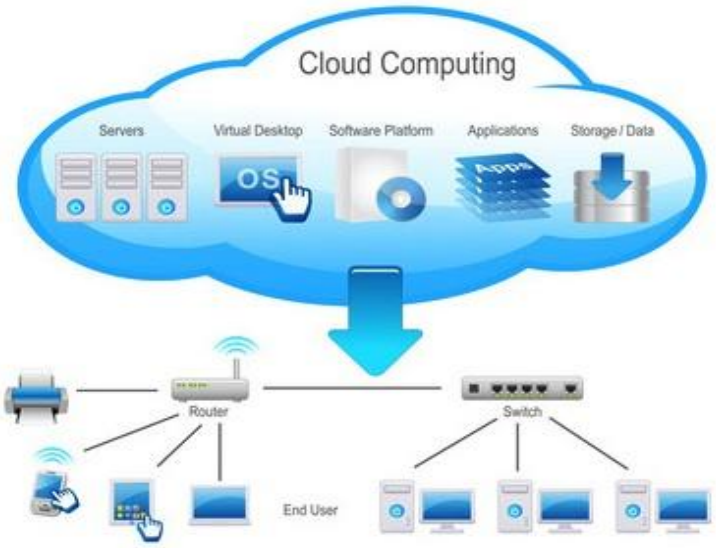


Figure 10 Private cloud outline example [10]

Επιπλέον, τα private Clouds σταθεροποιούν και βελτιστοποιούν την απόδοση ενός ήδη υπάρχοντος hardware σε ένα συγκεκριμένο data center μέσω της τεχνολογίας του Virtualization που χρησιμοποιούν, βελτιώνοντας κατά πολύ έτσι την αποτελεσματικότητα του data center ενώ παράλληλα μειώνουν και τα λειτουργικά κόστη. Στα ιδιωτικά νέφη χρησιμοποιείται από τους οργανισμούς λογισμικό όπως για παράδειγμα VMWare, OpenNebula, CloudStack, vCloud Director και OpenStack.

- Νέφος Κοινότητας (Community cloud):

  Σε αυτό το μοντέλο η δομή του νέφους έχει οργανωθεί για να εξυπηρετεί μια κοινή λειτουργία ή ένα κοινό σκοπό. Μπορεί να χρησιμοποιείται από έναν οργανισμό ή από πολλούς οργανισμούς που έχουν σαν κοινό τόπο κάποιο ενδιαφέρον ή στόχο (π.χ. αποστολή, απαιτήσεις ασφάλειας, πολιτική, σκέψεις υποχωρητικότητας – συμβιβασμού, ασφάλεια κτλ.) . Το νέφος κοινότητας μπορεί να το διαχειρίζονται οι οργανισμοί που το χρησιμοποιούν ή ένας τρίτος.

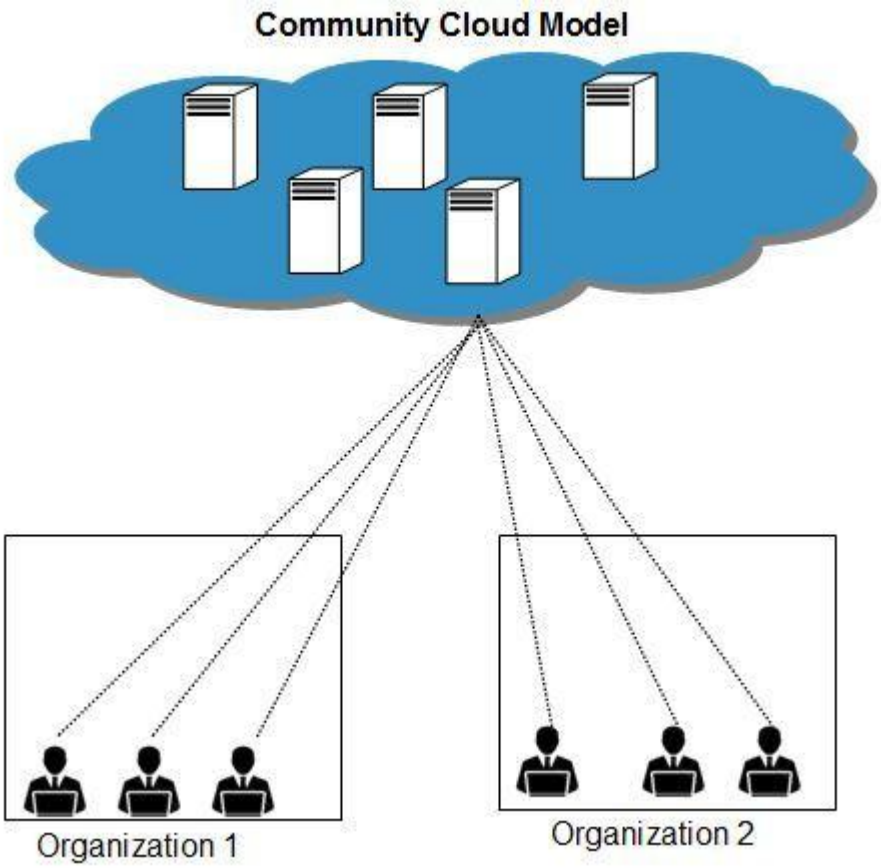


Figure 11 Community cloud model outline [11]

- Υβριδικό νέφος (Hybrid cloud):

  Η δομή ενός υβριδικού νέφους συνδυάζει πολλά μοντέλα νεφών (ιδιωτικό, κοινοτικό ή δημόσιο). Τα νέφη αυτά διατηρούν τα ιδιαίτερα χαρακτηριστικά τους, αλλά είναι συνδεδεμένα μαζί σαν μια μονάδα.

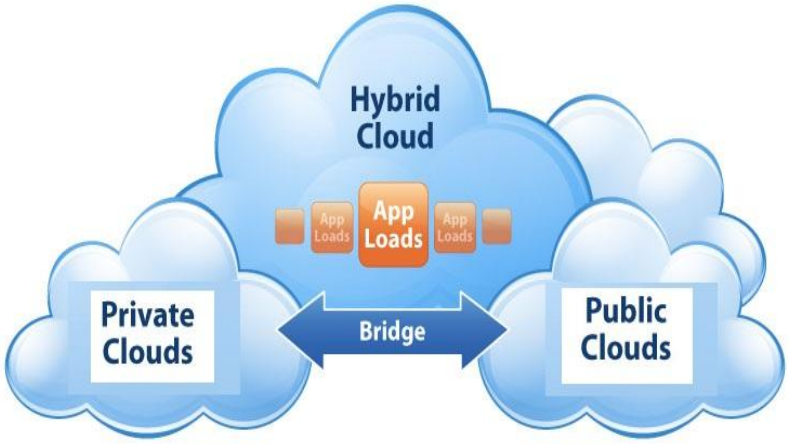


Figure 12 Hybrid cloud model outline [12]

2.1.6 Μοντέλα υπηρεσιών Cloud Computing

Τα μοντέλα υπηρεσιών Cloud περιγράφουν τον τρόπο με τον οποίο οι υπηρεσίες cloud γίνονται διαθέσιμες στους πελάτες. Τα πιο ουσιώδη μοντέλα υπηρεσιών περιλαμβάνουν ένα συνδυασμό των IaaS (infrastructure as a service), PaaS (platform as a service) και SaaS (software as a service). Επίσης υπάρχη αλληλοεξάρτηση μεταξύ αυτών των μοντέλων. [2]

- IaaS (Infrastructure as a Service)

  Το μοντέλο Υποδομή-ως-Υπηρεσία παρέχει στοιχεία υποδομών στους πελάτες όπως για παράδειγμα εικονικές μηχανές (virtual machines), εικονικό αποθηκευτικό χώρο, εικονικά δίκτυα (VLANs), προγράμματα προστασίας (firewall), εξισορροπητές φορτίου (load balancers) καθώς και άλλους υπολογιστικούς πόρους που μπορούν να αξιοποιήσουν οι χρήστες ανα πάσα στιγμή. Με το IaaS, ο φορέας παροχής των υπηρεσιών διαχειρίζεται όλη την υποδομή μέσω hypervisors οι οποίοι τρέχουν τα virtual machines , ενώ ο χρήστης είναι αρμόδιος για το χαμηλότερο επίπεδο λογισμικού, δηλαδή το λειτουργικό σύστημα και τις εφαρμογές.

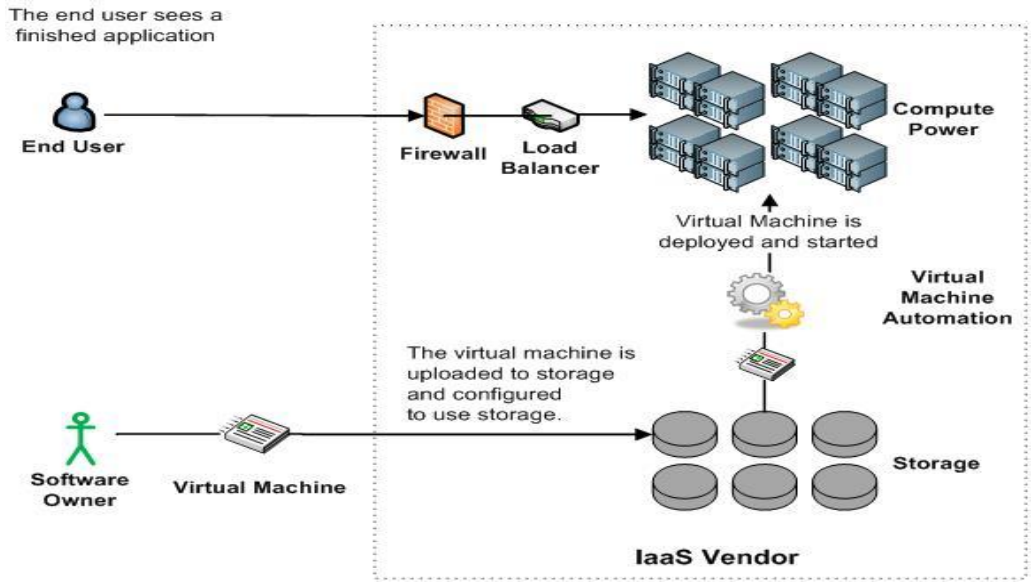


Figure 13 IaaS model example [13]

Οι Amazon Web Services, OpenStack, IBM και VMware ειναι κάποιοι από τους μεγαλύτερους παρόχους IaaS υπηρεσιών.

- PaaS (Platform as a Service)

Το μοντέλο Πλατφόρμα-ως-Υπηρεσία παρέχει μια ήδη υπάρχουσα υπολογιστική πλατφόρμα εφαρμογών στον πελάτη, συμπεριλαμβανομένων λειτουργικό σύστημα, περιβάλλον εκτέλεσης προγραμματιστικής γλώσσας, βάση δεδομένων ή ένα web server. Οι χρήστες δε χρειάζεται να δαπανήσουν χρόνο για τη δημιουργία της υποδομής που χρειάζεται για τις εφαρμογές τους. Τα στοιχεία που προσφέρει το PaaS διαμορφώνονται και παρέχονται ανάλογα τις απαιτήσεις των εφαρμογών και διαχειρίζονται συνήθως μέσω ενός API που προσφέρει ένα σύνολο λειτουργιών γι αυτο το σκοπό. Το Google AppEngine είναι ένας διάσημος πάροχος PaaS υπηρεσιών. Επίσης οι Amazon Web Services όπως και το Windows Azure παρέχει και PaaS υπηρεσίες εκτός απο IaaS υπηρεσίες.

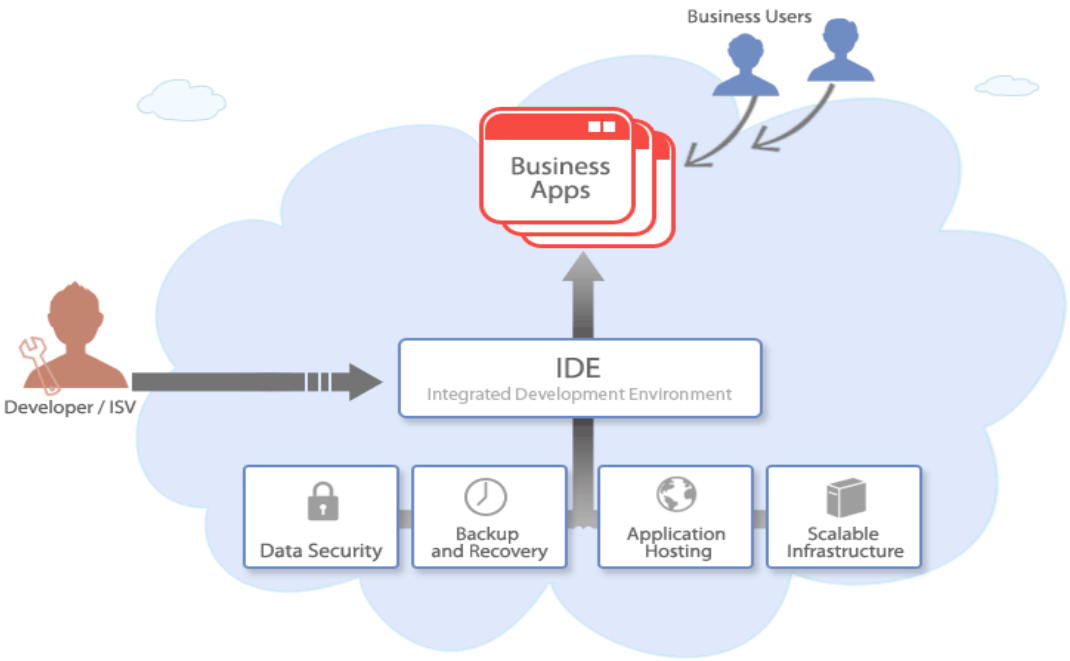


Figure 14 PaaS model example [14]

- SaaS (Software as a Service)

Στο μοντέλο Λογισμικό-ως-Υπηρεσία οι χρήστες έχουν πρόσβαση σε έτοιμες λύσεις/εφαρμογές λογισμικού και βάσεις δεδομένων. Ο πάροχος υπηρεσιών SaaS έχει τον πλήρη έλεγχο αυτών των εφαρμογών, αναλαμβάνοντας τόσο την εγκατάσταση όσο και τη διαχείριση τους, απαλλάσσοντας τους χρήστες από τέτοιες ενέργειες και βοηθώντας τις επιχειρήσεις να μειώσουν κατά πολύ τα έξοδα σε υπολογιστικούς πόρους. Παραδείγματα SaaS υπηρεσιών αποτελούν το e-mail, οι πλατφόρμες κοινωνικής δικτύωσης, τα συστήματα διαχείρισης project κτλ. Το μοντέλο SaaS συχνά αναφέρεται σαν "Software on Demand" και κοστολογείται συνήθως ανα χρήση

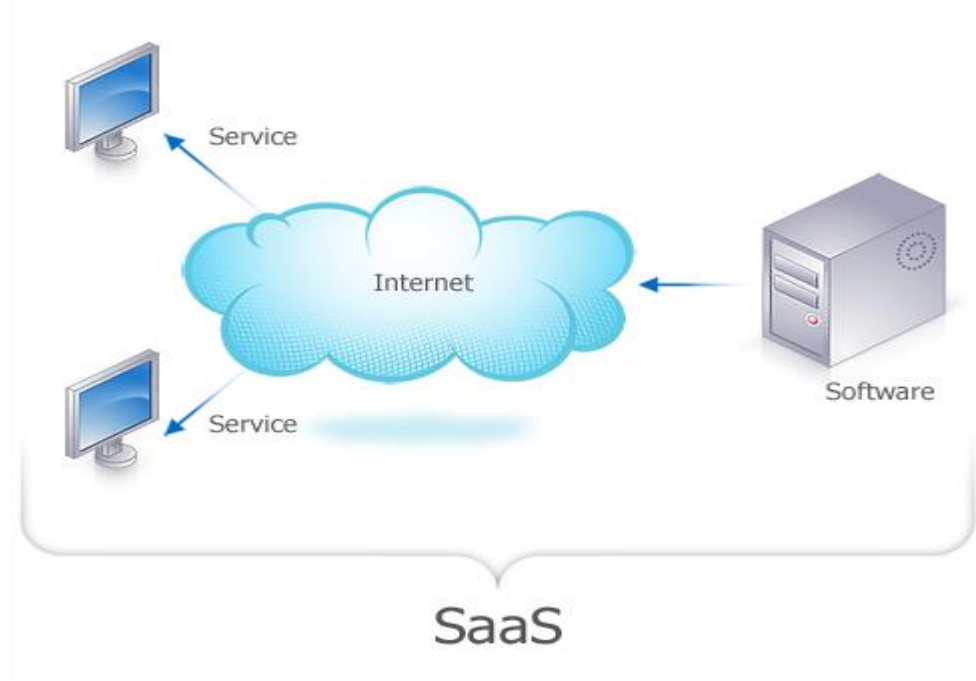


Figure 15 SaaS model outline [15]

Τα μοντέλα υπηρεσιών υπολογιστικού νέφους αναπαριστώνται συνήθως σε όρους στοίβας υλικού/λογισμικού. Μια τέτοια αναπαράσταση αποκαλείται μοντέλο αναφοράς νέφους (Cloud Reference Model) . Στο κατώτατο σημείο της στοίβας είναι το υλικό ή η υποδομή που περιλαμβάνει το δίκτυο. Καθώς κινούμαστε προς τα πάνω στη στοίβα, κάθε μοντέλο υπηρεσιών κληρονομεί τις ιδιότητες του μοντέλου υπηρεσιών που βρίσκεται από κάτω του.

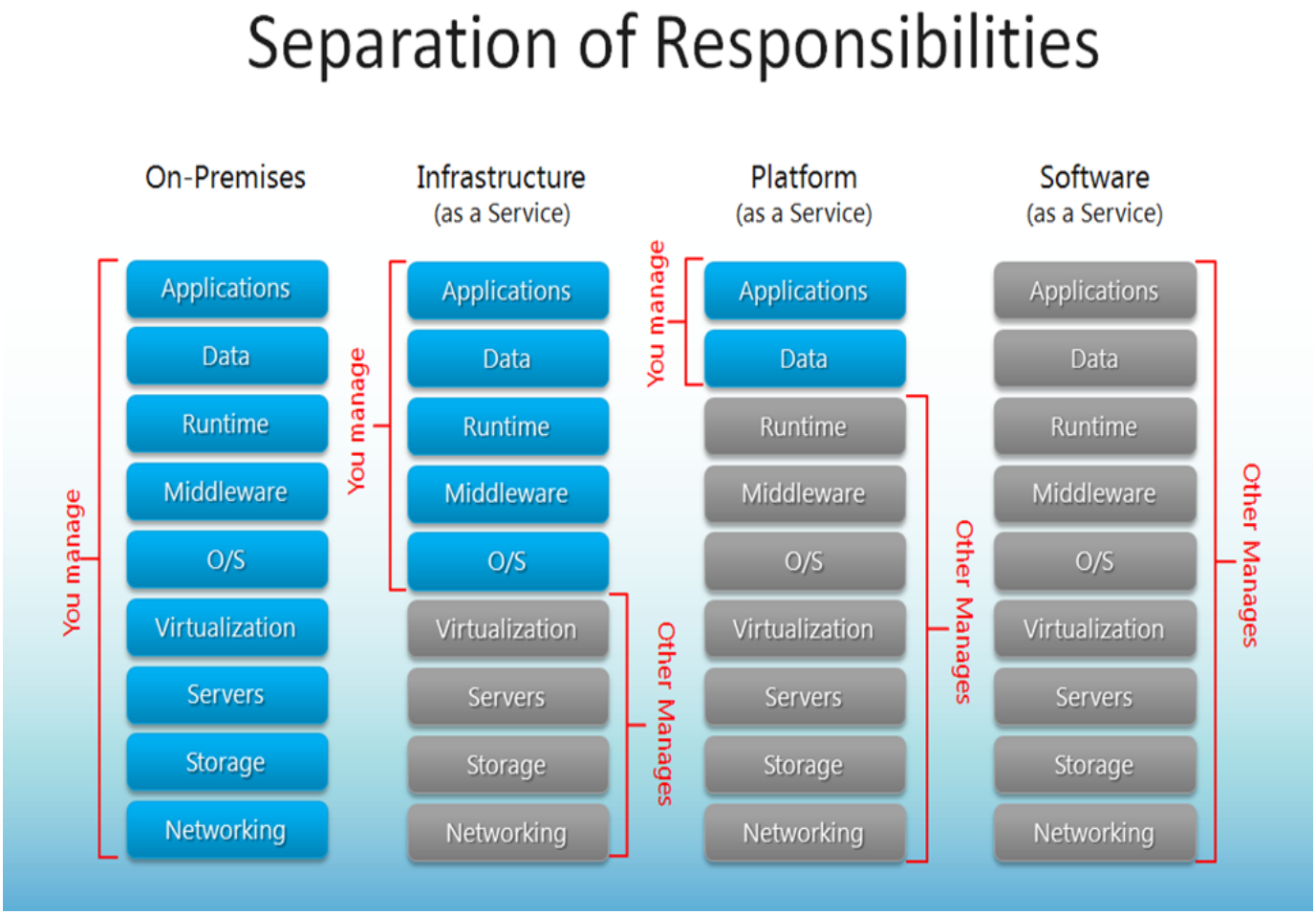


Figure 16 The 3 main cloud models responsibilities [16]

Τον τελευταίο καιρό έχουν κάνει την εμφάνιση τους και τα δύο παρακάτω μοντέλα:

- MaaS (Metal as a Service)

  Το Metal as a Service (MaaS) αποτελεί μια καινούρια δομή παροχής cloud υπηρεσιών που αναπτύχθηκε από τους δημιουργούς του λειτουργικού συστήματος Ubuntu και φέρνει τη γλώσσα του cloud στους φυσικούς servers. Έχει σχεδιαστεί να διευκολύνει και να αυτοματοποιεί την ανάπτυξη και δυναμική παροχή μεγάλων υπολογιστικών περιβαλλόντων όπως servers καθορίζοντας το hardware αναλόγως την υπηρεσία. Υπηρετεί ως ένα στρώμα (layer) κάτω από το IaaS. Με μια απλή δικτυακή διεπαφή επιτρέπεται η πρόσθεση, παροχή, ενημέρωση ή ανακύκλωση των servers κατ' απαίτηση. [8] [9]

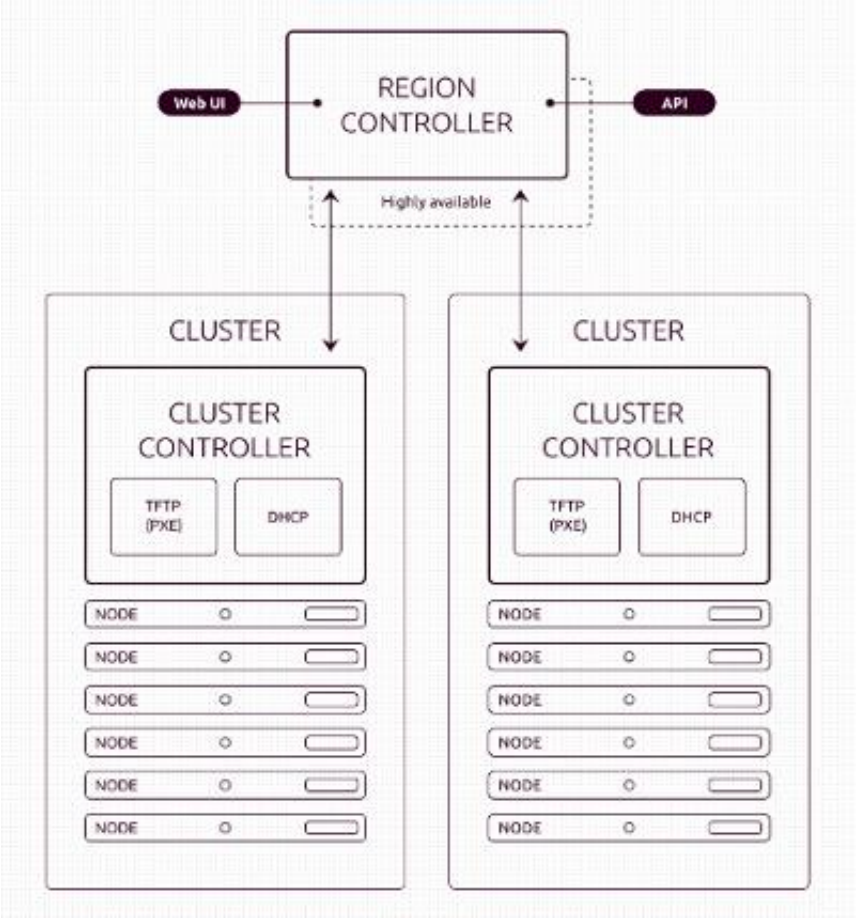


Figure 17 MaaS model [17]

- NaaS (Network as a Service)

To Network as a Service (NaaS) είναι ένα καινούριο μοντέλο στο χώρο του cloud computing που αφορά τη διανομή υπηρεσιών δικτύου εικονικά πάνω από το διαδίκτυο από ιδιοκτήτες της υποδομής σε τρίτους (συνήθως με πληρωμή ανα χρήση ή συνδρομή). Κάποιες απο τις υπηρεσίες που παρέχονται: [10]

1. Εικονικό ιδιωτικό δίκτυο (VPN) :

   Επεκτείνει ένα ιδιωτικό δίκτυο και τους πόρους του δίνοντας του τη δυνατότητα αποστολής/λήψης δεδομένων διαμέσου διαμοιρασμένων ή ιδιωτικών δικτύων.

2. Εύρος ζώνης κατ απαίτηση :

   Η χωρητικότητα του δικτύου ορίζεται δυναμικά με βάση τις απαιτήσεις των κόμβων/χρηστών.

3. Εικονοποίηση mobile δικτύου :

   Ένας κατασκευαστής δικτύων τηλεπικοινωνιών χειρίζεται ένα δίκτυο και παρέχει τις δυνατότητες του σε τρίτους με χρέωση ανά χρήση.

Χαρακτηριστικά: [11]

- Ορατότητα δικτύου :

  Ακριβής γνώση της τοπολογίας του δικτύου με παροχή αυτής της γνώσης στους χρήστες για βέλτιστη αξιοποίηση του δικτύου.

- Ενδο-δικτυακή ανάλυση και επεξεργασία :

  Έλεγχος της προώθησης πακέτων σε κόμβους με σκοπό τη δημιουργία ειδικών πρωτοκόλλων δρομολόγησης και ειδικών firewalls αναλόγως τον χρήστη.

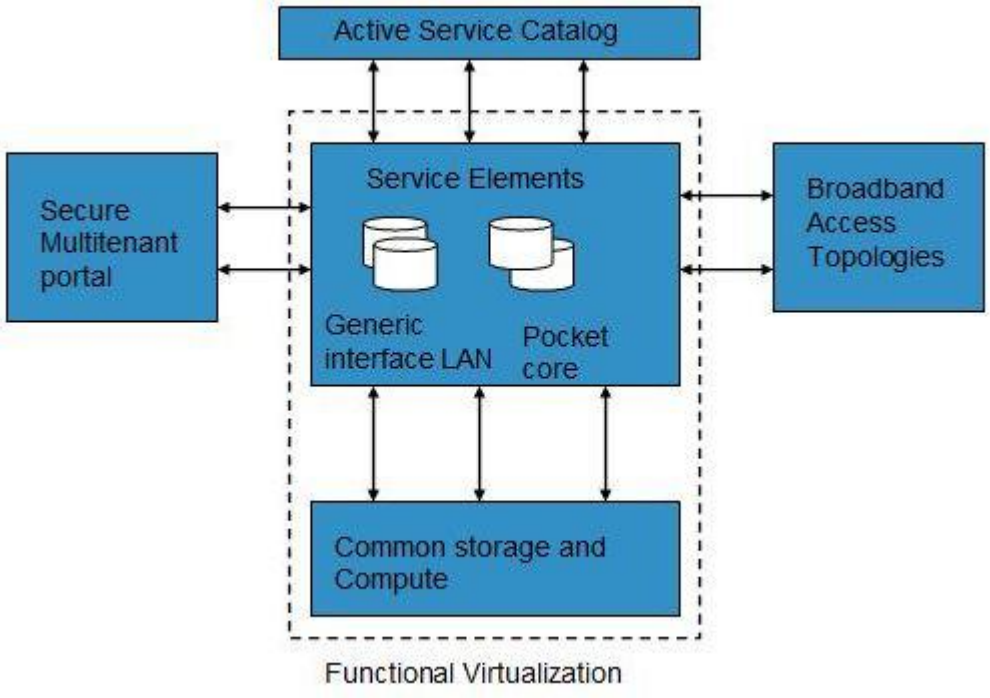


Figure 18 NaaS model [18]

2.1.7 Παραδείγματα Cloud Computing

Σήμερα ο αριθμός των εφαρμογών και των προγραμμάτων που χρησιμοποιούν την τεχνολογία του cloud computing αυξάνεται ραγδαία.

Διαδικτυακές υπηρεσίες ηλεκτρονικού ταχυδρομείου όπως το Gmail (SaaS) και το Outlook (SaaS) είναι από τις πρώτες υπηρεσίες που ανέβηκαν στο «σύννεφο» παρέχοντας μεγαλύτερη γκάμα λειτουργιών και επιλογών επεξεργασίας στους χρήστες από ένα απλό μέσο ηλεκτρονικής αλληλογραφίας. Η ηλεκτρονική αλληλογραφία κάθε χρήστη είναι αποθηκευμένη στους servers της εκάστοτε υπηρεσίας, χωρίς να χρειάζεται να είναι αποθηκευμένη τοπικά στον υπολογιστή του χρήστη και άμεσα προσβάσιμη από οποιοδήποτε σημείο.

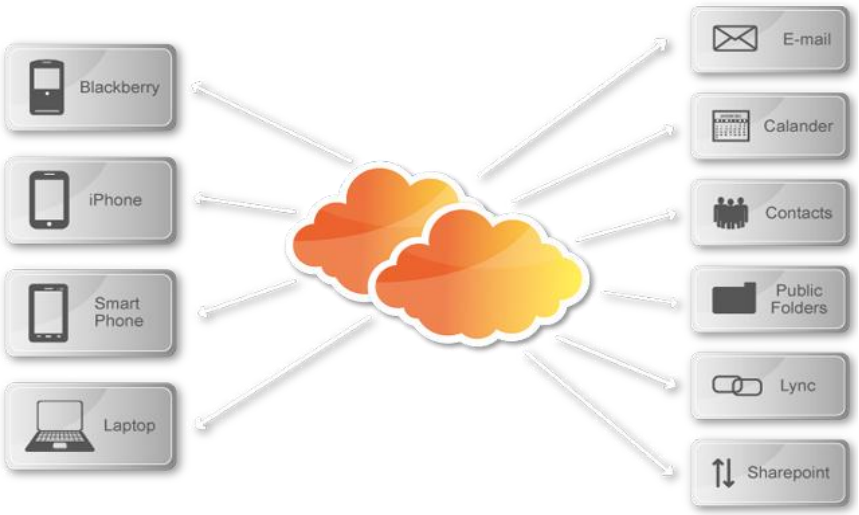


Figure 19 E-mail and cloud [19]

Υπηρεσίες αποθήκευσης στο cloud εμφανίζονται συνεχώς και με ολοένα περισσότερο χώρο για το χρήστη. Το Dropbox, το Windows SkyDrive, το GoogleDrive και το Ubuntu One (SaaS) αποτελούν υπηρεσίες αποθήκευσης, συγχρονισμού, φύλαξης και sharing αρχείων μεγάλου όγκου. Επιτρέπουν την άμεση πρόσβαση στα αρχεία αυτά από οποιοδήποτε υπολογιστή, σε πολλαπλούς χρήστες αν αυτό έχει επιτραπεί ενώ μερικές φορές το κατέβασμα των αρχείων δεν είναι απαραίτητο και ο χρήστης μπορεί να δει φωτογραφίες, βίντεο κτλ. απευθείας μέσω ενος browser.

Figure 20 Cloud and some of its services

Η κοινωνικη δικτύωση, ένα απο τα μεγαλύτερα φαινόμενα σε παγκόσμιο επίπεδο, υποστηρίζεται σε τεράστιο βαθμό από το cloud μέσω δημοφιλών κοινωνικών εφαρμογών (SaaS) όπως το Facebook, το Linkedin, το Twitter, εφαρμογών VoIP όπως το Skype και το Google Voice καθώς και media υπηρεσιών όπως το Youtube.

Figure 21 Cloud and some applications

Το cloud έχει μπει δυνατά και στο χώρο των επιχειρήσεων. Το salesforce όπως και τα SugarCRM και ZohoCRM (SaaS) είναι Customer Relationship Management συστήματα που λειτουργούν αποκλειστικά στο cloud και χρησιμοποιούνται από μεγάλες εταιρείες παγκοσμίως.

Figure 22 Cloud and CRM [20]

2.2 Πλατφόρμες monitoring σε συστήματα Cloud Computing

Οργανισμοί συνεχώς εκφράζουν ενδιαφέρον στην αξιοποίηση cloud computing υπηρεσιών με σκοπό την βελτιστοποίηση της ευελιξίας και της επεκτασιμότητας των IT υπηρεσιών που παρέχονται στους τελικούς χρήστες. Από την άλλη μεριά όμως, η παρακάτω πρόκληση γίνεται ιδιαίτερα εμφανής: η μειωμένη ορατότητα στην απόδοση των υπηρεσιών που παρέχονται στους χρήστες. Αυτό σε πρακτικό επίπεδο ερμηνεύεται ως πιθανή άγνοια από την πλευρά του οργανισμού για μειωμένη απόδοση, για προβλήματα κατά την παροχή ή λειτουργία της υπηρεσίας και για άλλους παράγοντες που αποτελούν τροχοπέδη για την παροχή βέλτιστου QoS (quality of service) στον χρήστη ο οποίος τελικά είναι ο μόνος που έχει επίγνωση των παραπάνω προβλημάτων. Αποτέλεσμα ήταν η ανάγκη για ανάπτυξη συστημάτων monitoring μέσω των οποίων οι οργανισμοί θα είχαν τη δυνατότητα ανά πάσα στιγμή να γνωρίζουν για την κατάσταση των προσφερόμενων υπηρεσιών και να μπορούν να επέμβουν όποτε χρειαστεί για την αντιμετώπιση έκτακτων καταστάσεων.

Η σημασία του monitoring σε μια cloud υποδομή μπορεί εύκολα να διαπιστωθεί από παρακάτω ωφέλη που προσφέρει: [12]

- ✓ Ορατότητα στο cloud

  Ένα IaaS cloud περιβάλλον παρέχει μεγάλη ευελιξία στη δημιουργία νέων server instances ανά πάσα στιγμή, χωρίς ανησυχία για τη διαχείριση της πραγματικής υποδομής. Εν τούτοις, ακόμα και σε ένα διαχειρίσιμο cloud, συνίστανται προληπτικές ενέργειες και έλεγχοι των instances. Ένα cloud instance που βρίσκεται σε ένα απομακρυσμένο third-party κέντρο δεδομένων χρειάζεται προσεκτική παρακολούθηση όπως ακριβώς θα συνέβαινε αν ένας server ήταν δίπλα μας οπότε θα χαμε αυτόματα συνείδηση συγκρουόμενων διεργασιών που τρέχουν ταυτόχρονα, διαρροών μνήμης όπως επίσης και της ανάγκης προσθήκης επιπλέον χωρητικότητας σε περίπτωση περιόδων υψηλής δραστηριότητας.

- ✓ Απόδοση εφαρμογών & server

  Η απόδοση επηρεάζει άμεσα τους χρήστες εφαρμογών που τρέχουν σε κάποιο cloud instance. Είναι απαραίτητη η γνώση εαν μια εφαρμογή υπερκαταναλώνει ή υποκαταναλώνει υπολογιστικούς πόρους. Συνεπώς ένας υπεύθυνος παρακολούθησης μιας cloud υποδομής πρέπει να παρακολουθήσει παραμέτρους όπως για παράδειγμα τον αποθηκευτικό χώρο, τις μονάδες εισόδου-εξόδου (I/O), τη μνήμη, τη CPU και το δίκτυο. Επίσης υποχρεούται να λάβει υπόψη του διάφορες καταστάσεις όπως αν η ταχύτητα του συστήματος έχει ελαττωθεί , αν οι υπολογιστικοί πόροι υπερκαταναλώνονται καθώς και να θέτει ένα χρονικό όριο ώστε να ενημερώνεται έγκαιρα για τυχόν προβλήματα πριν υπάρξει αντίκτυπος στον πελάτη.

- ✓ Χωρητικότητα και κατανάλωση

  Μια άλλη θεώρηση-κλειδί είναι η τυχούσα επίδραση μακροχρόνιας χρήσης υπολογιστικών πόρων στην χωρητικότητα. Υπάρχουν δύο σενάρια. Στο πρώτο ένα φαινόμενο μακράς διαρκείας μπορεί να οφείλεται σε λανθασμένο συγχρονισμό συστήματος ή λάθη στον κώδικα όπως διαρροή μνήμης (memory leak). Ένα τέτοιο φαινόμενο μπορεί να διαγνωστεί από συμπτώματα αυξημένης κατανάλωσης πόρων που κάνουν το σύστημα πιο αργό. Το κλειδί είναι η έγκαιρη αντιμετώπιση πριν τα συμπτώματα γίνουν εμφανή. Στο δεύτερο σενάριο ο συνεχώς αυξανόμενος αριθμός χρηστών των εφαρμογών όπως και η αύξηση των αρχείων και του κώδικα στο σύστημα χρειάζεται να υποστηρίζεται από έγκαιρη παράλληλη αύξηση της χωρητικότητας.

Μια από τις πιο δημοφιλείς πλατφόρμες monitoring παρουσιάζεται στη συνέχεια.

2.2.1 *Nagios*

Το Nagios είναι μια ανοιχτού-κώδικα εφαρμογή λογισμικού με σκοπό την παρακολούθηση (monitoring) συστημάτων, δικτύων και υποδομών, προσφέροντας μια ευρεία γκάμα υπηρεσιών monitoring για servers, εφαρμογές και υπηρεσίες. Ενημερώνει τους χρήστες σε περίπτωση προβλήματος καθώς και όταν έχει επιλυθεί το πρόβλημα. Επίσης το Nagios παρέχει ολοκληρωμένη παρακολούθηση υπηρεσιών web, cloud computing και storage. Μέσω της χρήσης του επιτυγχάνεται αποτελεσματικό monitoring σε ποικιλία φυσικών αλλά και εικονικών μηχανημάτων με αποτέλεσμα αυξημένη διαθεσιμότητα αλλά και γρήγορη ανίχνευση διακοπών λειτουργίας δικτύου και προβλημάτων στο περιβάλλον του νέφους. [13]

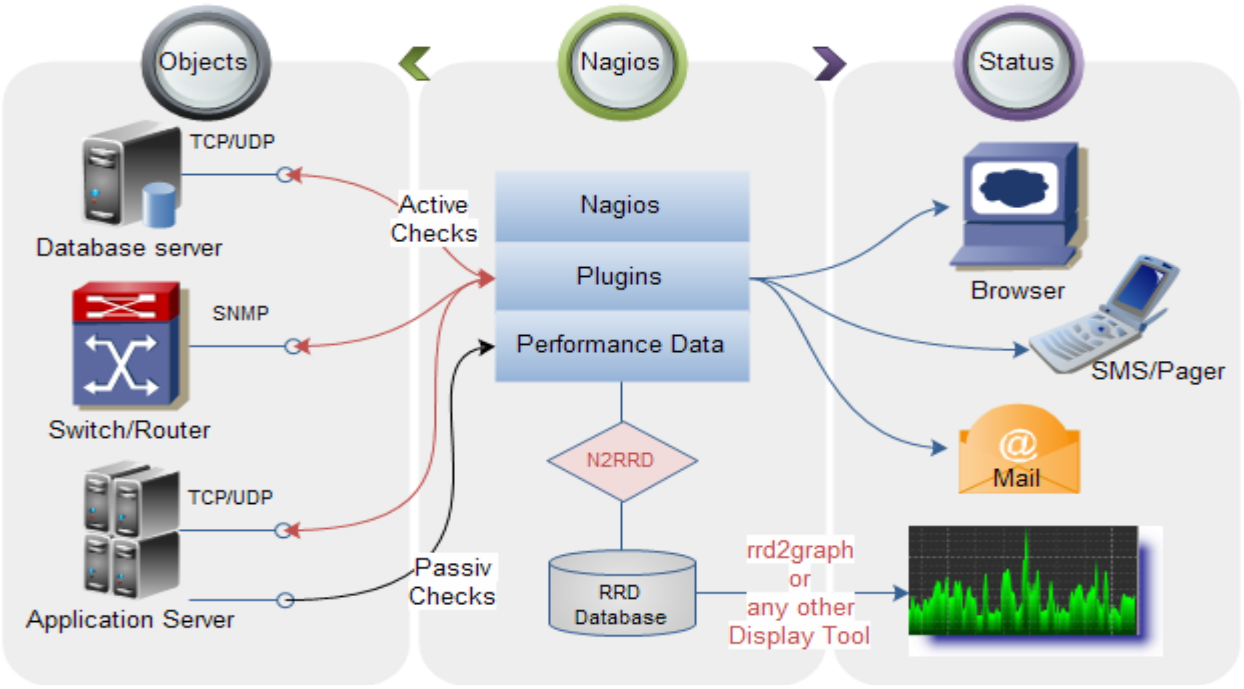


Figure 23 Operating principle of Nagios [21]

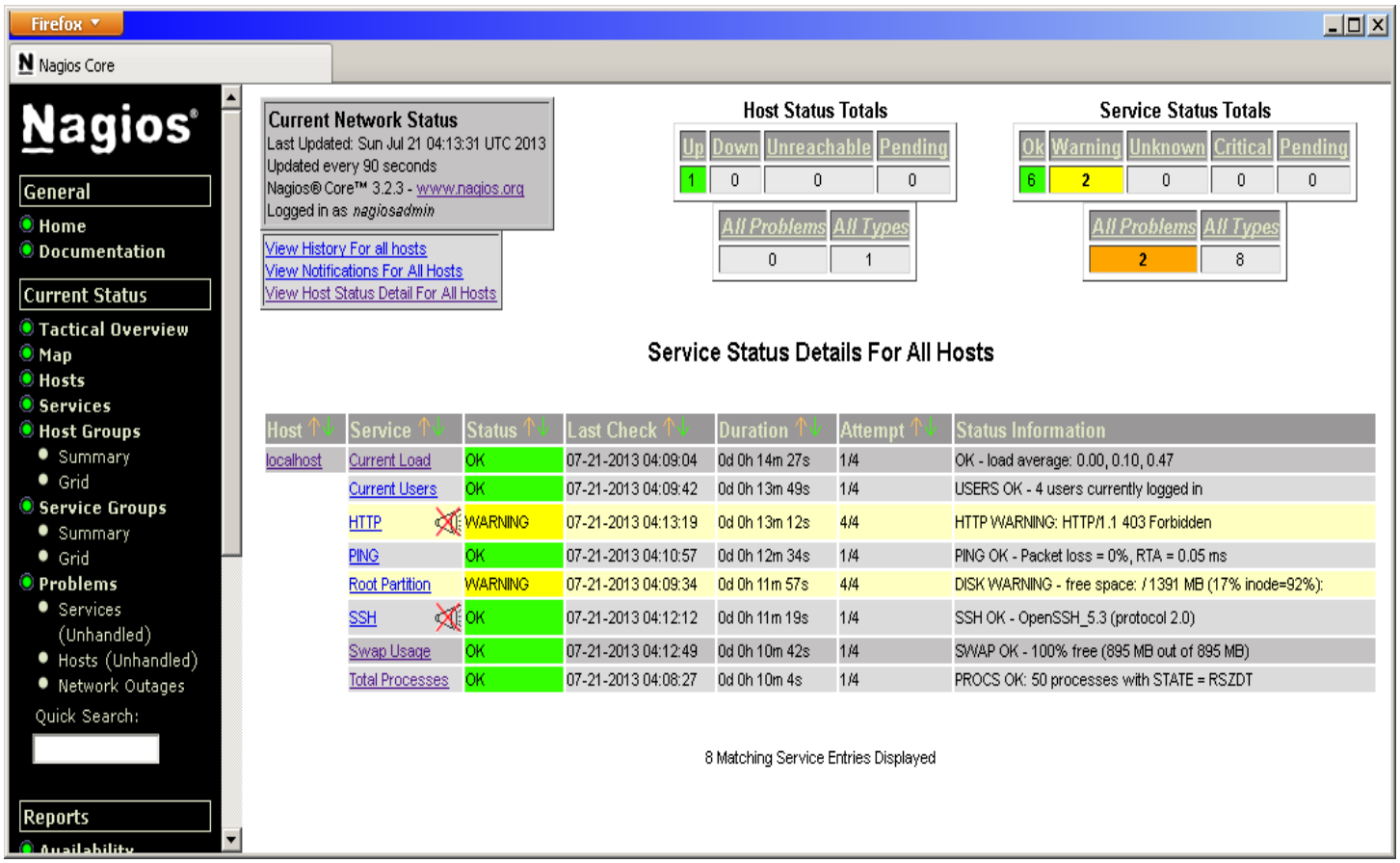


Figure 24 Παράδειγμα interface του Nagios

Ανάπτυξη υπηρεσιοστρεφούς πλατφμόρμας παρακολούθησης και ελέγχου υποδομής υπολογιστικού νέφους

3.1 Αρχιτεκτονική

3.1.1 Μοντέλο Πλατφόρμας

Το μοντέλο που υλοποιήθηκε χαρακτηρίζεται ως ένα client-server μοντέλο. Πρόκειται για μια δομή που διαμοιράζει εργασίες μεταξύ πόρων ή υπηρεσιών που αποτελούν την πλευρά του server και των αιτώντων υπηρεσιών που αποτελούν την πλευρά του client. [14]

Το client-server μοντέλο περιγράφει τη σχέση των συνεργαζόμενων προγραμμάτων σε μια εφαρμογή. Ο server παρέχει μια λειτουργία ή υπηρεσία σε ένα ή περισσότερους clients από τους οποίους προέρχονται αιτήσεις για τέτοιου είδους υπηρεσίες.

Γενικότερα, μια υπηρεσία (service) αποτελεί μια αφηρημένη έννοια υπολογιστικών πόρων και δεν είναι αναγκαία η ενασχόληση του client με τον τρόπο που ο server χειρίζεται κάθε αίτηση (request). Ο client μπορεί απλά να χει την ικανότητα να καταννοεί τις απαντήσεις (responses) που στέλνονται πίσω από τον server βασιζόμενος στο πρωτόκολλο εφαρμογής, το περιεχόμενο και τη μορφοποίηση της πληροφορίας χωρίς αυτό να είναι αναγκαστικά απαραίτητο.

Η επικοινωνία μεταξύ clients και servers επιτυγχάνεται με την ανταλλαγή μηνυμάτων μέσω ενός μοτίβου μηνυμάτων αίτησης-απάντησης (request-response). Για την επιτυχή επικοινωνία απαιτούνται μια κοινή «γλώσσα» και συγκεκριμένοι κανόνες που ορίζονται σε κάποιο πρωτόκολλο επικοινωνίας. Όλα τα client-server πρωτόκολλα λειτουργούν στο στο στρώμα εφαρμογής του OSI (application layer). Για τον περαιτέρω φορμαλισμό της ανταλλαγής πληροφορίας, μπορεί να εφαρμοστεί από τον server κάποιο API (διεπαφή προγραμματισμού εφαρμογών). Το API αποτελεί ένα στρώμα αφαίρεσης για πόρους

όπως βάσεις δεδομένων ή λογισμικό. Περιορίζοντας την επικοινωνία σε συγκεκριμένη μορφή περιεχομένου, επιτυγχάνεται με περισσότερη ευκολία το parsing, δηλαδή η ανάλυση strings από σύμβολα είτε σε φυσική είτε σε προγραμματιστική γλώσσα.

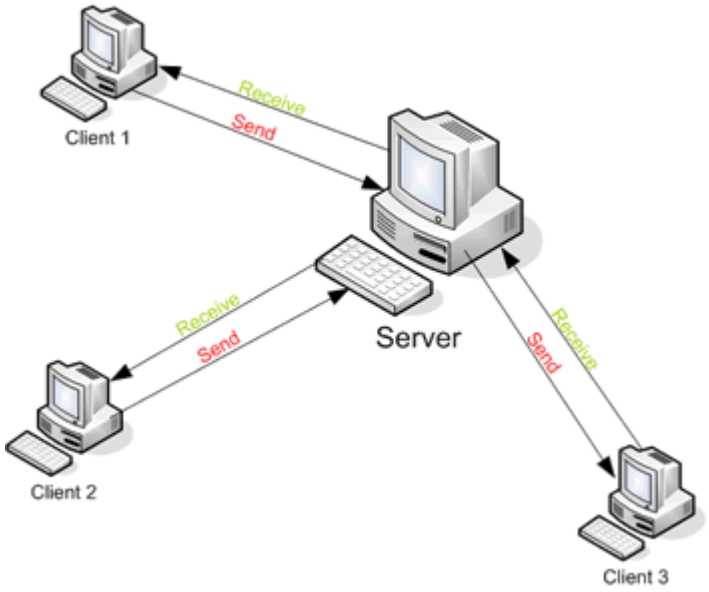


Figure 25 Simplified client-server model [22]

3.1.2 *Συστατικά στοιχεία πλατφόρμας monitoring cloud υποδομής*

Η αρχιτεκτονική του συστήματος μπορεί να αναλυθεί σε 2 επιμέρους τμήματα: το middleware και την πλατφόρμα.

3.1.2.1 Middleware

Ως middleware χρησιμοποιείται μια cloud computing πλατφόρμα ως IaaS λύση, το OpenStack. Γενικότερα ο ρόλος ενός middleware είναι να συνδέει υπολογιστές και μηχανήματα με άλλες εφαρμογές, να υποστηρίξει την επικοινωνία μεταξύ πολλαπλών στοιχείων λογισμικού καθώς και λειτουργίες εισόδου-εξόδου. Αποτελεί το στρώμα (software layer) μεταξύ του λειτουργικού συστήματος και των εφαρμογών. Συνήθως αναφέρεται και ως συνδετικός κρίκος σε ένα μοντέλο client-server. [15]

Αναλυτικότερα ακολουθεί μια περιγραφή της αρχιτεκτονικής του OpenStack. [16]

Το OpenStack είναι μια ανοιχτού κώδικα cloud computing πλατφόρμα λογισμικού. Η συγκεκριμένη τεχνολογία ελέγχει μεγάλα pools υπολογιστικών, αποθηκευτικών και δικτυακών πόρων διαμέσου ενός data center και μπορεί να διαχειριστεί από τους χρήστες μέσω ενός δικτυακά βασισμένου πίνακα ελέγχου (dashboard) όπως και με τη χρήση command-line εργαλείων ή ενός RESTful API.

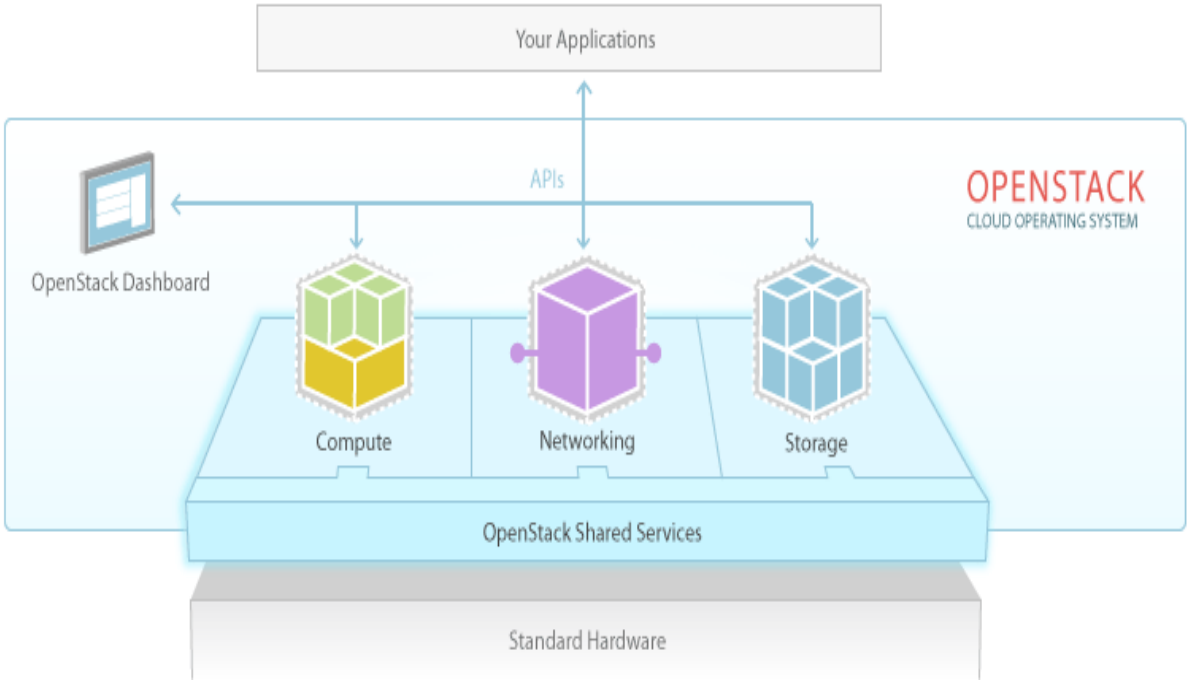


Figure 26 Openstack layers [23]

Το OpenStack project ως σύνολο σχεδιάστηκε με σκοπό τη διανομή ενός επεκτάσιμου σε μεγάλο βαθμό cloud λειτουργικού συστήματος. Για το λόγο αυτό κάθε μία από τις υπηρεσίες-συστατικά στοιχεία έχουν σχεδιαστεί να συνεργάζονται για την παροχή ενός πλήρους IaaS. Αυτό επιτυγχάνεται μέσω μέσω APIs που κάθε υπηρεσία προσφέρει και είναι διαθέσιμα στους χρήστες.

Παρακάτω αναφέρονται και περιγράφονται οι υπηρεσίες που συνθέτουν την OpenStack αρχιτεκτονική μαζί το χαρακτηριστικό όνομα του project στο οποίο ανήκουν:

- Dashboard (Horizon)

  Παρέχει μια web βασισμένη αυτο-εξυπηρετούμενη πύλη για αλληλεπίδραση με τις OpenStack υπηρεσίες, όπως η έναρξη ενός instance, η ανάθεση μιας IP address και ο συντονισμός ελέγχων πρόσβασης.

- Compute (Nova)

  Διαχειρίζεται τον κύκλο ζωής των υπολογιστικών instances σε ένα περιβάλλον OpenStack. Κάποιες από τις ευθύνες αυτής της υπηρεσίας είναι να «γεννά», να χρονοδρομολογεί και να θέτει εκτός λειτουργίας virtual machines.

- Networking (Neutron)

  Επιτρέπει δικτυακή συνδεσιμότητα μεταξύ των υπόλοιπων OpenStack υπηρεσιών, παρέχοντας ένα API για τους χρήστες για να ορίσουν δίκτυα και οτιδήποτε περιλαμβάνουν. Υποστηρίζει όλες τις νέες τεχνολογίες και παρόχους δικτύων.

- Object Storage (Swift)

  Αποθηκεύει και ανακτά τυχαία αδόμητα data objects μέσω ενός RESTful, HTTP βασισμένου API. Υποστηρίζει αναπαραγωγή data με αποτέλεσμα να είναι ανθεκτικό σε λάθη.

- Block Storage (Cinder)

  Παρέχει συνεχές block αποθηκευτικό χώρο στα instances που τρέχουν και παράλληλα η αρχιτεκτονική του επιτρέπει τη δημιουργία και διαχείριση των συσκευών αποθήκευσης.

- Identity Service (Keystone)

  Παρέχει πιστοποίηση(authentication) και εξουσιοδότηση(authorization) για τις υπόλοιπες OpenStack υπηρεσίες.

- Image Service (Glance)

  Αποθηκεύει και ανακτά disk images εικονικών μηχανημάτων και χρησιμοποείται σε εκτενή βαθμό από την Compute υπηρεσία.

- Telemetry (Ceilometer)

  Εκτελεί λειτουργίες monitoring σε OpenStack cloud για σκοπούς αξιολόγησης, τιμολόγησης αλλά και για στατιστικούς λόγους.

- Orchestration (Heat)

  «Ενορχηστρώνει» πολλαπλές σύνθετες cloud εφαρμογές χρησιμοποιώντας ειδικού τύπου template format μέσω ενός Openstack-τύπου REST API και ενός συμβατικού με cloud Query API.

- Database Service (Trove)

  Παρέχει επεκτάσιμη και αξιόπιστη Cloud Database-as-a-Service λειτουργία για συσχετιστικές και μη database engines.

Στο παρακάτω σχήμα απεικονίζεται παραστατικά η αρχιτεκτονική του OpenStack με όλες τις υπηρεσίες του και πώς διασυνδέονται και αλληλεπιδρούν μεταξύ τους

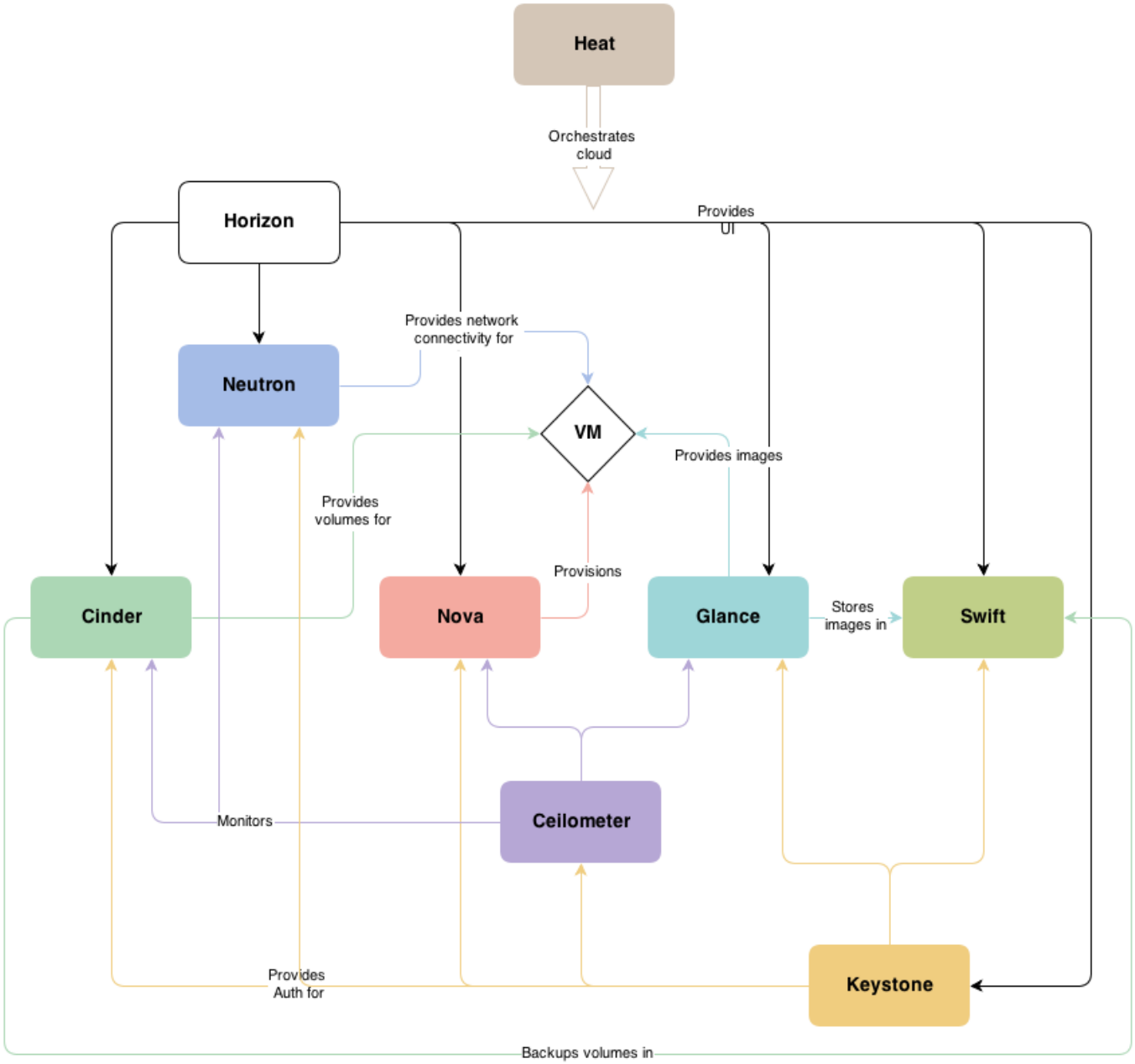


Figure 27 Openstack conceptual architecture [24]

3.1.2.2 Cloud Infrastructure Platform

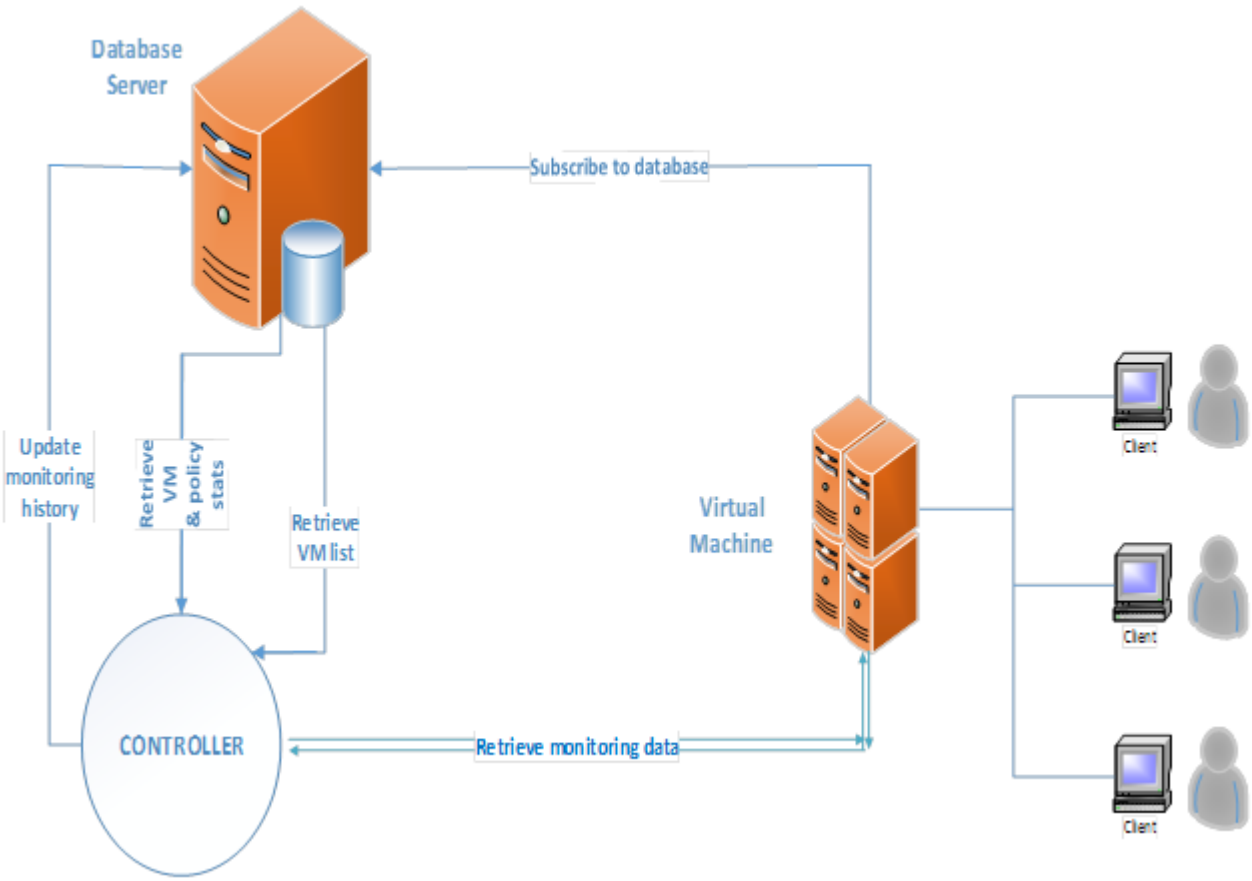


Figure 28 Σχεδιάγραμμα πλατφόρμας υλοποίησης

Η παραπάνω πλατφόρμα που υλοποιήθηκε βρίσκεται ένα layer πάνω από το middleware και αποτελείται από τα παρακάτω συστατικά στοιχεία:

Virtual Machine

Το Virtual Machine (VM) αποτελεί μια προσομοίωση ενός συγκεκριμένου υπολογιστικού περιβάλλοντος. Τα VMs λειτουργούν βασισμένα στην αρχιτεκτονική και τις λειτουργίες ενός πραγματικού ή υποθετικού υπολογιστή και η εφαρμογή τους μπορεί να περιλαμβάνει ειδικό εξοπλισμό και λογισμικό. Αιτήσεις για επεξεργαστική ισχύ, μνήμη, σκληρό δίσκο και άλλα στοιχεία εξοπλισμού διαχειρίζονται από το στρώμα εικονοποίησης (virtualization layer). Τα VMs δημιουργούνται και τρέχουν μέσα σε ένα τέτοιο layer, όπως μια virtualization πλατφόρμα ή κάποιο σύστημα ελέγχου και παρακολούθησης VMs (hypervisors) το οποίο τρέχει πάνω από κάποιον client ή λειτουργικό σύστημα. Το virtualization layer μπορεί να χρησιμοποιηθεί για τη δημιουργία πολλών μεμονωμένων Virtual Machine περιβαλλόντων.

Τυπικά τα φιλοξενούμενα λειτουργικά συστήματα και προγράμματα δεν έχουν γνώση ότι τρέχουν σε μια εικονική πλατφόρμα. Αυτό έχει ως αποτέλεσμα η εγκατάστασή τους να επιτυγχάνεται με τον ίδιο τρόπο όπως σε ένα φυσικό server. Ένα επιπλέον πλεονέκτημα αποτελεί το γεγονός ότι εφαρμογές και υπηρεσίες που τρέχουν σε ένα VM δεν παρεμβάλουν ούτε το host λειτουργικό σύστημα ούτε άλλα VM.

Controller

Ο controller παρέχει το κεντρικό διαχειριστικό σύστημα για τις υπηρεσίες monitoring. Μερικές από αυτές οι οποίες θα περιγραφούν με περισσότερη λεπτομέρεια σε επόμενο υποκεφάλαιο είναι:

- Ενημερώνεται για τα VMs που είναι σε κατάσταση λειτουργίας καθώς και δέχεται πληροφορίες σχετικές με το hardware τους.
- Παρακολουθεί ανά τακτά χρονικά διαστήματα τα VMs και ενημερώνει τις βάσεις δεδομένων με στατιστικά δεδομένα που πάρθηκαν συγκεκριμένες χρονικές στιγμές.

- Επιτελεί λειτουργίες ελέγχου και κρίσης σχετικά με το πότε κάποιο μηχάνημα βρίσκεται σε κατάσταση critical οπότε χρειάζεται να σηκωθεί και άλλο VM.

Database

Ο ρόλος της βάσης δεδομένων είναι η αποθήκευση πληροφορίας που συλλέγει ο server/controller σχετικά με τα VMs. Για τη δημιουργία της πλατφόρμας, υλοποιήθηκαν 3 επιμέρους βάσεις δεδομένων με ξεχωριστό ρόλο η καθεμία.

Πρώτη και κυριότερη η Vm Database η οποία αποθηκεύει τις εξης πληροφορίες

1. Όνομα του VM
2. IP διεύθυνση
3. Σκληρός δίσκος
4. Μνήμη RAM
5. CPU πυρήνες
6. Κατάσταση VM

Επίσης υλοποιήθηκε μια Monitoring Database η οποία αποθηκεύει δεδομένα που κρατά ο server χάρη στο monitoring ανά συγκεκριμένο χρονικό διάστημα όπως για παράδειγμα η CPU, η μνήμη ram και ο σκληρός δίσκος που είναι σε χρήση τη συγκεκριμένη στιγμή. Τέλος θεωρήθηκε αναγκαία και η δημιουργία μιας Policy Database. Ρόλος της είναι κρατά τα όρια/thresholds της CPU και της RAM για κάθε VM τα οποία αν ξεπεραστούν τότε ο controller «ρίχνει» σε κατάσταση “down” το συγκεκριμένο VM.

3.1.3 Μηχανισμός παρακολούθησης

Η λειτουργία του monitoring στη συγκεκριμένη πλατφόρμα αναλαμβάνεται εξ ολοκλήρου από την πλευρά του server και πιο ειδικά από τον controller. Είναι μια υπηρεσία που προσφέρεται στον client ο οποίος στην πραγματικότητα είναι τα VMs που βρίσκονται σε κατάσταση "UP".

Η λογική είναι ότι monitoring του VM πραγματοποείται ανά συγκεκριμένο χρονικό διάστημα, τα δεδομένα συλλέγονται από τον controller και τοποθετούνται στις αντίστοιχες βάσεις δεδομένων και το τελικό βήμα είναι η σύγκρισή τους με βάση τα thresholds που έχουν οριστεί ώστε να αποφανθεί αν η κατάσταση του VM είναι εκτός κινδύνου, μπορεί να συνεχιστεί η λειτουργία του και δε χρειάζεται επιπρόσθετο VM να «σηκωθεί» ώστε να υποστηρίξει τις νέες ανάγκες του client.

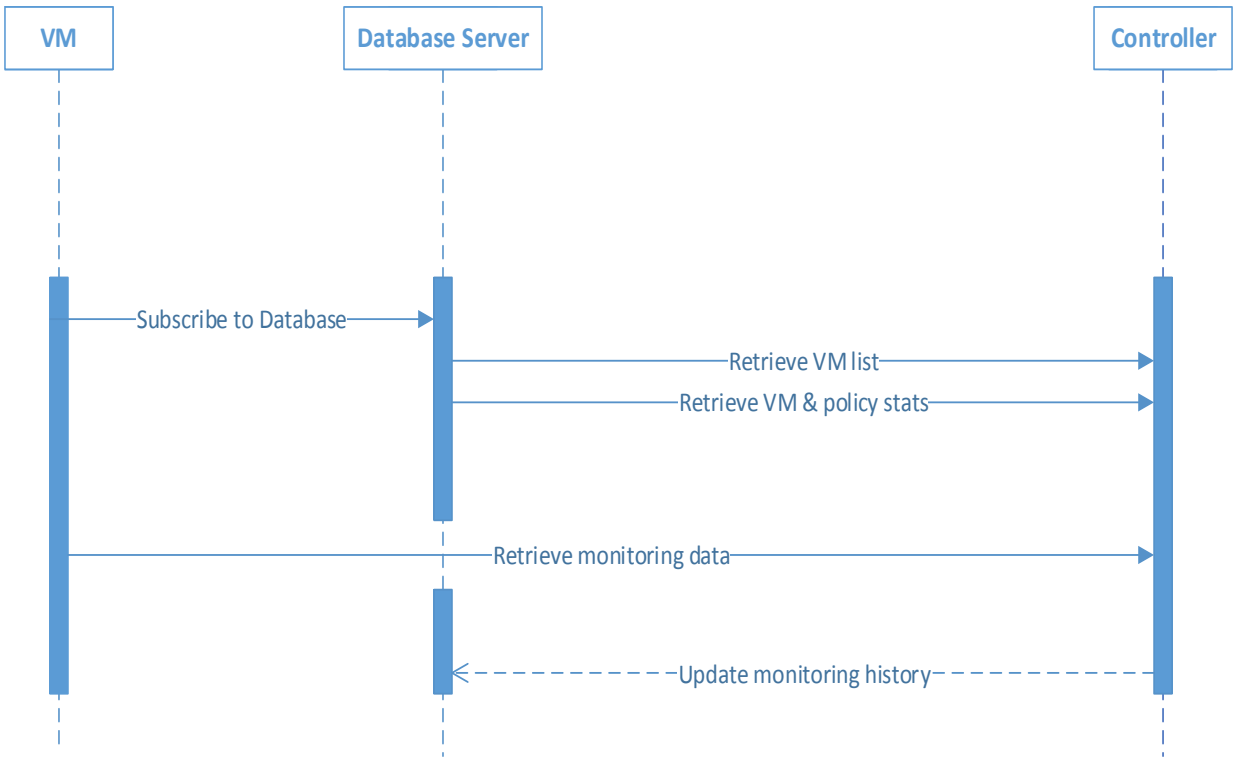


Figure 29 Sequence diagram of platform actions

3.2 Υλοποίηση

3.2.1 Java Project

Το project που υλοποιήθηκε στο Eclipse χωρίζεται σε δυο επιμέρους μέρη. Το πρώτο μέρος είναι το VMInstance που «απεικονίζει» ένα virtual machine instance (ή και περισσότερα) και το δεύτερο ο Controller που περιέχει όλες τις λειτουργίες του Controller.

3.2.1.1 VMInstance

Το project VMInstance περιέχει 3 packages. Στο package model περιέχονται τα αρχεία Vm.java, Monitoring.java και Policy.java. Στα συγκεκριμένα java αρχεία φτιάχνουμε κλάσεις-αντικείμενα τύπου Vm, Monitoring και Policy αντίστοιχα το καθένα με τα δικές του μεταβλητές-σταθερές και δικές του μεθόδους-συναρτήσεις.

Ξεκινώντας απο το Vm.java χρησιμοποιώντας το annotation @XmlRootElement πετυχαίνουμε mapping της κλάσης σε ένα XML στοιχείο. Με το annotation @Entity ορίζουμε ότι η κλάση αποτελεί μια οντότητα (entity) οπότε αυτόματα τοποθετείται στο persistence.xml. Τέλος μέσω της χρήσης των @Table και @NamedQuery ορίζεται ο κυρίως table της οντότητας και αντίστοιχα ένα στατικό query που εφαρμόζεται στην οντότητα.

Κάνοντας implemented το interface Serializable, δηλώνουμε οτι το object είναι σειριοποιήσιμο. Το table Vm με autogenerated κλειδί id περιέχει τα πεδία

- availableDisk
- availableRam
- cpuCores
- ip
- state
- VMname

Τα παραπάνω αποτελούν χαρακτηριστικά γνωρίσματα του συγκεκριμένου virtual machine.
Επίσης ορίζονται μέθοδοι και συναρτήσεις για την ανάκτηση ή το «σετάρισμα» των παραπάνω πεδίων.

Αντίστοιχα το Monitoring.java δομείται κατά ακριβώς τον ίδιο τρόπο απλά με τα δικά του πεδία και τις δικές του συναρτήσεις-μεθόδους

- cpuvalue
- diskvalue
- ramvalue
- timestamp

Επιπρόσθετα ορίζεται ένα Javabean τύπου Vm με το οποίο είναι δυνατή η αντιστοίχιση πολλών γραμμών-«μετρήσεω» του table Monitoring σε μια μοναδική γραμμή του πίνακα Vm , με λίγα λόγια ένα Vm μπορεί να έχει πολλαπλές γραμμές monitoring στον πίνακα.

Ίδια λογική ακολουθείται και για το Policy.java με πεδία

- CPUthres
- RAMthres

Στο DataAccess.java δημιουργείται ένας EntityManager ο οποίος είναι ενα interface με σκοπό την διαχείριση ενός συνόλου από entity instances (με μοναδικό identity) που ορίζεται από ένα persistence unit (@PersistenceContext). Το τελευταίο αναλαμβάνει το mapping με μια μοναδική database.
Επίσης μέσω του @Stateless δηλώνουμε ότι το session bean είναι stateless με ανεξάρτητες λειτουργίες. Μέσα στην κλάση περιλαμβάνονται μέθοδοι και συναρτήσεις για τη δημιουργία, διαγραφή, ενημέρωση ή αναζήτηση entity instances. Χρησιμοποιείται το annotation @Transactional ώστε να επιτρέπονται από την εφαρμογή transactions στα διαχειριζόμενα beans.

Το Controller.java περιέχει όλες τις λειτουργίες που επιτελεί ο controller στα VM instances καθώς και το monitoring. Περαιτέρω επεξήγηση ακολουθεί σε επόμενο υποκεφάλαιο.

Στο package servlets περιέχονται τα ServiceRegistry.java και ServletInit.java. Εδώ περιλαμβάνεται ότι έχει να κάνει με το VM instance αυτό καθεαυτό.

Στο ServletInit.java ορίζουμε ένα servlet με τις αντίστοιχες μεθόδους init() και destroy(). Θεωρώντας ότι προσομοιώνουμε έναν VM server, με την πρώτη μέθοδο δημιουργούμε ένα καινούριο object, βρίσκουμε την IP διεύθυνση καθώς και άλλα στοιχεία όπως CPU, RAM, disk τα οποία καταγράφονται και αποθηκεύονται στην database. Με τη δεύτερη μέθοδο βρίσκουμε τον εκάστοτε VM server με βάση την IP του και θέτουμε την κατάστασή του σε “DOWN”.

Το ServiceRegistry.java περιέχει το interface για registry του service provider instance. Ο Provider παρέχεται από την κλαση InstanceService.

Στο package service περιλαμβάνεται το αρχείο InstanceService.java. Περιλαμβάνει όλες τις υπηρεσίες GET ή POST που μπορούν να κληθούν μαζί με το αντίστοιχο path τους. Με το @Singleton annotation δηλώνουμε ότι για το EJB (Enterprise Javabean) υπάρχει ένα μόνο instance ανά application ανεξάρτητα του αριθμού των clients που θέλουν να χουν πρόσβαση.

Χρησιμοποιώντας σε κάθε μέθοδο το annotation @Produces(MediaType.APPLICATION_JSON),το JBoss αναλαμβάνει τη μετατροπή του Java object μέσω Jackson σε JSON format αυτόματα.

Το τελευταίο αρχείο άξιο αναφοράς είναι το persistence.xml. Περιέχεται στο META-INF directory. Ορίζει ένα persistence unit με μοναδικό όνομα. Μέσω του transaction-type=”JTA” και του jta-data-source ορίζεται η database στην οποία «δείχνει» το persistent unit.

3.2.1.2 Controller

Μέσα στο project του Controller έχουμε δημιουργήσει ένα package model όπως το αντίστοιχο model στο project VMInstance. Το persistence.xml εξακολουθεί να είναι το ίδιο.
Η διαφορά βρίσκεται στο ControllerServlet.java. Ουσιαστικά πρόκειται για ένα servlet που τρέχει τις λειτουργίες του Controller σαν ένα java app σε έναν server. Αναλυτικότερη περιγραφή ακολουθεί στο υποκεφάλαιο του Monitoring.

3.2.2 Βάση Δεδομένων

Στο MySQL Workbench υλοποιήθηκε ένα MySQL instance και ένα schema με όνομα vm_db το οποίο περιλαμβάνει τους πίνακες vm, monitoring και policies της database. Κάθε table περιλαμβάνει όλα τα πεδία που αναφέρονται και στα project. Το κάθε πεδίο έχει το δικό του type ενώ το id είναι primary key, μοναδικό και auto-αυξανόμενο. Σε κάθε γραμμή του πίνακα vm με μοναδικό id μπορεί να αντιστοιχούν περισσότερες απο μία γραμμές των πινάκων monitoring και policies κάτι το οποίο επιτυγχάνεται τροποποιώντας τα indexes των δυο τελευταίων πινάκων.

Στην επόμενη σελίδα ακολουθεί μια σχηματική αναπαράσταση των databases και πώς συνδέονται μεταξύ τους.

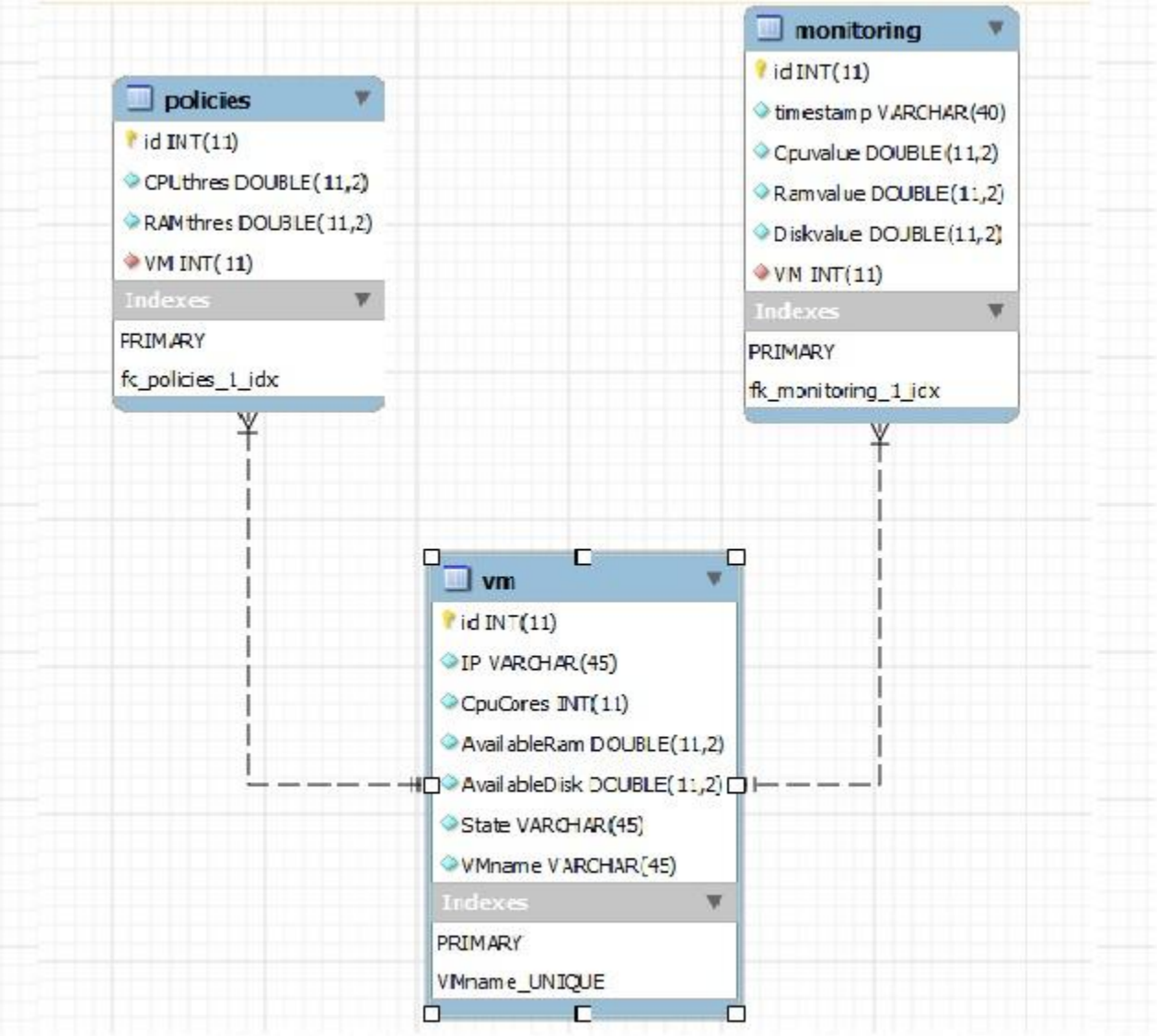


Figure 30 Databases outline and connections

3.2.3 Monitoring

Σε προηγούμενο κεφάλαιο αναφέρθηκε η λογική και η λειτουργία του monitoring το οποίο επιτυγχάνεται με το μοντέλο που υλοποιήθηκε. Παρακάτω περιγράφεται αναλυτικότερα σε επίπεδο κώδικα η διαδικασία.

Ας θεωρήσουμε ότι στην πλευρά του client έχουμε ένα σύνολο από Virtual Machines. Κάθε VM έχει το δικό του instance το οποίο τρέχει ξεχωριστά από τα υπόλοιπα. Αρχικά το VM συγκεντρώνει πληροφορίες όπως IP, CPU, RAM, σκληρό δίσκο μέσω Sigar κλάσεων. Εφόσον φτιάξουμε ένα DataAccess object χρησιμοποιούμε μεθόδους της κλάσης vm ώστε να γράψουμε όλα τα παραπάνω στην database. Αν δεν υπάρχει το VM, τότε δημιουργείται ένα entity instance και γίνεται mapping στην αντίστοιχη database. Αν υπάρχει, τότε θέτει το state του VM σε κατάσταση "UP" και γίνεται αντίστοιχη ενημέρωση στη βάση.

Στην πλευρά του Controller, όλες οι λειτουργίες του χρονοδρομολογούνται και επαναλαμβάνονται ανά ένα συγκεκριμένο χρονικό διάστημα (για την ακρίβεια 20 seconds) με τη χρήση των Timer και Timertask. Σε πρώτο στάδιο ο Controller ανακτά τη λίστα με τα VMs που έχουν καταχωρηθεί στην database. Για κάθε ένα από τα VMs της λίστας εκτελείται monitoring. Αρχικά αξιοποιώντας την IP address ενός VM μπορούμε μέσω resteasy client να ανακτήσουμε monitoring πληροφορία από το ίδιο το instance του VM μέσω json format και να την αποθηκεύσουμε στην database χρησιμοποιώντας το DataAccess object. Η monitoring πληροφορία συλλέγεται μέσω της χρήσης κλάσεων του Sigar API και αποθηκεύεται στην database monitoring. Στην καταχώρηση κάνουμε mapping μέσω javabean στο αντίστοιχο VM στο οποίο έγινε το monitoring.

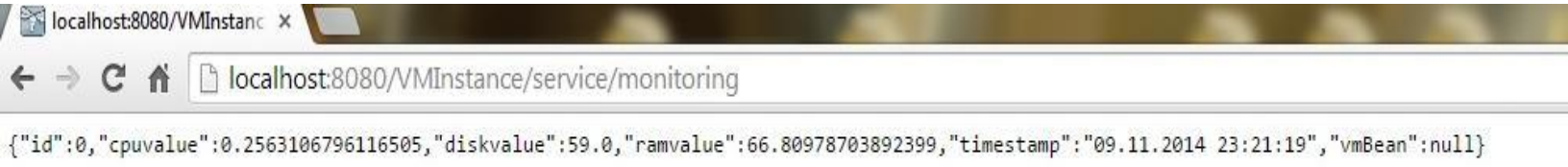


Figure 31 JSON format string έπειτα από «χτύπημα» της υπηρεσίας

Στη συνέχεια ακολουθεί το κομμάτι ελέγχου του Controller. Ανακτώντας πληροφορία από όλες τις databases και αφού εφαρμόσουμε τα thresholds της database policies στα VM stats κάνουμε σύγκριση με τα δεδομένα του monitoring.

Αν κάποιο από τα όρια παραβιαστεί τότε θέτουμε σε κατάσταση "DOWN" το συγκεκριμένο VM, μέχρι να «ξανασηκωθεί» όταν θα επιτρέπουν οι συνθήκες.

3.2.4 Αποτέλεσμα

Έχοντας εγκαταστήσει στο Eclipse τον WildFly 8.0 Runtime Server και έχοντας προσθέσει τα δύο projects του Controller και VMInstance, «ξεκινάμε» τον server.

Επειδή το Eclipse εγκαταστάθηκε τοπικά, ο υπολογιστής εργασίας αναλαμβάνει ρόλο ενός VM. Οπότε στην κονσόλα θα εκτυπωθούν αποτελέσματα σχετικά με το συγκεκριμένο μηχάνημα. Αντίστοιχα στο MySQL Workbench εγγράφονται όλες οι πληροφορίες στις vm , monitoring και policies databases.

Εκκινώντας τον server έχουμε τα εξής αποτελέσματα στην επόμενη σελίδα.

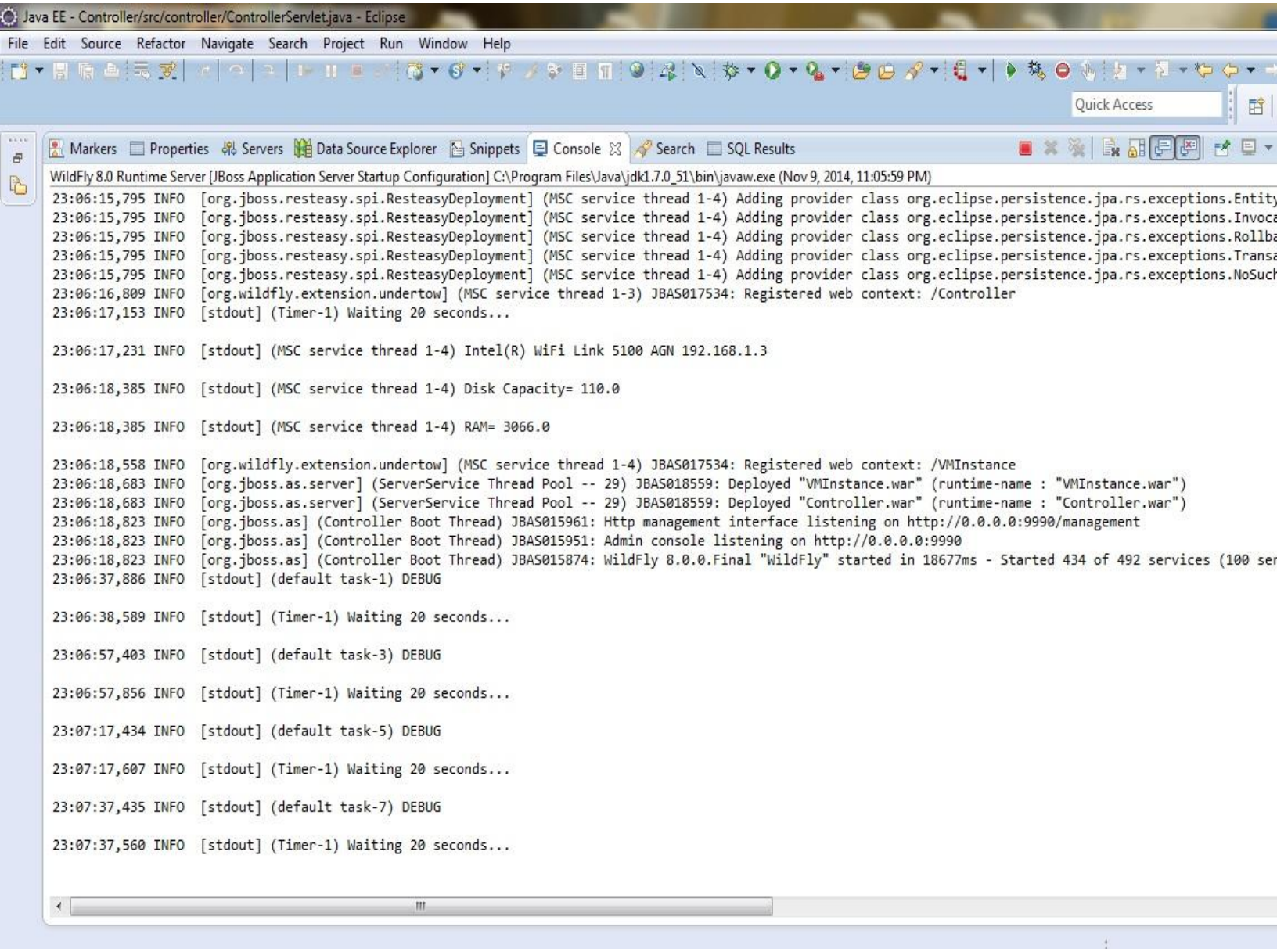


Figure 32 Output της monitoring διαδικασίας στο Eclipse

Αρχικά τυπώνονται οι πληροφορίες του Virtual Machine και ακολουθεί η monitoring διαδικασία κάθε 20 δευτερόλεπτα.

Στα επόμενα snapshots έχουν κρατηθεί κάποια δεδομένα που αποθηκεύτηκαν στις 3 databases.

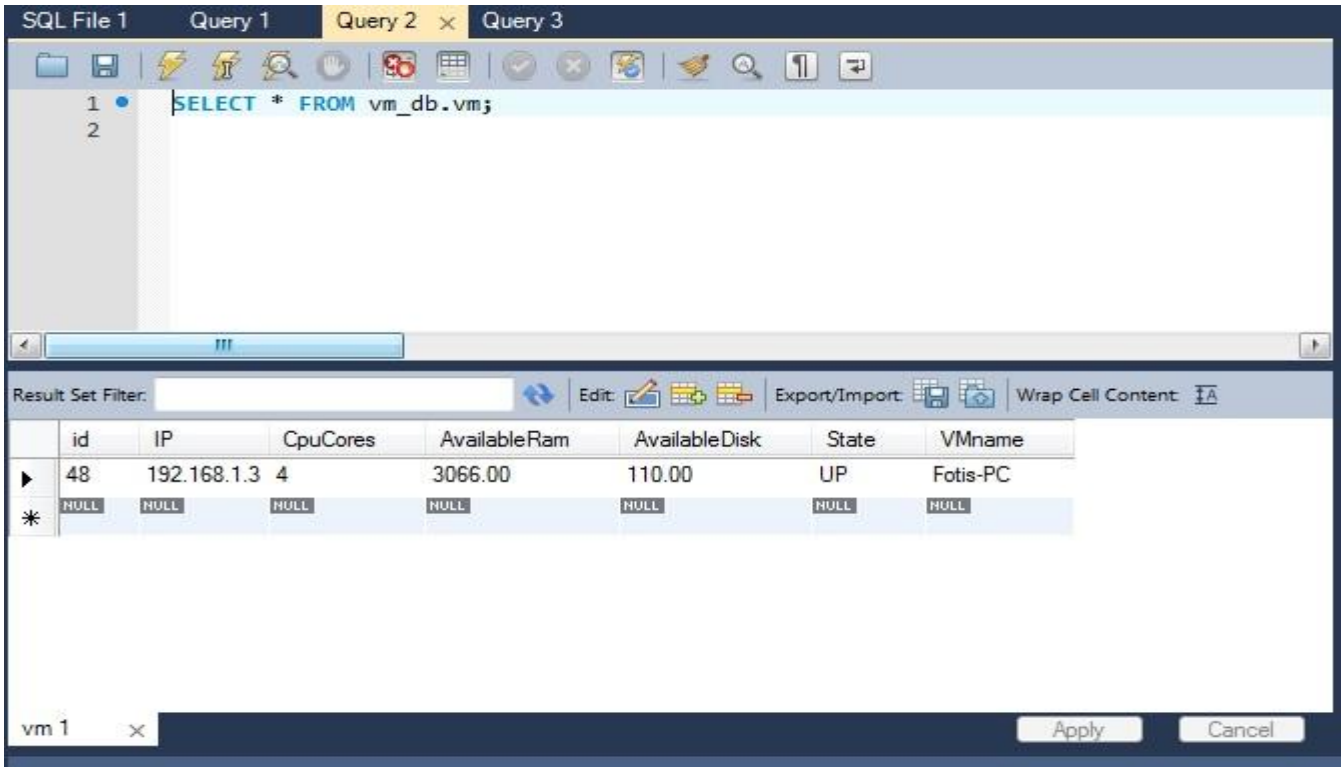


Figure 33 MySQL Workbench VM database output

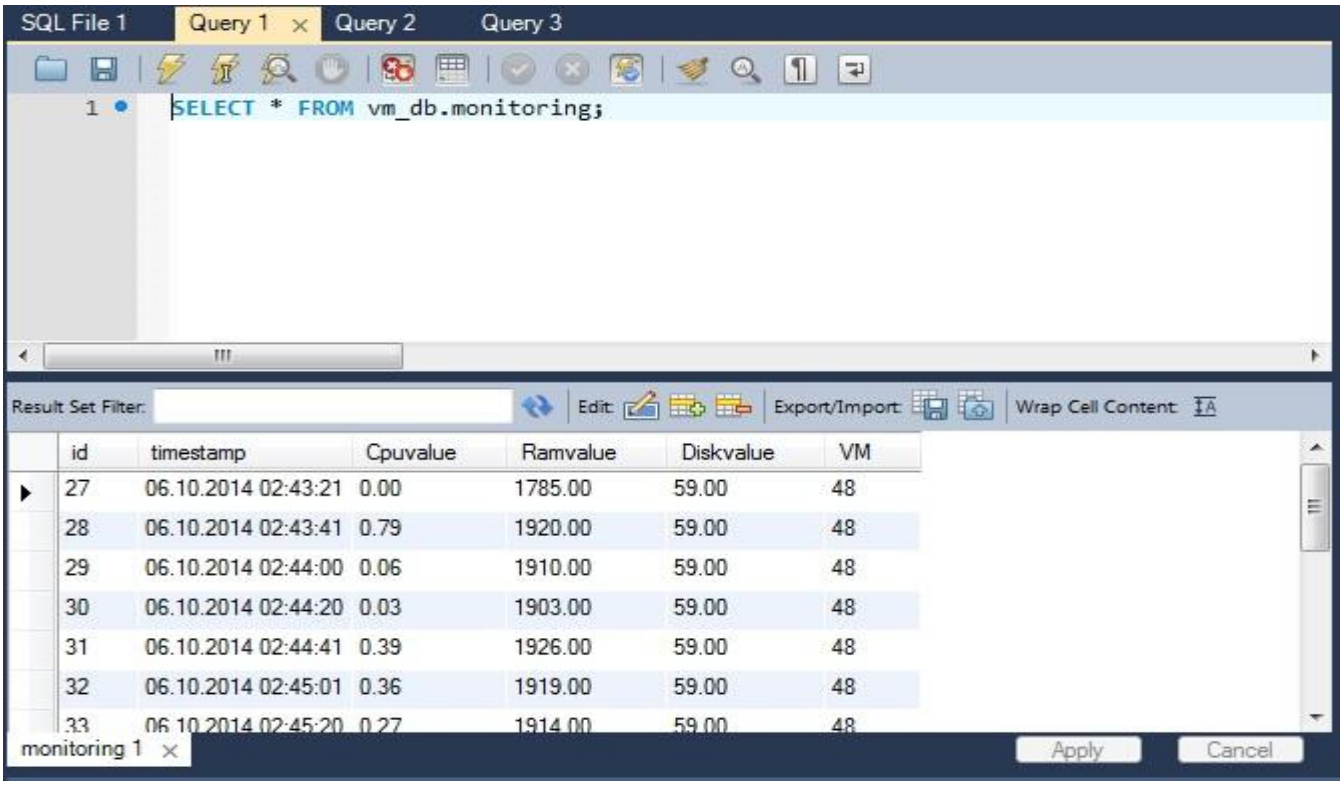


Figure 34 MySQL Workbench Monitoring database output

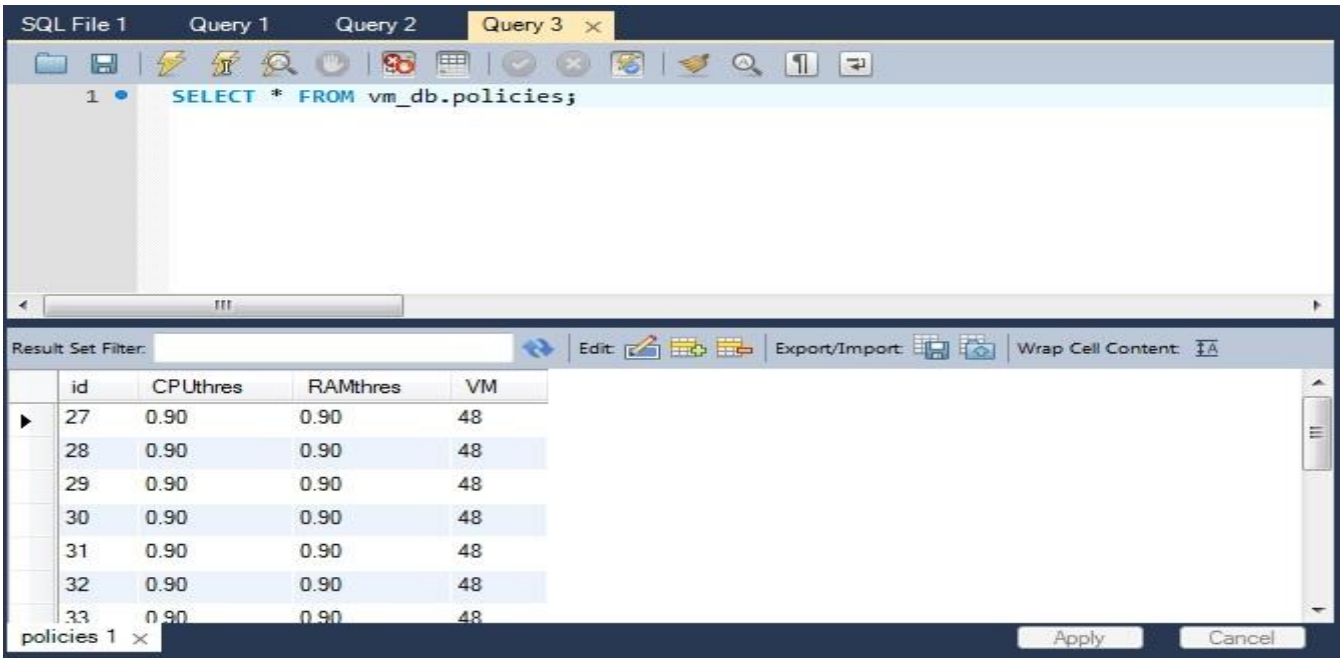


Figure 35 MySQL Workbench Policies database output

Πειραματική Διαδικασία

4.1 Αρχιτεκτονική

Η πλατφόρμα που περιγράφεται στο προηγούμενο κεφάλαιο υλοποίηθηκε σε πραγματικό επίπεδο υπό πραγματικές συνθήκες. Ακολουθεί αναλυτικό σχήμα που απεικονίζει την ολοκληρωμένη Cloud υποδομή.

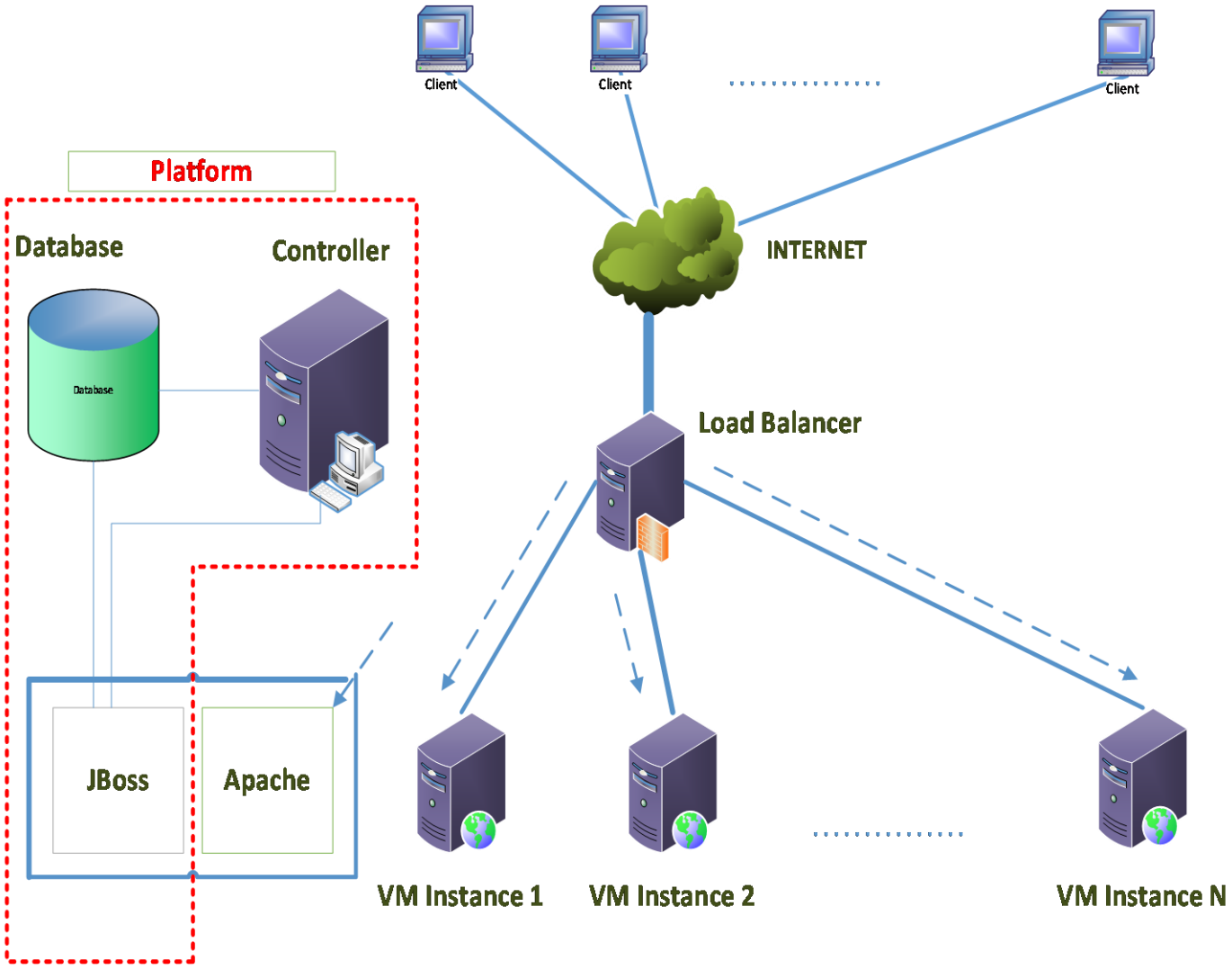


Figure 36 Ολοκληρωμένη απεικόνιση Cloud υποδομής

Αναλυτικότερα:

Η αρχιτεκτονική της υποδομής αποτελείται από τα παρακάτω δομικά στοιχεία:

- Client

  Στην τρέχουσα πειραματική διαδικασία ως client θεωρούμαστε εμείς. Ειδικότερα μέσω της εφαρμογής Apache JMeter έχουμε τη δυνατότητα τοπικά να «στήσουμε» ένα web test plan, προσομοιώνοντας πραγματικά σενάρια φόρτου (load) σε servers. Για παράδειγμα ένας συγκεκριμένος αριθμός χρηστών στέλνει συνεχώς ή για ορισμένο χρονικό διάστημα ένα ή περισσότερα HTTP requests τύπου GET «χτυπώντας» μια συγκεκριμένη http σελίδα.

- Load Balancer

  Πρόκειται για έναν απλό Cloud Server τον οποίο έχουμε configur-άρει να λειτουργεί ως ένας απλός Apache Server. Ορίζουμε ως BalancerMembers τους servers στους οποίους θα διαμοιράζει τα requests ο load balancer. Ως μέθοδο loadbalancing θα χρησιμοποιήσουμε έναν “byrequest” balancing αλγόριθμο ο οποίος είναι παρόμοιος με τον αλγόριθμο Round Robin.

- Virtual Machines / Cloud Servers

  Έχουμε στη διάθεση μας έναν αριθμό από cloud servers τους οποίους διαθέτουμε και τρέχουν πάνω από Openstack. Τους σετάρουμε ώστε να κάνουμε allocation την επιθυμητή μνήμη και τον επιθυμητό δίσκο. «Τρέχουν» τόσο ως JBoss Servers λειτουργώντας ως monitoring virtual machine instances ο καθένας όσο και ως Apache Servers ταυτόχρονα που τρέχουν ένα PHP πρόγραμμα το οποίο εκτελεί απλούς υπολογισμούς.

- Controller

  Όπως και τα Virtual Machines, ο Controller είναι και εκείνος ένας cloud server με τη διαφορά ότι τρέχει μόνο ως JBoss Server και παρέχει το κεντρικό διαχειριστικό σύστημα για τις υπηρεσίες monitoring.

- Database

  Η βάση δεδομένων χρησιμοποιεί την ίδια IP address με τον Controller αλλά «ακούει» σε διαφορετικό port.

4.2 Εξοπλισμός

Στη διάθεση μας έχουμε ένα δυναμικό αριθμό από cloud servers ο οποίος μπορεί να αυξομειωθεί αναλόγως τις ανάγκες της πειραματικής διαδικασίας. Για λόγους ευκολίας όλος ο κώδικας σχετικά με το configuration παρατίθεται στο Παράρτημα.

Τα VMs έχουν τα εξής χαρακτηριστικά:

- ✓ Εύρος IP διευθύνσεων 10.0.1.6:8080 εως 10.0.1.7:8080
- ✓ CPU 1 core 3 GHz
- ✓ RAM = 2 GB
- ✓ Disk = 20 GB
- ✓ JBoss Web Server

  Εγκατεστημένος στο path /opt/wildfly-8.1.0.Final/ . «Ακούει» στην port 9990. Αναγκαίες τροποποιήσεις έγιναν στο standalone.xml .
- ✓ Apache Web Server

  Εγκατεστημένος στο path /etc/apache2/ . «Ακούει» στην port 80.

- ✓ PHP

  Το αρχείο calc.php βρίσκετα στο /var/www/html/ .
- ✓ VM Instance

  Στο url http:/10.0.1.x:9990/console/App.html#deployments (όπου x το τελευταίο octet της IP address ενός VM) προσθέτουμε το VMInstance.war το οποίο περιέχει τον αντίστοιχο κώδικα που περιγράφηκε στο προηγούμενο κεφάλαιο.

Ως Controller χρησιμοποιούμε ένα virtual machine με ανάλογα χαρακτηριστικά όπως παραπάνω και μικρές διαφορές.

- ✓ IP διεύθυνση 10.0.1.5:8080
- ✓ Controller Instance

  Στο url http:/10.0.1.5:9990/console/App.html#deployments προσθέτουμε το Controller.war το οποίο περιέχει τον αντίστοιχο κώδικα που περιγράφηκε στο προηγούμενο κεφάλαιο.
- ✓ Database

  Κάνουμε login στην database μέσω MySql Workbench χρησιμοποιώντας την ίδια IP address του Controller αλλά με port 3306. Έχουμε ήδη δημιουργήσει τα schemas των databases που έχουν περιγραφεί στο προηγούμενο κεφάλαιο.

Ως Load Balancer χρησιμοποιήσαμε ένα virtual machine με τη διαφορά ότι δεν έχει εγκατασταθεί JBoss ή κάτι επιπρόσθετο εκτός από τον Apache Web Server.

- ✓ IP διεύθυνση 10.0.1.8:80
- ✓ Apache Web Server configuration

  Εγκατεστημένος στο path /etc/apache2/ . «Ακούει» στην port 80. Αναγκαίες τροποποιήσεις πραγματοποιήθηκαν στο 000-default.conf .

Ο Apache JMeter ο οποίος θα αναλάβει ρόλο client(s) έχει εγκατασταθεί τοπικά στον υπολογιστή εργασίας.

4.3 Σενάρια

Πριν από την εκτέλεση κάθε σεναρίου συνδεόμαστε στα μηχανήματα. Επίσης συνδεόμαστε στο MySQL Workbench χρησιμοποιώντας την IP address 10.0.1.5 αλλά μέσω του port 3306.

Αρχικά ξεκινάμε τον JBoss server στα VM instances τρέχοντας το script standalone.sh . Έχουμε φροντίσει προηγουμένως να ρυθμίσουμε σωστα το path μέσω της εντολής

export JAVA_OPTS=”-Djava.library.path=/home/Ubuntu/

όπως επίσης κάνουμε restart τον Apache server με apachectl –k restart, κάτι το οποίο επαναλαμβάνεται κάθε φορά ενδιάμεσα των ποικίλων σεναρίων της πειραματικής διαδικασίας.

Στη συνέχεια τρέχουμε τον JBoss server στον Controller και αυτομάτως ξεκινά το monitoring των VM instances που έχουμε διαθέσει.

Για την επίτευξη του στόχου μας, ξεκινάμε και τον Apace JMeter και ορίζουμε το configuration του web test plan που θέλουμε να τρέξουμε.

Η πειραματική διαδικασία αποτελείται από τα παρακάτω ***σενάρια***:

- 1ο Σενάριο

  Αριθμός VMs: 2
  PHP loop counter: 50000
  JMeter Configuration:
    Users: 5
    Ramp-up period: 0
    Loop: Forever
    URL: 10.0.1.8/calc

- 2ο Σενάριο

  Αριθμός VMs: 2

  PHP loop counter: 50000

  JMeter Configuration:

    Users: 20

    Ramp-up period: 0

    Loop: Forever

    URL: 10.0.1.8/calc

- 3^{ο} Σενάριο

  Αριθμός VMs: 2

  PHP loop counter: 50000

  JMeter Configuration:

    Users: 50

    Ramp-up period: 0

    Loop: Forever

    URL: 10.0.1.8/calc

- 4^{0} Σενάριο

  Αριθμός VMs: 2

  PHP loop counter: 50000

  JMeter Configuration:

    Users: 100

    Ramp-up period: 0

    Loop: Forever

    URL: 10.0.1.8/calc

- 5ο Σενάριο

  Αριθμός VMs: 2

  PHP loop counter: 80000

  JMeter Configuration:

    Users: 5

    Ramp-up period: 0

    Loop: Forever

    URL: 10.0.1.8/calc

- 6ο Σενάριο

  Αριθμός VMs: 2

  PHP loop counter: 80000

  JMeter Configuration:

    Users: 20

    Ramp-up period: 0

    Loop: Forever

    URL: 10.0.1.8/calc

- 7ο Σενάριο

  Αριθμός VMs: 4

  PHP loop counter: 80000

  JMeter Configuration:

    Users: 100

    Ramp-up period: 0

    Loop: Forever

    URL: 10.0.1.8/calc

- 8ο Σενάριο

Αριθμός VMs: 4

PHP loop counter: 160000

JMeter Configuration:

Users: 100

Ramp-up period: 0

Loop: Forever

URL: 10.0.1.8/calc

Table 1 Σενάρια πειραματικής διαδικασίας

Σενάρια	1	2	3	4	5	6	7	8
Αριθμός VM	2						4	
PHP Loop counter	50000				80000			160000
Users	5	20	50	100	5	20	100	100
Ramp-up period	0							
Loop	Forever							
URL	10.0.1.8/calc							

4.4 Αποτελέσματα

Τα σενάρια και τα αποτελέσματά τους μπορούν να κατηγοριοποιηθούν με κριτήριο πρώτον τον αριθμό των διαθέσιμων virtual machines, δεύτερον τις επαναλήψεις του php κώδικα και τρίτον τον αριθμό των χρηστών/threads. Τα αποτελέσματα της διαδικασίας του monitoring παρατίθενται σε μορφή ποσοστού.

Virtual Machines = 2

Τα στοιχεία των εικονικών μηχανημάτων φαίνονται στον παρακάτω πίνακα.

Table 2 2-VM Instances

id	IP	CpuCores	AvailableRam	AvailableDisk	State	VMname
167	10.0.1.6	1	2001	19	DOWN	mon-inst-1
168	10.0.1.7	1	2001	19	DOWN	mon-inst-2

Τα ανώτατα όρια/thresholds που ακολουθούν τις πολιτικές monitoring που έχουμε εφαρμόσει φαίνονται στον παρακάτω πίνακα υπό την μορφή (ποσοστό/100).

Table 3 2-VM Instances Policy

id	CPUthres	RAMthres	VM
643	0.9	0.9	167
644	0.9	0.9	168

Loop = 50000

5 Users

Table 4 2-VM Instances Monitoring, 50000 loop, 5 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2159	14.12.2014 15:46:48	2	39.78	10	167
2160	14.12.2014 15:46:49	2	39.38	10	168
2161	14.12.2014 15:47:07	0	39.68	10	167
2162	14.12.2014 15:47:08	0	39.23	10	168
2163	14.12.2014 15:47:27	0	39.88	10	167
2164	14.12.2014 15:47:28	2	39.28	10	168
2165	14.12.2014 15:47:47	37.25	40.38	10	167
2166	14.12.2014 15:47:48	38.78	39.83	10	168
2167	14.12.2014 15:48:07	62	40.53	10	167
2168	14.12.2014 15:48:08	64	40.03	10	168
2169	14.12.2014 15:48:27	38.78	40.63	10	167

2170	14.12.2014 15:48:28	38	40.13	10	168
2171	14.12.2014 15:48:47	41.18	40.63	10	167
2172	14.12.2014 15:48:48	64.58	40.23	10	168
2173	14.12.2014 15:49:07	44	40.68	10	167
2174	14.12.2014 15:49:08	52	40.23	10	168
2175	14.12.2014 15:49:27	42.86	40.68	10	167
2176	14.12.2014 15:49:28	51.02	40.28	10	168
2177	14.12.2014 15:49:47	39.58	40.73	10	167
2178	14.12.2014 15:49:48	46.94	40.28	10	168
2179	14.12.2014 15:50:07	30	40.78	10	167
2180	14.12.2014 15:50:08	48	40.28	10	168
2181	14.12.2014 15:50:27	42	40.83	10	167
2182	14.12.2014 15:50:28	43.75	40.28	10	168
2183	14.12.2014 15:50:47	44	40.83	10	167
2184	14.12.2014 15:50:48	48.98	40.33	10	168
2185	14.12.2014 15:51:07	47.06	40.83	10	167

2186	14.12.2014 15:51:08	50.98	40.33	10	168
2187	14.12.2014 15:51:27	44	40.88	10	167
2188	14.12.2014 15:51:28	52	40.33	10	168

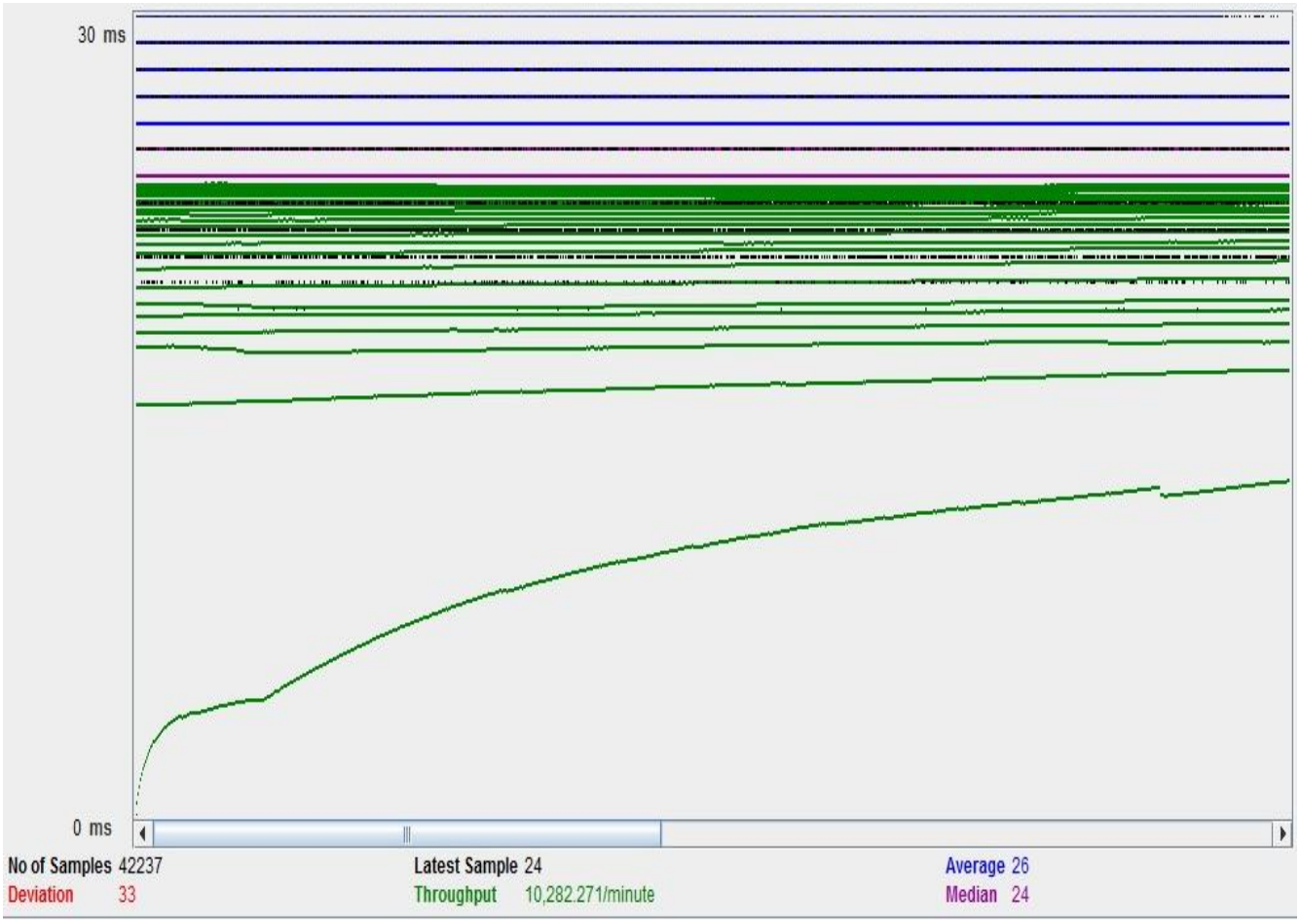


Figure 37 JMeter Graph 2-VMs, 50000 loop, 5 users

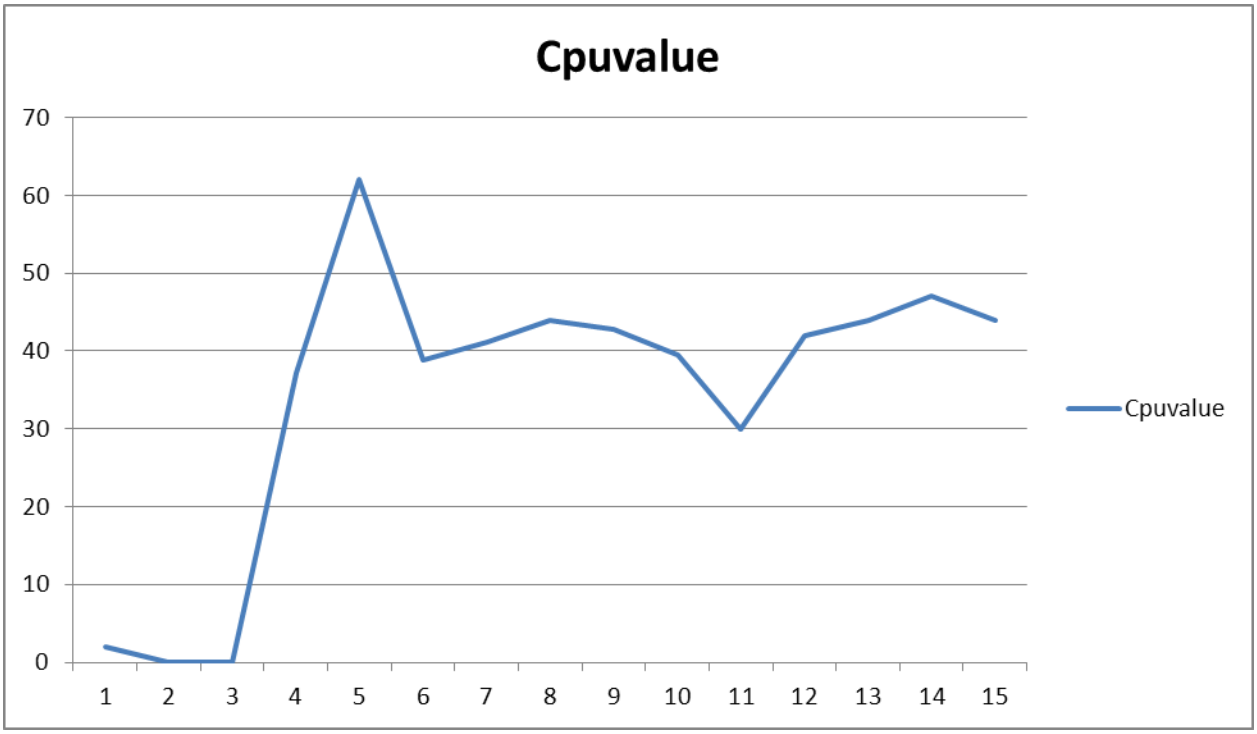


Figure 38 CPU Percentage Instace 1, 2-VMs, 50000 loop, 5 users

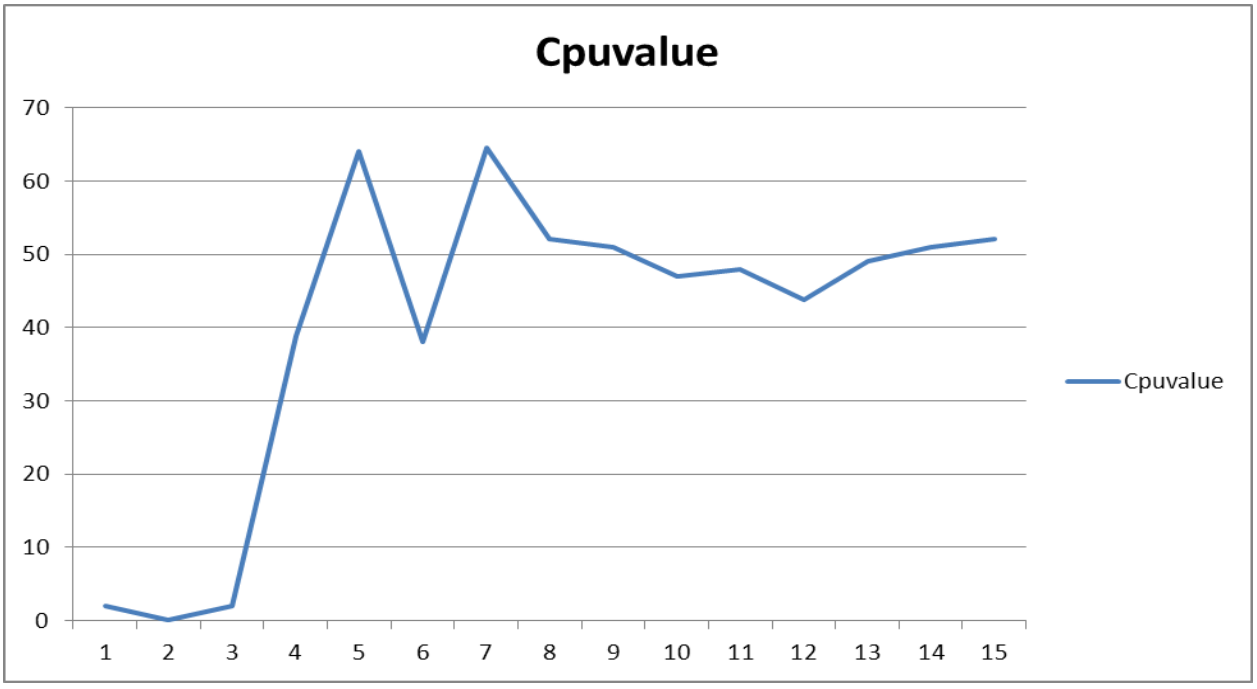


Figure 39 CPU Percentage Instace 2, 2-VMs, 50000 loop, 5 users

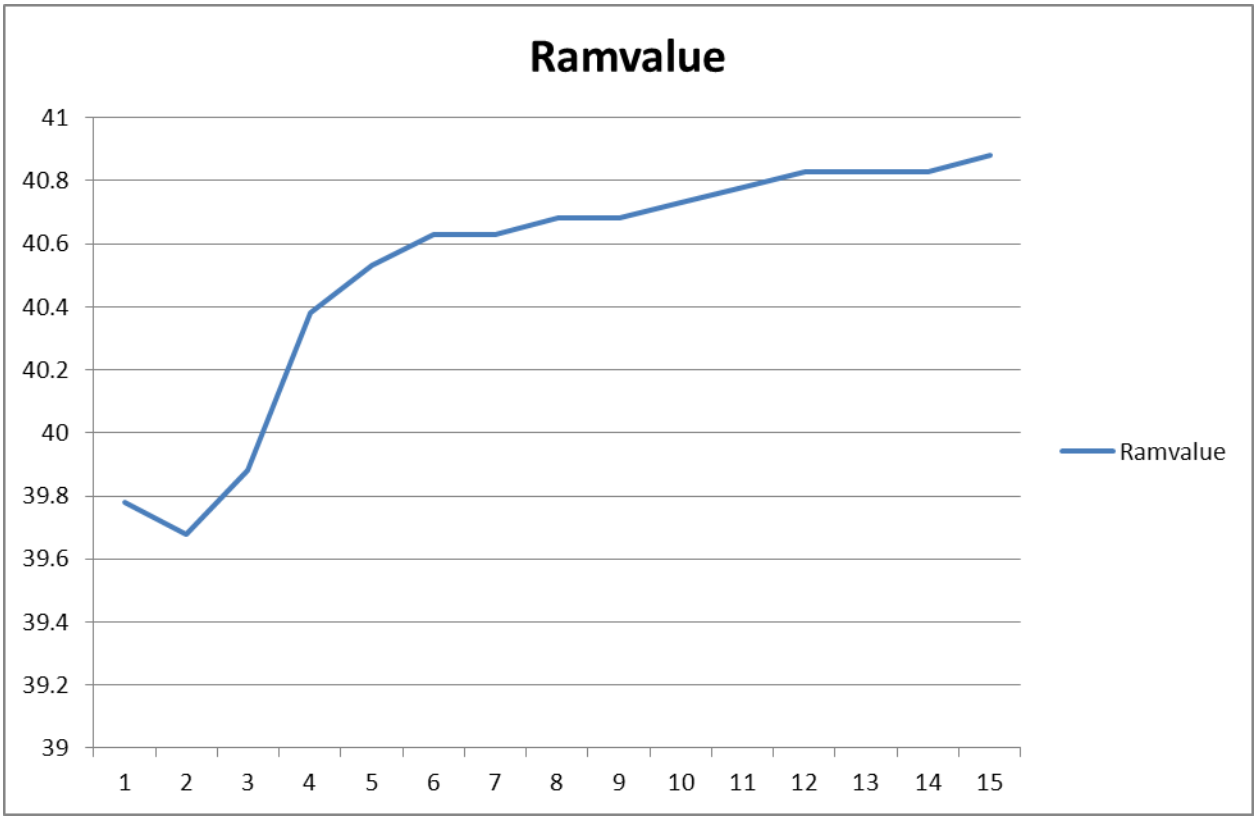


Figure 40 RAM Percentage Instace 1, 2-VMs, 50000 loop, 5 users

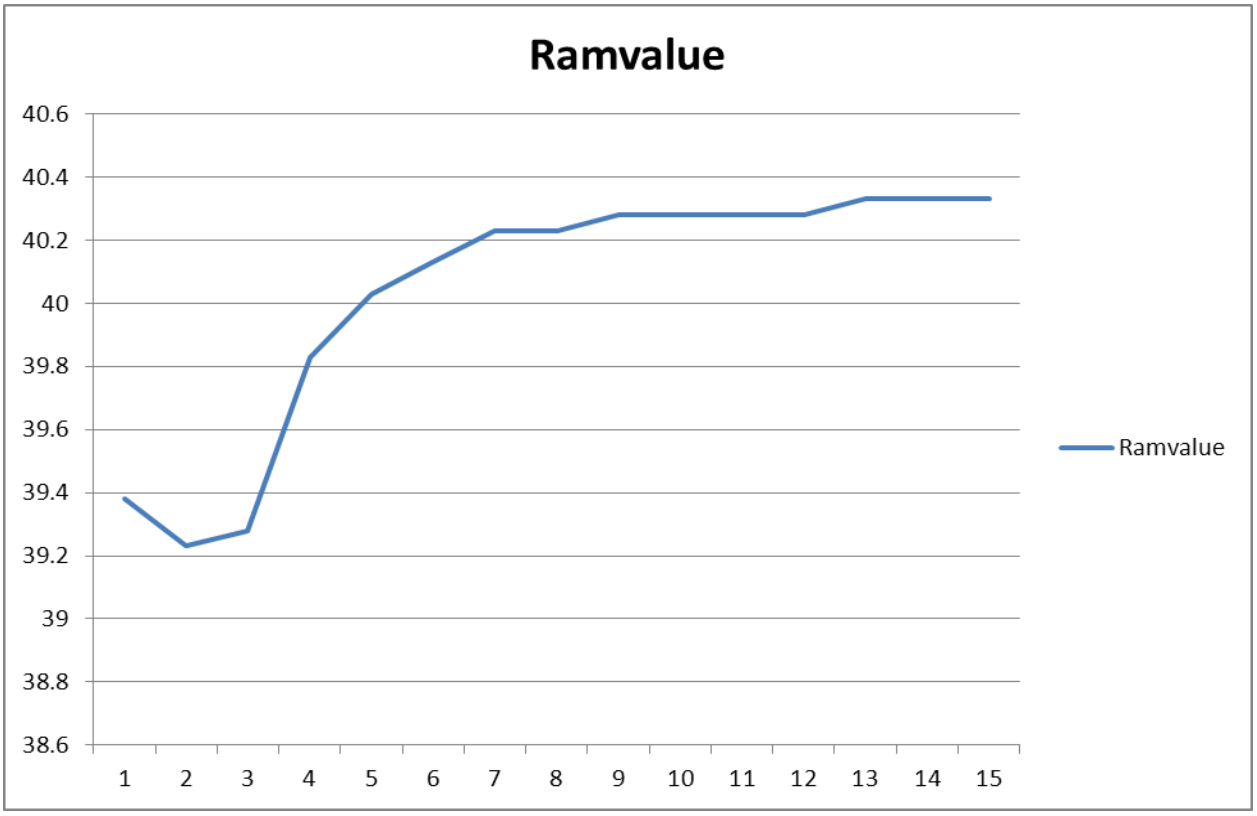


Figure 41 RAM Percentage Instace 2, 2-VMs, 50000 loop, 5 users

20 Users

Table 5 2-VM Instances Monitoring, 50000 loop, 20 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2189	14.12.2014 15:56:50	6	40.28	10	169
2190	14.12.2014 15:56:51	4	39.48	10	170
2191	14.12.2014 15:57:09	2	40.48	10	169
2192	14.12.2014 15:57:10	0	39.78	10	170
2193	14.12.2014 15:57:29	52.94	41.18	10	169
2194	14.12.2014 15:57:30	76	40.53	10	170
2195	14.12.2014 15:57:49	61.22	41.43	10	169
2196	14.12.2014 15:57:50	76	40.78	10	170
2197	14.12.2014 15:58:09	50	41.48	10	169
2198	14.12.2014 15:58:10	70	40.83	10	170
2199	14.12.2014 15:58:29	59.18	41.48	10	169
2200	14.12.2014 15:58:30	76	40.68	10	170

2201	14.12.2014 15:58:49	50	41.53	10	169
2202	14.12.2014 15:58:50	70	40.68	10	170
2203	14.12.2014 15:59:09	62	41.58	10	169
2204	14.12.2014 15:59:10	76	40.78	10	170
2205	14.12.2014 15:59:29	64.71	41.43	10	169
2206	14.12.2014 15:59:30	72.55	40.83	10	170
2207	14.12.2014 15:59:49	62.75	41.48	10	169
2208	14.12.2014 15:59:50	74.51	40.83	10	170
2209	14.12.2014 16:00:09	62.75	41.48	10	169
2210	14.12.2014 16:00:10	58	40.83	10	170
2211	14.12.2014 16:00:29	57.14	41.48	10	169
2212	14.12.2014 16:00:30	64	40.88	10	170
2213	14.12.2014 16:00:49	62	41.53	10	169
2214	14.12.2014 16:00:50	77.08	40.88	10	170
2215	14.12.2014	48	41.53	10	169

	16:01:09				
2216	14.12.2014 16:01:10	57.14	40.93	10	170
2217	14.12.2014 16:01:29	42	41.58	10	169
2218	14.12.2014 16:01:30	50	40.93	10	170

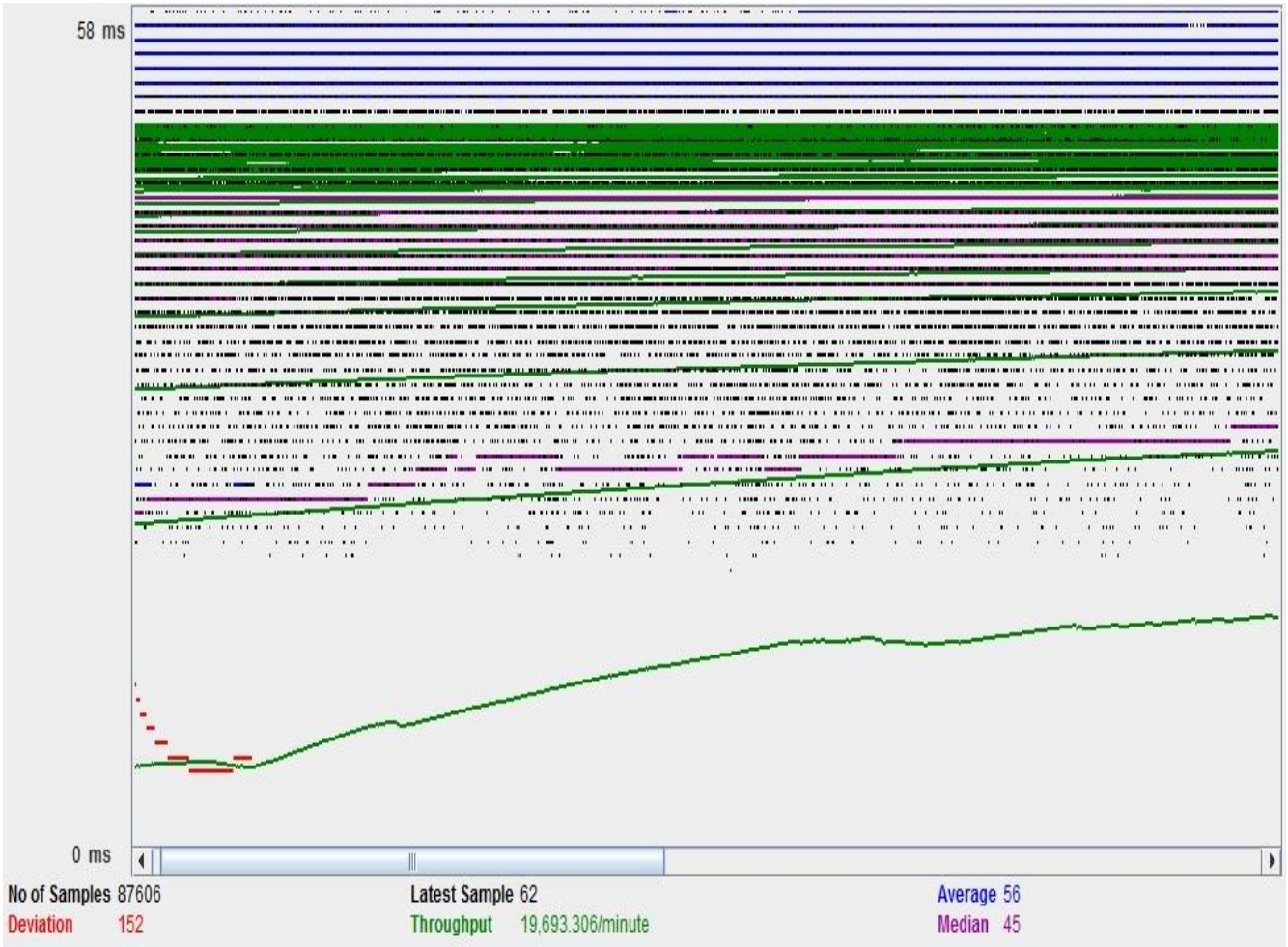


Figure 42 JMeter Graph 2-VMs, 50000 loop, 20 users

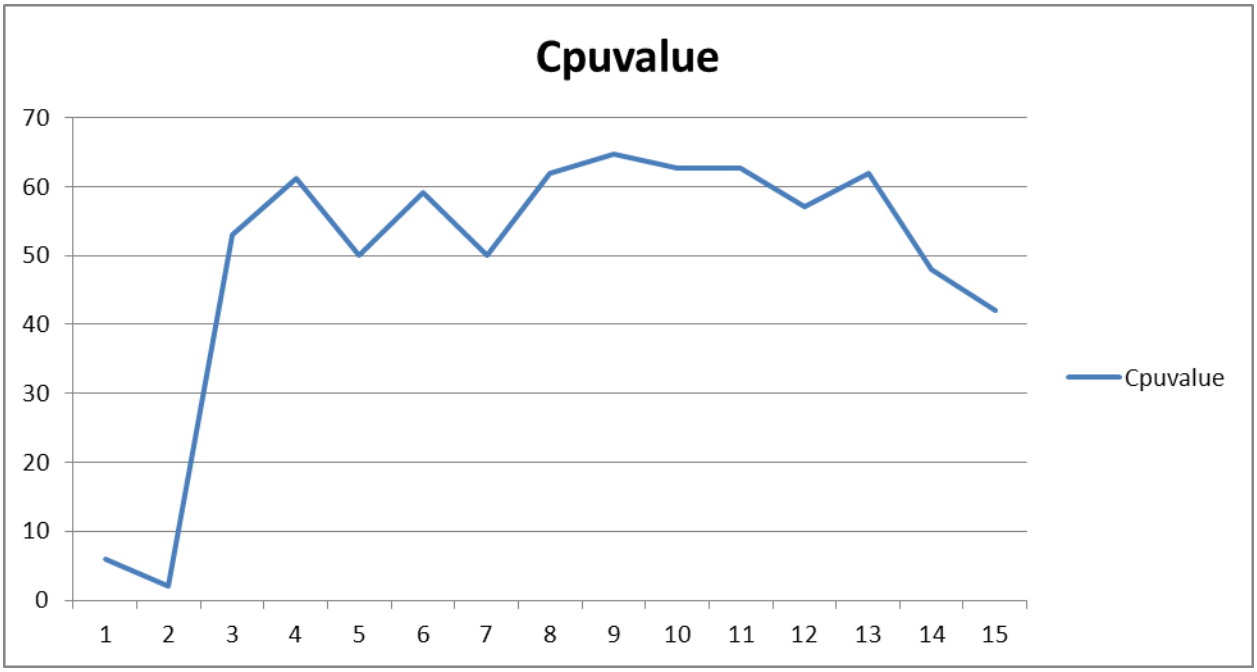


Figure 43 CPU Percentage Instace 1, 2-VMs, 50000 loop, 20 users

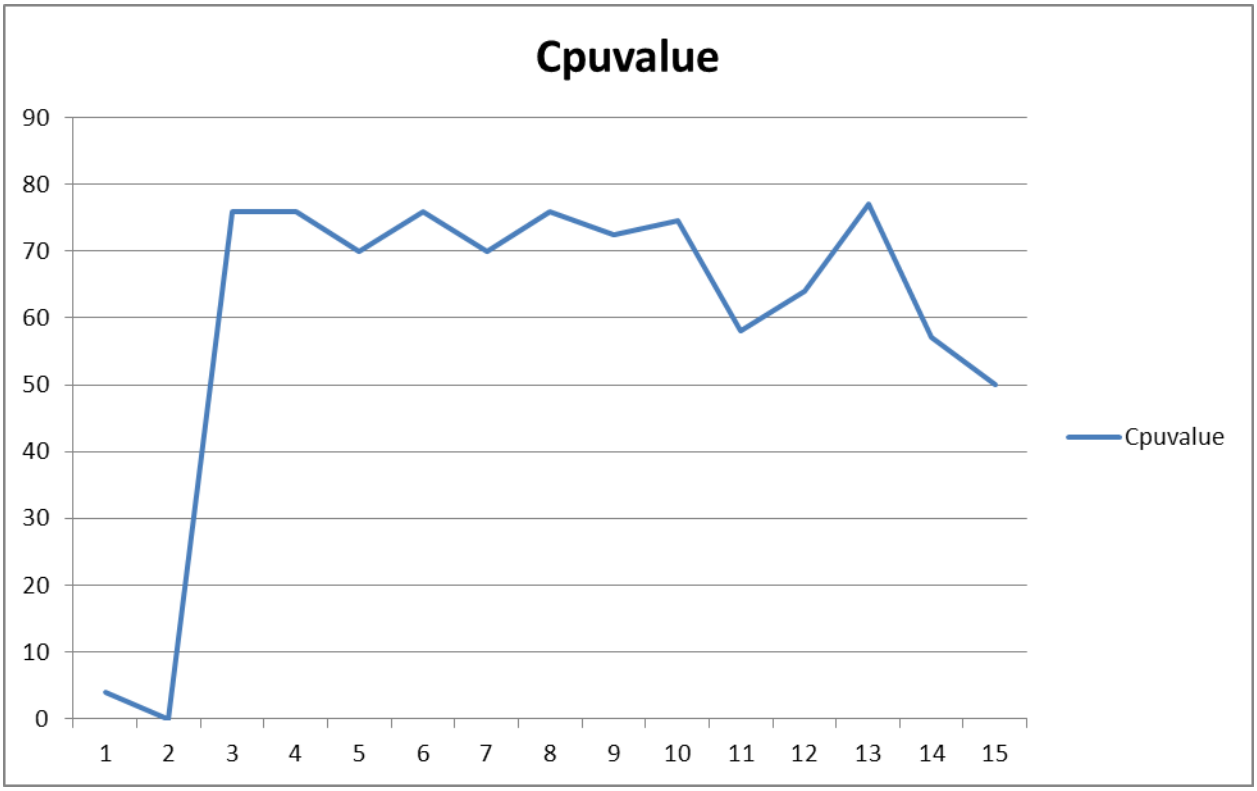


Figure 44 CPU Percentage Instace 2, 2-VMs, 50000 loop, 20 users

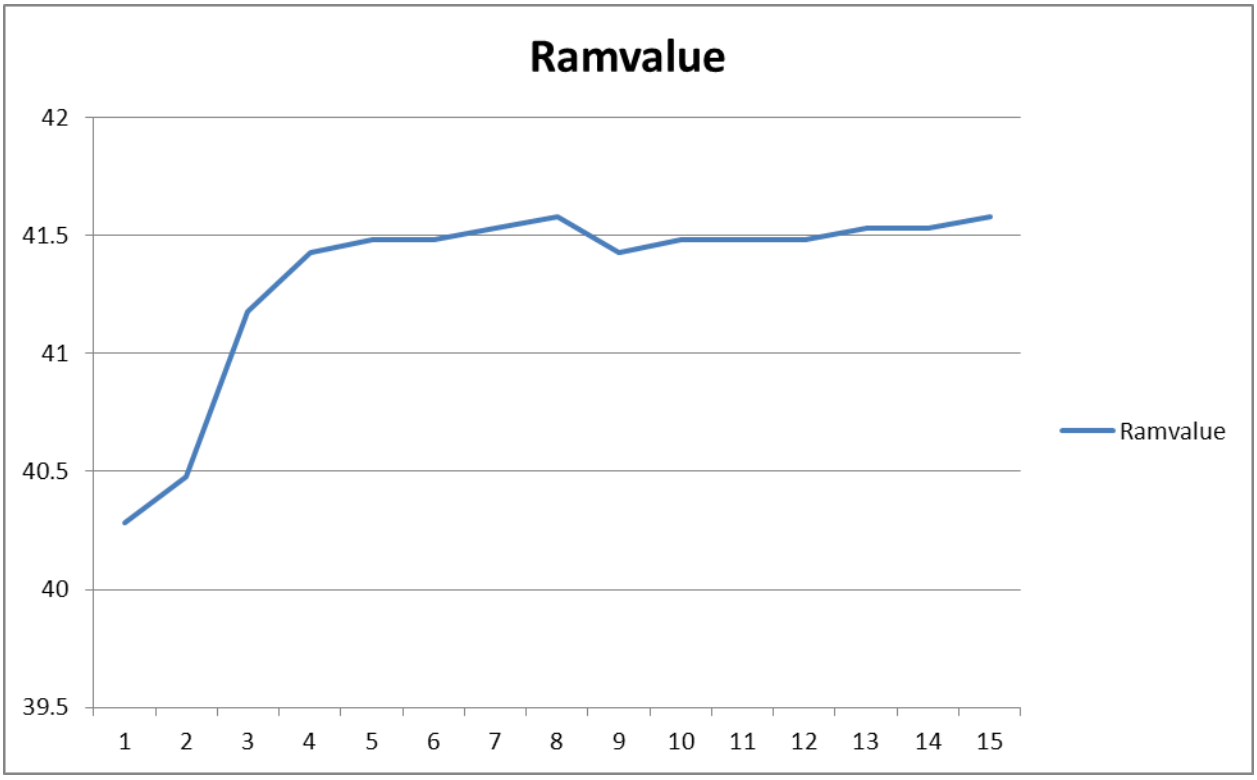


Figure 45 RAM Percentage Instace 1, 2-VMs, 50000 loop, 20 users

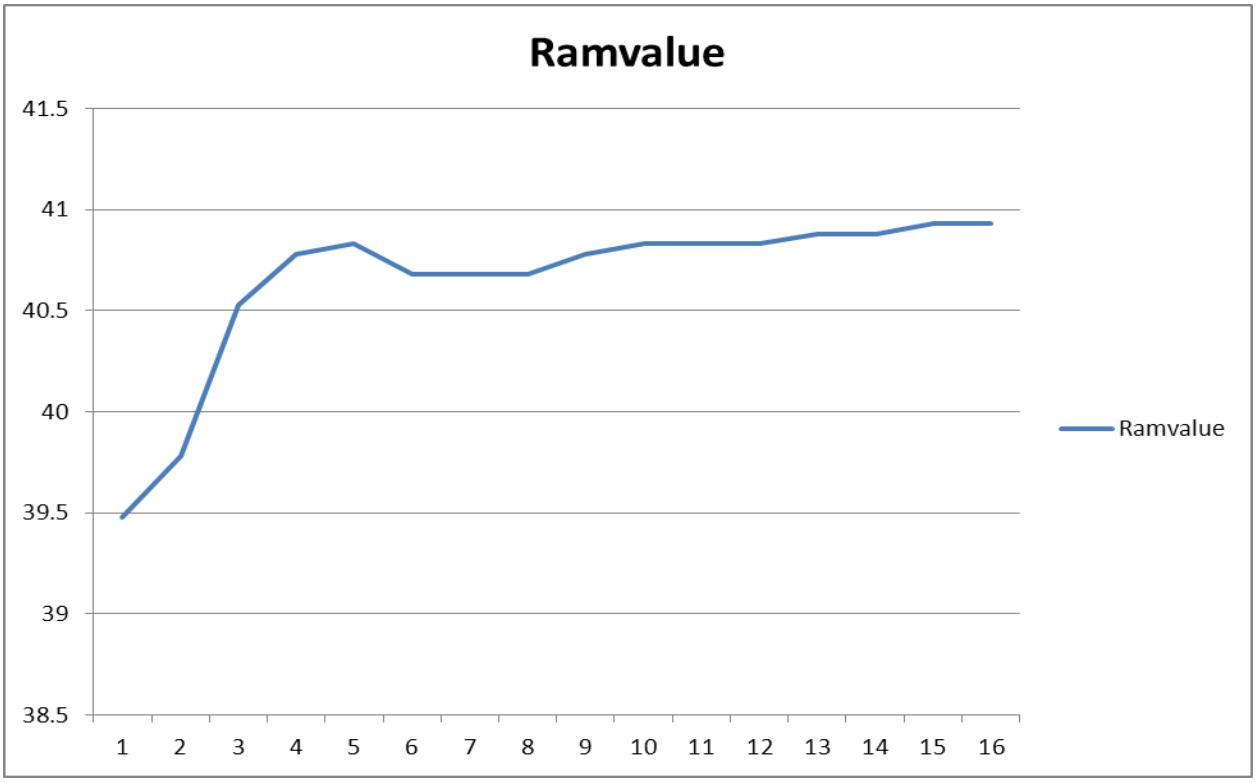


Figure 46 RAM Percentage Instace 2, 2-VMs, 50000 loop, 20 users

50 Users

Table 6 2-VM Instances Monitoring, 50000 loop, 50 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2219	14.12.2014 16:06:38	0	39.53	10	171
2220	14.12.2014 16:06:39	2.04	39.88	10	172
2221	14.12.2014 16:06:58	4	39.73	10	171
2222	14.12.2014 16:06:58	0	40.08	10	172
2223	14.12.2014 16:07:18	43.75	41.58	10	171
2224	14.12.2014 16:07:19	46	42.18	10	172
2225	14.12.2014 16:07:38	62	41.88	10	171
2226	14.12.2014 16:07:38	69.39	42.33	10	172
2227	14.12.2014 16:07:58	62	41.93	10	171
2228	14.12.2014 16:07:58	74	42.23	10	172
2229	14.12.2014 16:08:18	62	42.03	10	171
2230	14.12.2014 16:08:18	72	42.43	10	172
2231	14.12.2014 16:08:38	53.06	42.08	10	171
2232	14.12.2014 16:08:38	62.75	42.48	10	172
2233	14.12.2014 16:08:58	62	42.13	10	171
2234	14.12.2014 16:08:59	72	42.53	10	172
2235	14.12.2014 16:09:18	56	42.28	10	171
2236	14.12.2014 16:09:18	74	42.68	10	172
2237	14.12.2014 16:09:38	60.78	42.28	10	171

2238	14.12.2014 16:09:38	70.59	42.73	10	172
2239	14.12.2014 16:09:58	60	42.33	10	171
2240	14.12.2014 16:09:58	74.51	42.73	10	172
2241	14.12.2014 16:10:18	61.22	42.43	10	171
2242	14.12.2014 16:10:18	67.35	42.83	10	172
2243	14.12.2014 16:10:38	62.75	42.48	10	171
2244	14.12.2014 16:10:38	70	42.83	10	172
2245	14.12.2014 16:10:58	56	42.48	10	171
2246	14.12.2014 16:10:58	65.31	42.88	10	172
2247	14.12.2014 16:11:18	57.14	42.48	10	171
2248	14.12.2014 16:11:18	68.63	42.88	10	172

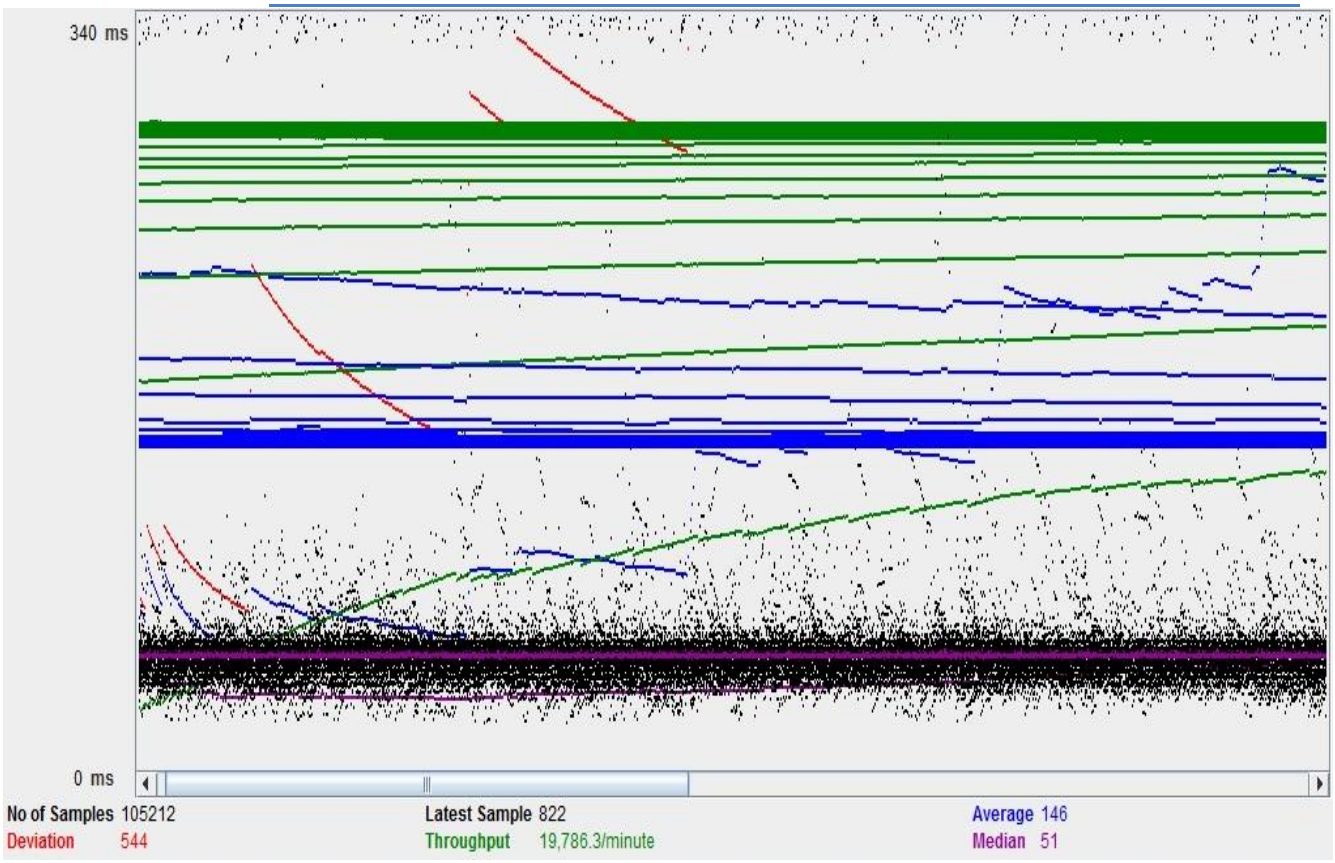


Figure 47 JMeter Graph 2-VMs, 50000 loop, 50 users

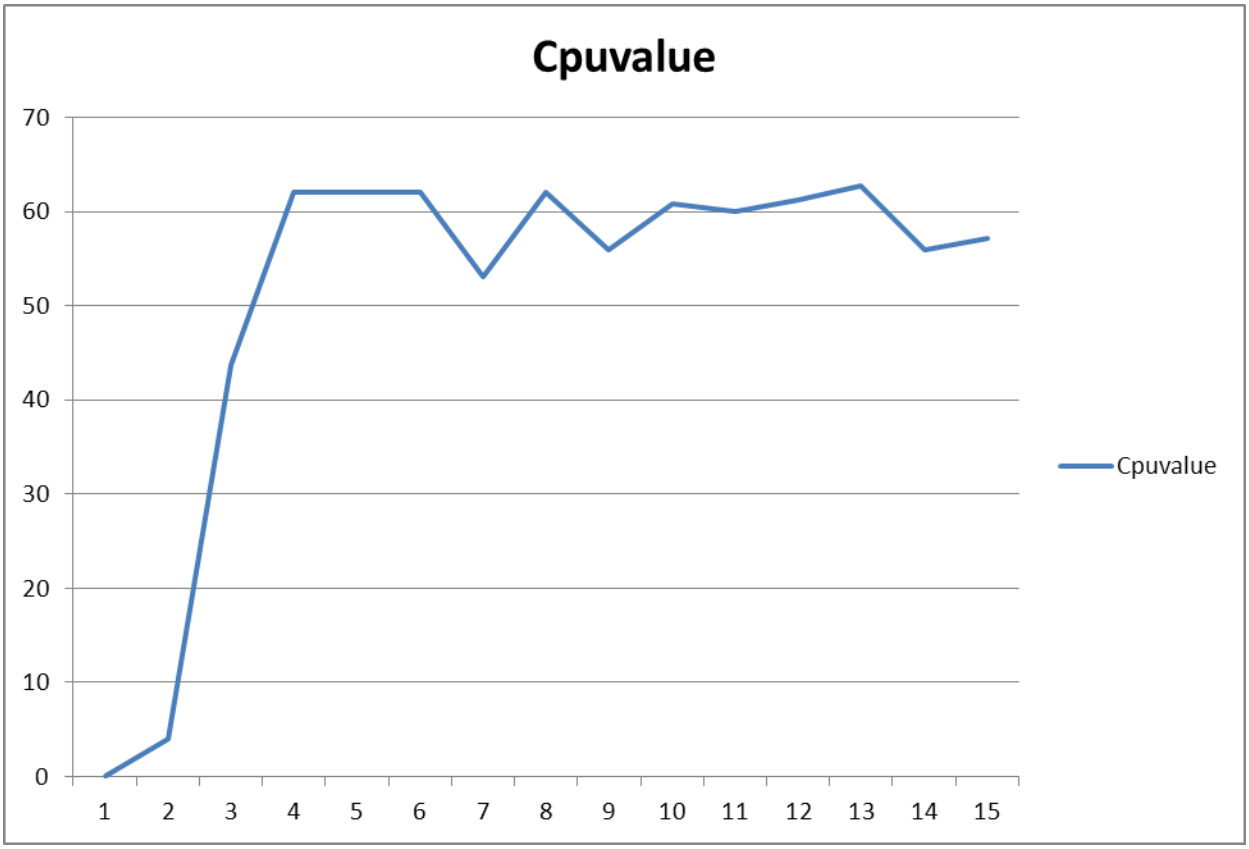


Figure 48 CPU Percentage Instace 1, 2-VMs, 50000 loop, 50 users

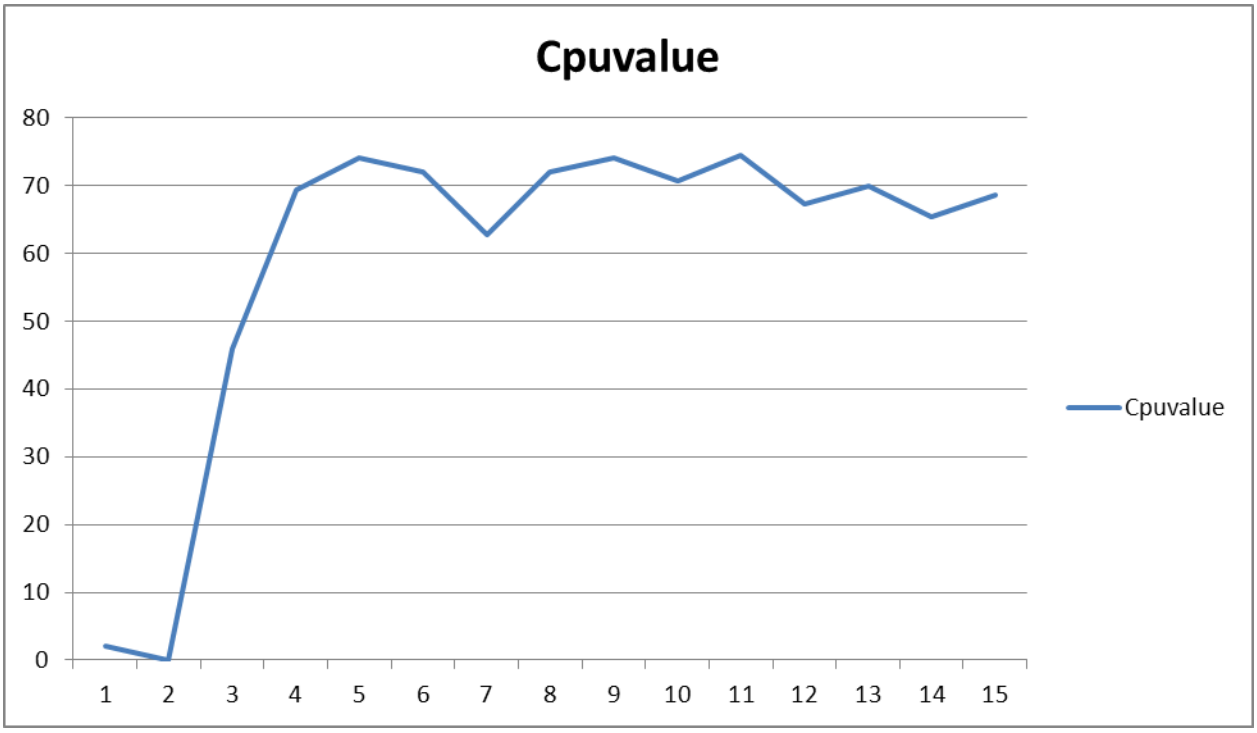


Figure 49 CPU Percentage Instace 2, 2-VMs, 50000 loop, 50 users

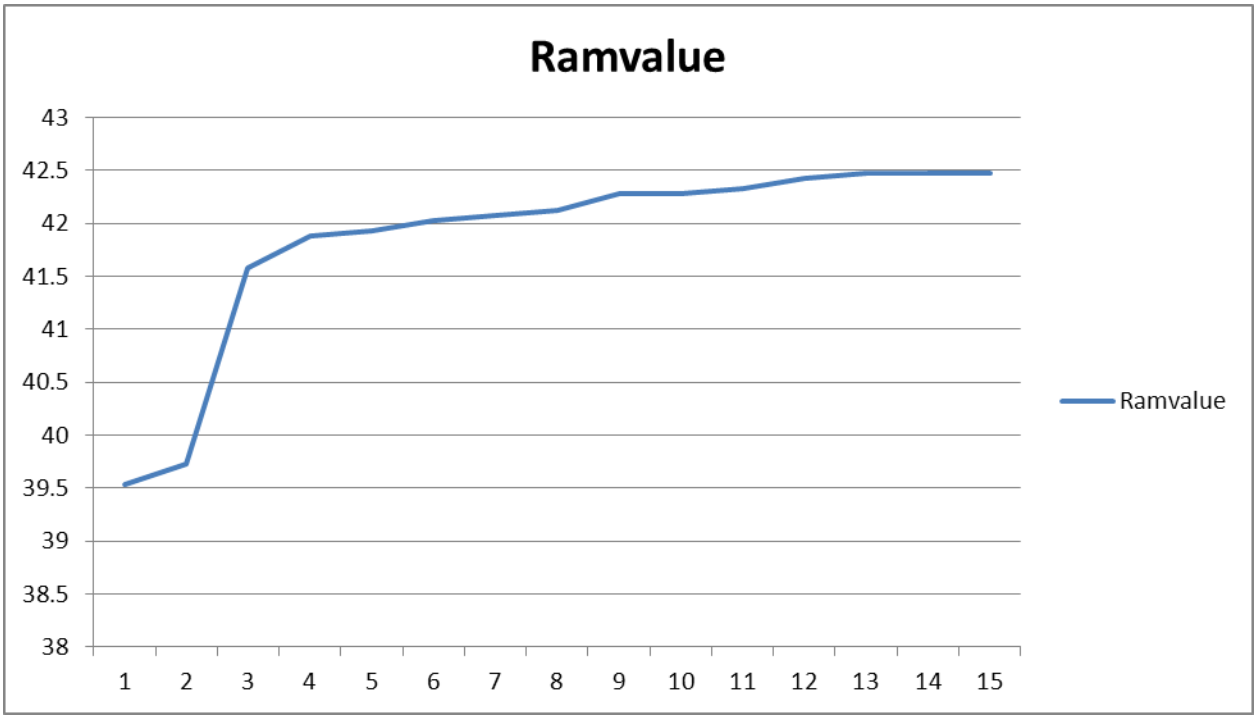


Figure 50 RAM Percentage Instace 1, 2-VMs, 50000 loop, 50 users

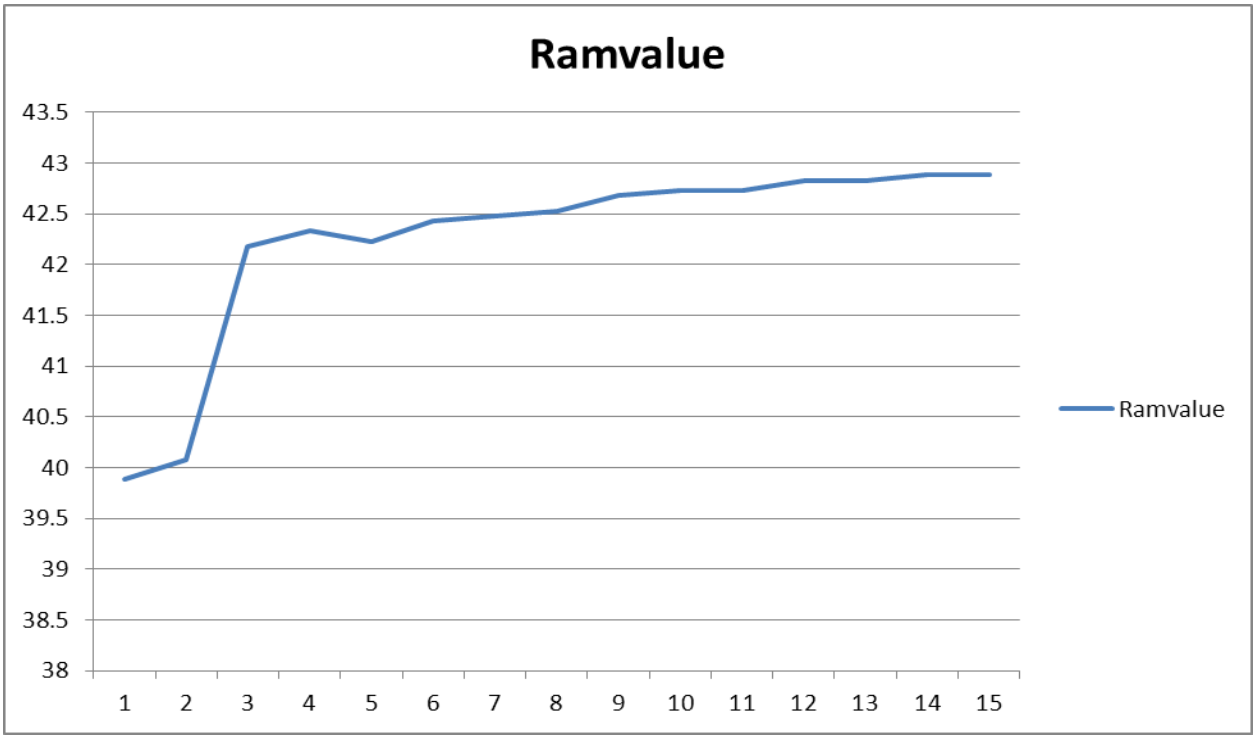


Figure 51 RAM Percentage Instace 2, 2-VMs, 50000 loop, 50 users

100 Users

Table 7 2-VM Instances Monitoring, 50000 loop, 100 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2285	14.12.2014 16:54:47	3.92	40.13	10	175
2286	14.12.2014 16:54:49	11.54	40.63	10	176
2287	14.12.2014 16:55:07	0	40.48	10	175
2288	14.12.2014 16:55:08	0	40.73	10	176
2289	14.12.2014 16:55:27	70.59	44.33	10	175
2290	14.12.2014 16:55:28	62.75	44.98	10	176
2291	14.12.2014 16:55:47	70.83	44.08	10	175
2292	14.12.2014 16:55:48	68.63	44.48	10	176
2293	14.12.2014 16:56:07	72	43.88	10	175
2294	14.12.2014 16:56:08	60	44.08	10	176
2295	14.12.2014 16:56:27	67.35	43.93	10	175
2296	14.12.2014 16:56:28	61.22	44.18	10	176
2297	14.12.2014 16:56:47	71.43	43.83	10	175
2298	14.12.2014 16:56:48	58.82	44.23	10	176
2299	14.12.2014 16:57:07	74	43.93	10	175
2300	14.12.2014 16:57:08	60.78	44.23	10	176
2301	14.12.2014 16:57:27	76	43.93	10	175
2302	14.12.2014 16:57:28	83.67	44.23	10	176
2303	14.12.2014 16:57:47	75	43.88	10	175
2304	14.12.2014 16:57:48	67.35	44.23	10	176
2305	14.12.2014 16:58:07	72.55	43.73	10	175
2306	14.12.2014 16:58:08	62	44.08	10	176
2307	14.12.2014 16:58:27	74	43.73	10	175
2308	14.12.2014 16:58:28	59.18	44.08	10	176

2309	14.12.2014 16:58:47	84	44.03	10	175
2310	14.12.2014 16:58:48	62	44.48	10	176
2311	14.12.2014 16:59:07	60	43.93	10	175
2312	14.12.2014 16:59:08	55.1	44.23	10	176
2313	14.12.2014 16:59:27	69.39	44.18	10	175
2314	14.12.2014 16:59:28	58	44.43	10	176

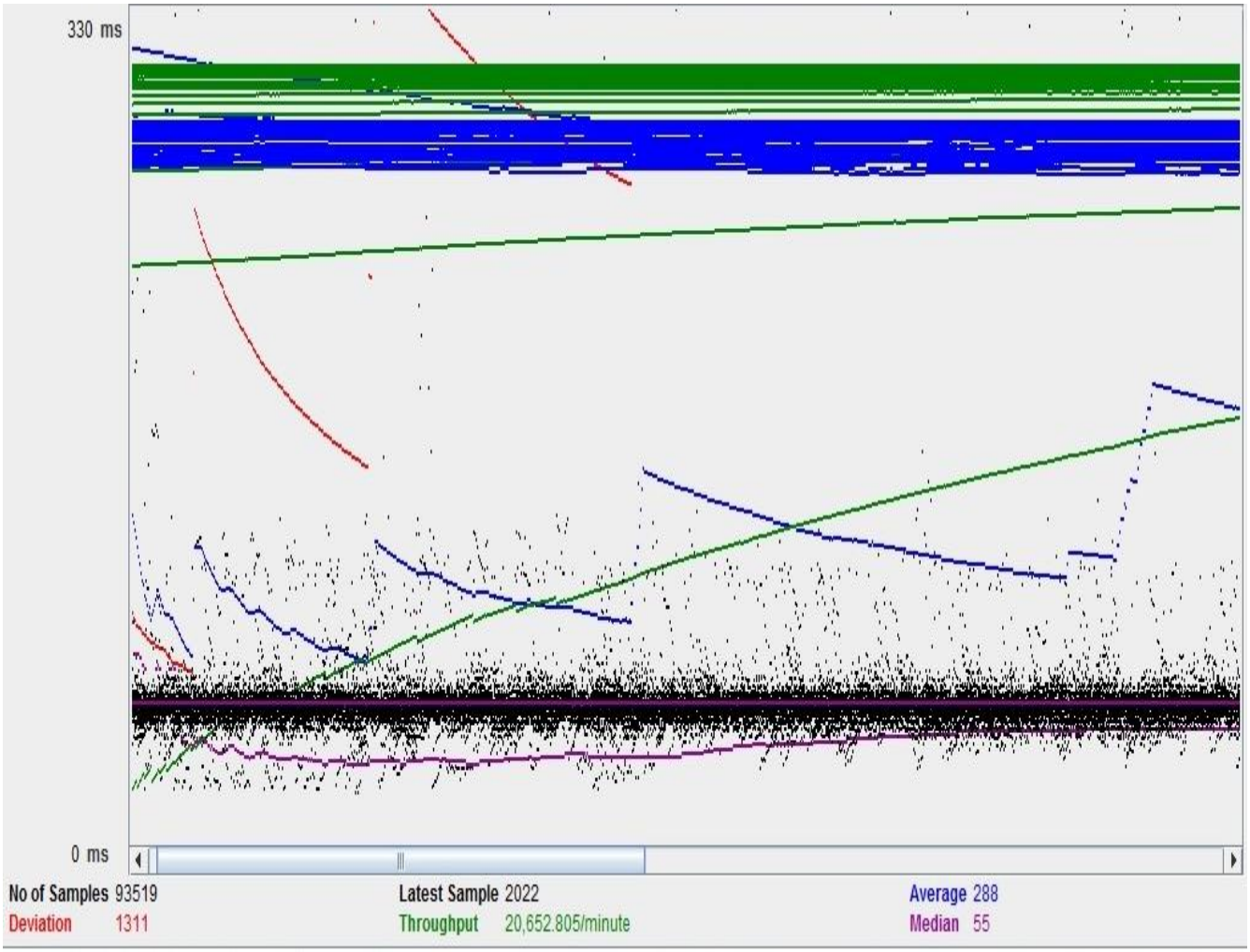


Figure 52 JMeter Graph 2-VMs, 50000 loop, 100 users

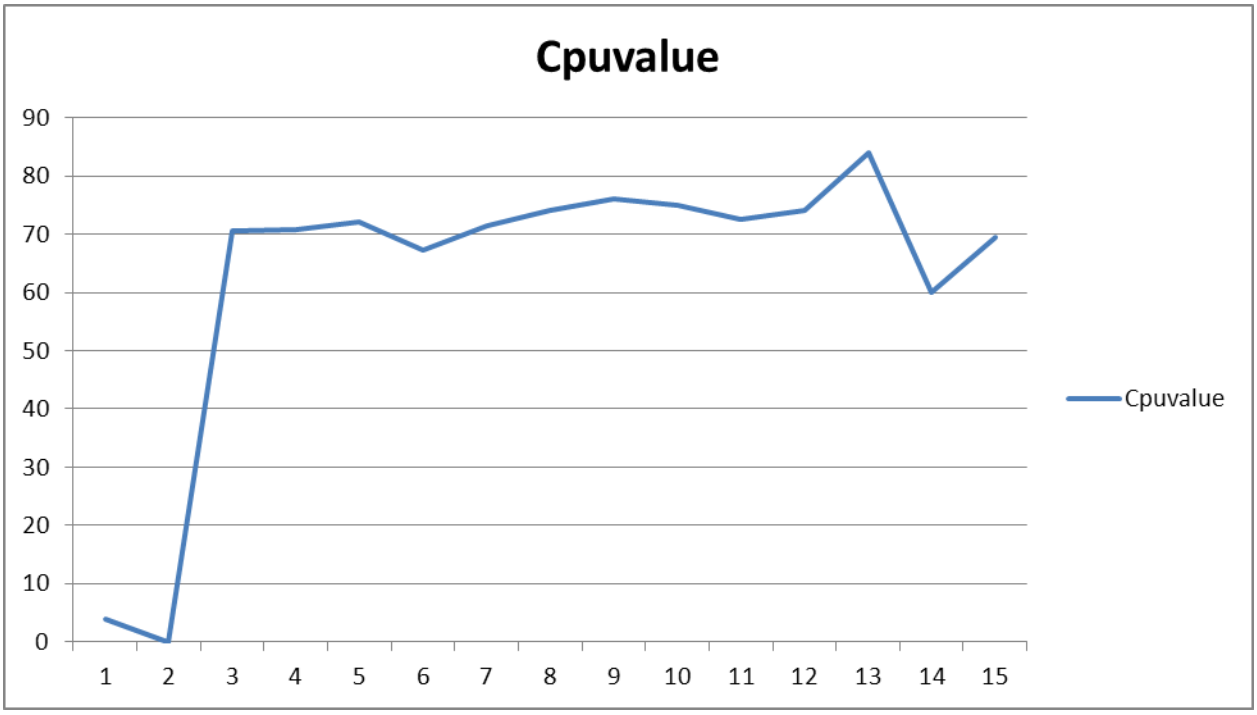


Figure 53 CPU Percentage Instace 1, 2-VMs, 50000 loop, 100 users

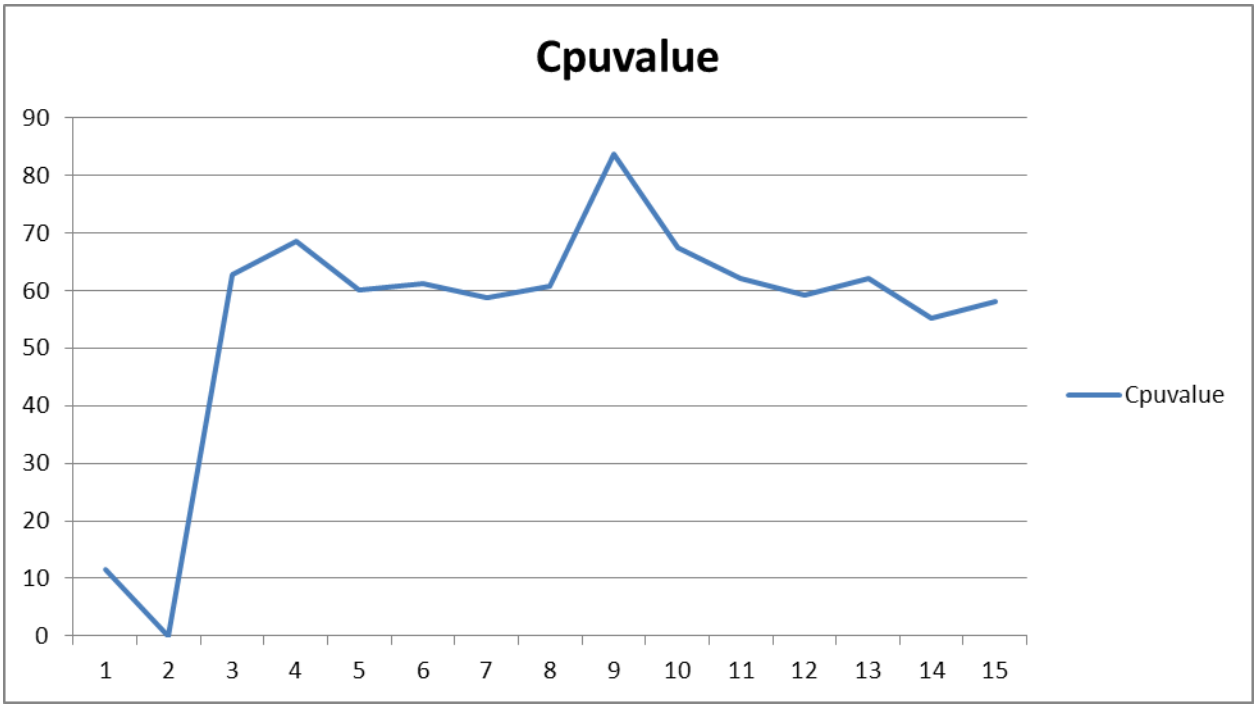


Figure 54 CPU Percentage Instace 2, 2-VMs, 50000 loop, 100 users

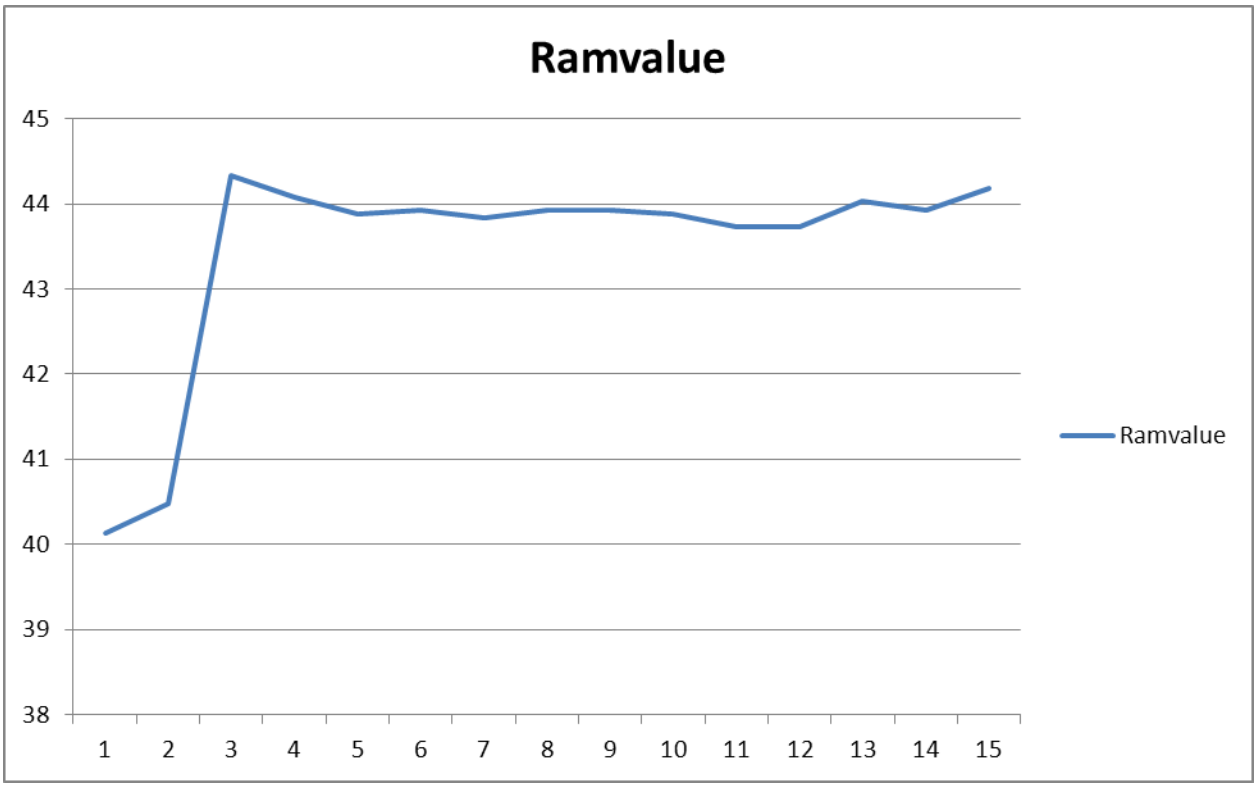


Figure 55 RAM Percentage Instace 1, 2-VMs, 50000 loop, 100 users

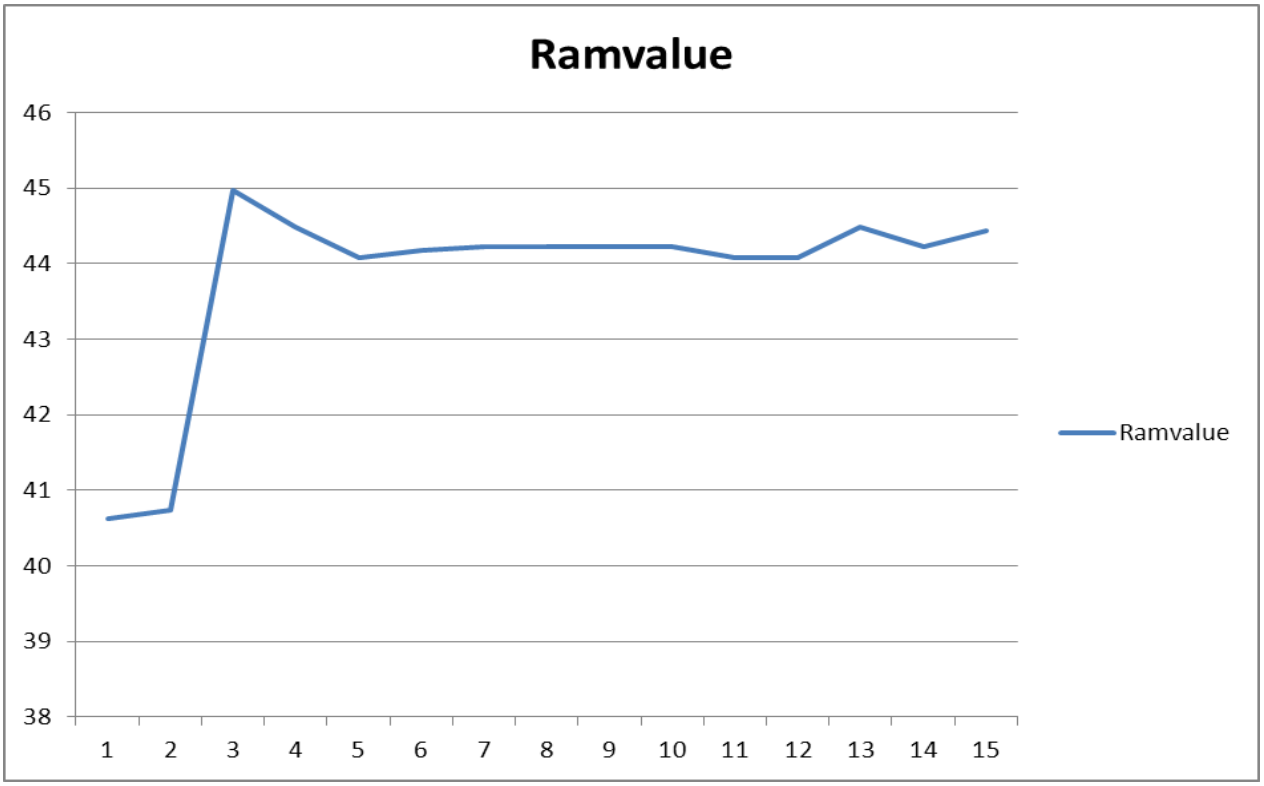


Figure 56 RAM Percentage Instace 2, 2-VMs, 50000 loop, 100 users

Loop=80000

5 users

Table 8 2-VM Instances Monitoring, 80000 loop, 5 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2315	14.12.2014 17:10:47	6	40.18	10	177
2316	14.12.2014 17:10:48	4	41.03	10	178
2317	14.12.2014 17:11:06	0	40.43	10	177
2318	14.12.2014 17:11:07	0	41.28	10	178
2319	14.12.2014 17:11:27	54	41.13	10	177
2320	14.12.2014 17:11:27	47.06	41.98	10	178
2321	14.12.2014 17:11:46	52.94	41.33	10	177
2322	14.12.2014 17:11:47	72.55	42.13	10	178
2323	14.12.2014 17:12:07	46	41.38	10	177

2324	14.12.2014 17:12:07	47.06	42.28	10	178
2325	14.12.2014 17:12:27	54.9	41.43	10	177
2326	14.12.2014 17:12:27	41.18	42.33	10	178
2327	14.12.2014 17:12:46	53.06	41.48	10	177
2328	14.12.2014 17:12:47	44.9	42.38	10	178
2329	14.12.2014 17:13:07	54	41.53	10	177
2330	14.12.2014 17:13:07	42.86	42.38	10	178
2331	14.12.2014 17:13:26	50	41.53	10	177
2332	14.12.2014 17:13:27	42.86	42.38	10	178
2333	14.12.2014 17:13:46	50	41.58	10	177
2334	14.12.2014 17:13:47	50	42.43	10	178
2335	14.12.2014 17:14:07	56	41.63	10	177

2336	14.12.2014 17:14:08	69.39	42.43	10	178
2337	14.12.2014 17:14:26	54	41.63	10	177
2338	14.12.2014 17:14:27	42	42.43	10	178
2339	14.12.2014 17:14:47	52.94	41.68	10	177
2340	14.12.2014 17:14:47	44.9	42.48	10	178
2341	14.12.2014 17:15:06	50	41.73	10	177
2342	14.12.2014 17:15:08	47.06	42.48	10	178
2343	14.12.2014 17:15:26	49.02	41.73	10	177
2344	14.12.2014 17:15:28	49.02	42.48	10	178
2345	14.12.2014 17:15:46	56.86	41.78	10	177
2346	14.12.2014 17:15:47	42	42.48	10	178
2347	14.12.2014 17:16:06	50	41.78	10	177

2348	14.12.2014 17:16:08	44	42.53	10	178

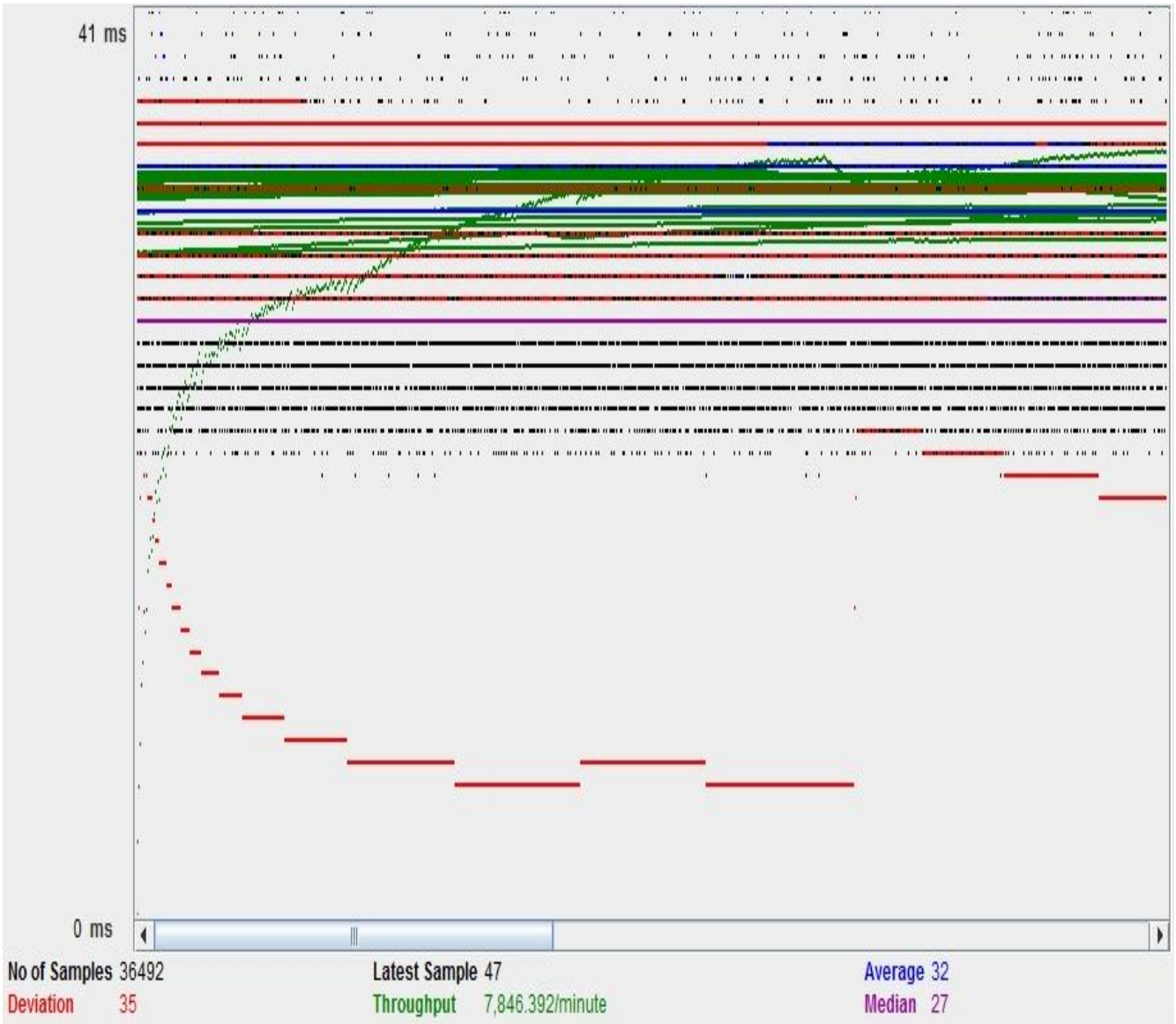


Figure 57 JMeter Graph 2-VMs, 80000 loop, 5 users

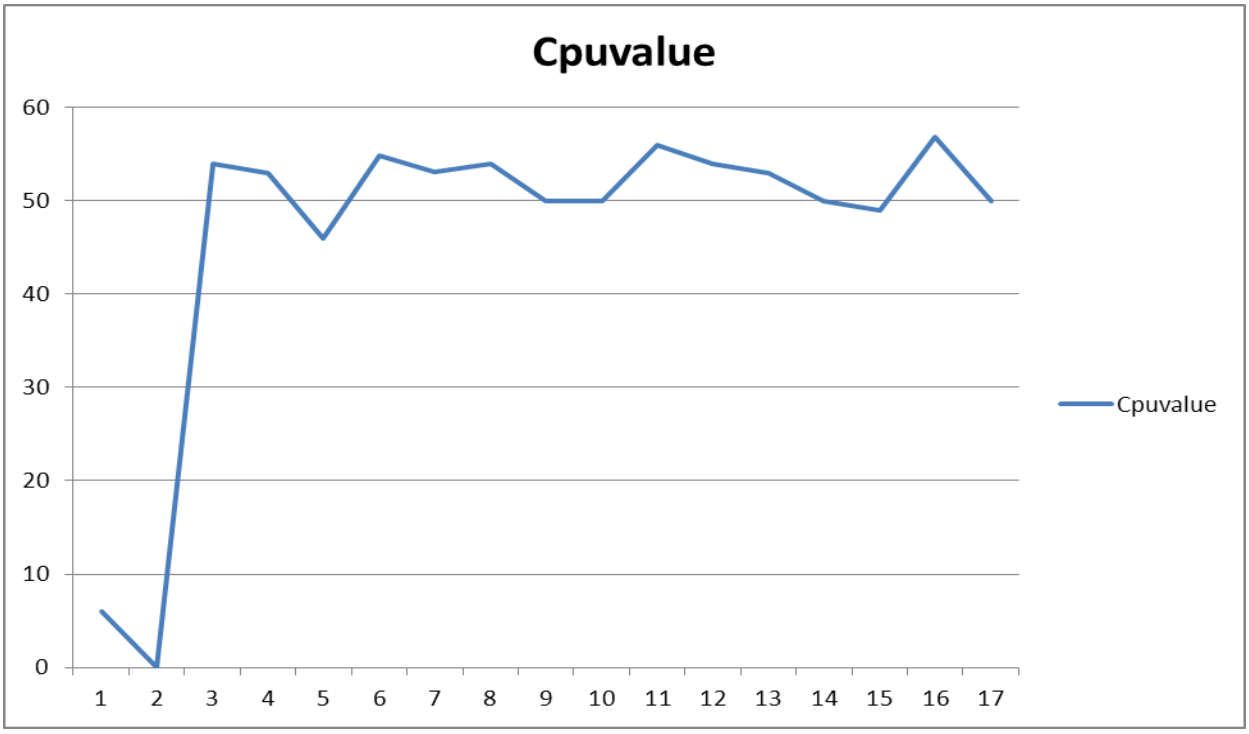


Figure 58 CPU Percentage Instace 1, 2-VMs, 80000 loop, 5 users

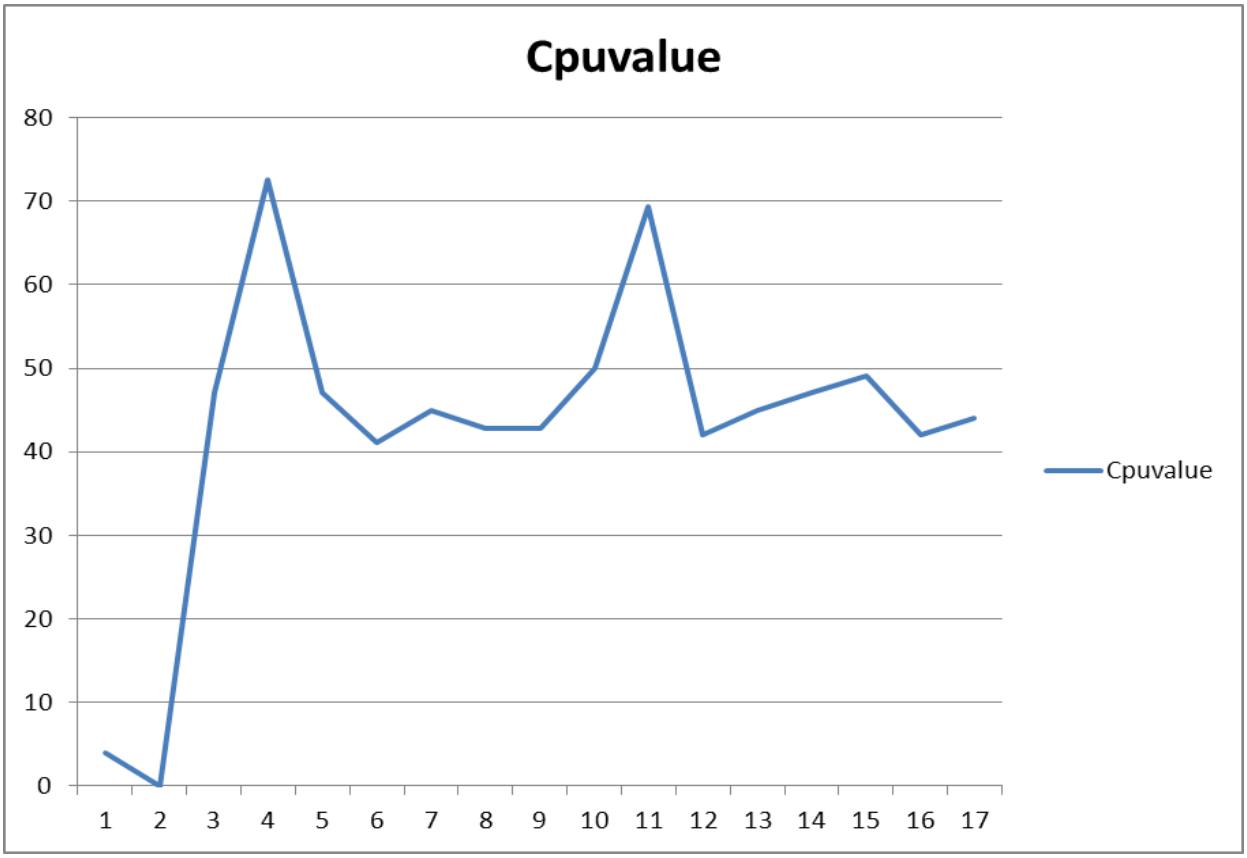


Figure 59 CPU Percentage Instace 2, 2-VMs, 80000 loop, 5 users

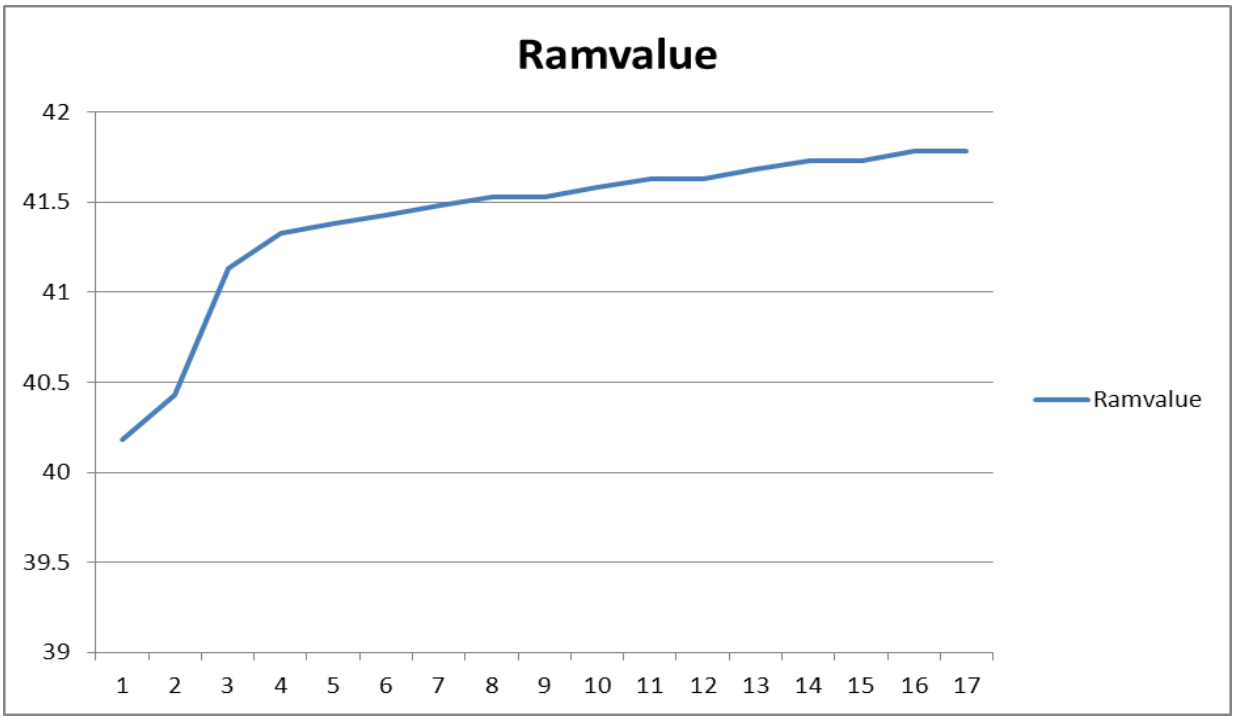


Figure 60 RAM Percentage Instace 1, 2-VMs, 80000 loop, 5 users

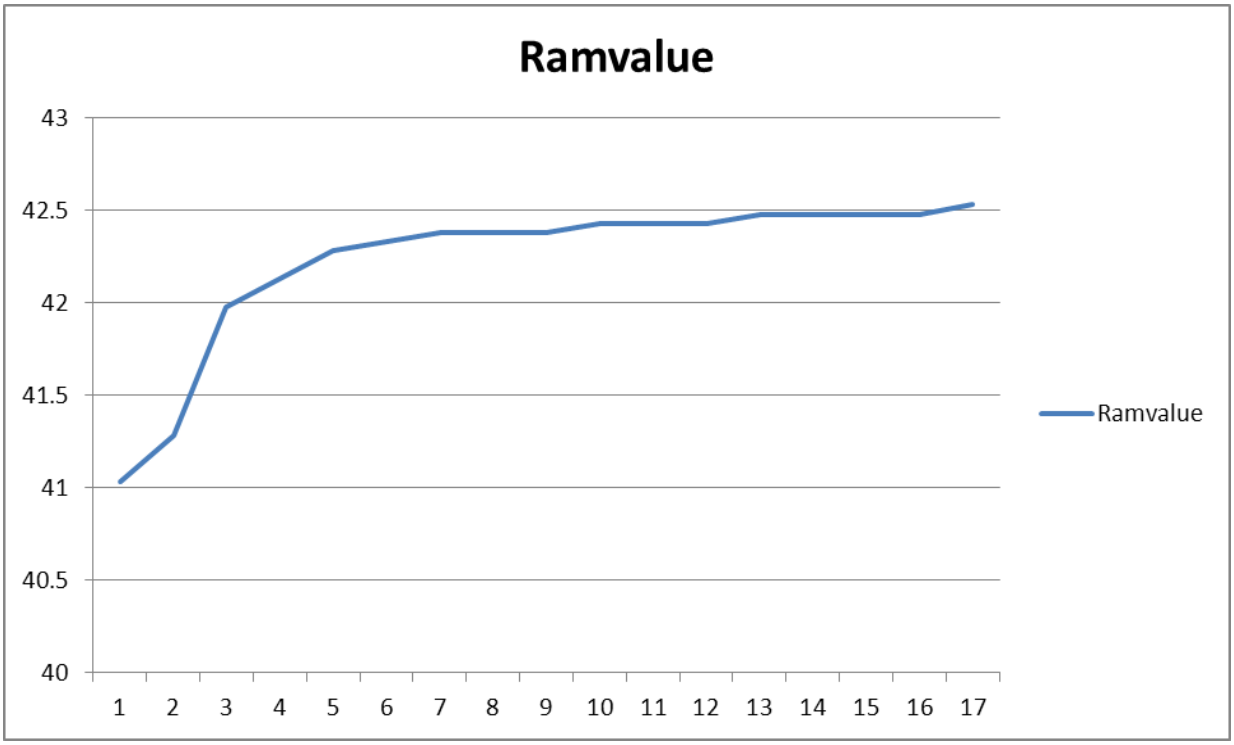


Figure 61 RAM Percentage Instace 2, 2-VMs, 80000 loop, 5 users

20 users

Table 9 2-VM Instances Monitoring, 80000 loop, 20 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2104	13.12.2014 17:05:49	2	41.58	10	156
2105	13.12.2014 17:05:50	1.96	41.78	11	157
2106	13.12.2014 17:06:08	0	41.88	10	156
2107	13.12.2014 17:06:09	4.08	42.03	11	157
2108	13.12.2014 17:06:28	74	43.03	10	156
2109	13.12.2014 17:06:29	90	43.18	11	157
2110	13.12.2014 17:06:48	77.55	43.18	10	156
2111	13.12.2014 17:07:08	73.47	43.18	10	156
2112	13.12.2014 17:07:28	75.51	43.23	10	156
2113	13.12.2014 17:07:48	73.47	43.28	10	156
2114	13.12.2014 17:08:08	78	43.33	10	156
2115	13.12.2014 17:08:28	75.51	43.33	10	156
2116	13.12.2014 17:08:48	76.47	43.38	10	156
2117	13.12.2014 17:09:08	71.43	43.48	10	156
2118	13.12.2014 17:09:28	68.63	43.53	10	156
2119	13.12.2014 17:09:48	72	43.53	10	156
2120	13.12.2014 17:10:08	78	43.53	10	156
2121	13.12.2014 17:10:28	73.08	43.58	10	156

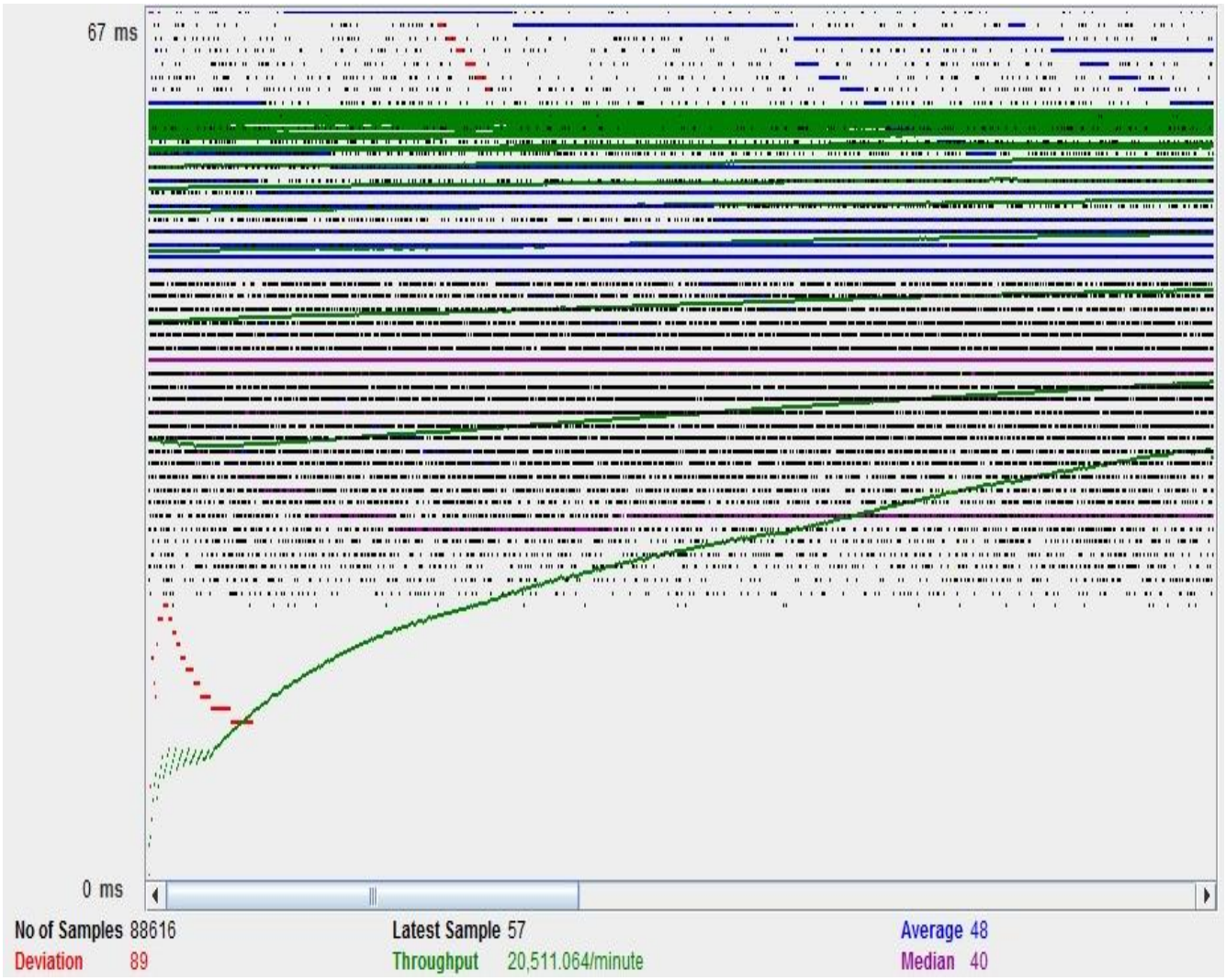


Figure 62 JMeter Graph 2-VMs, 80000 loop, 20 users

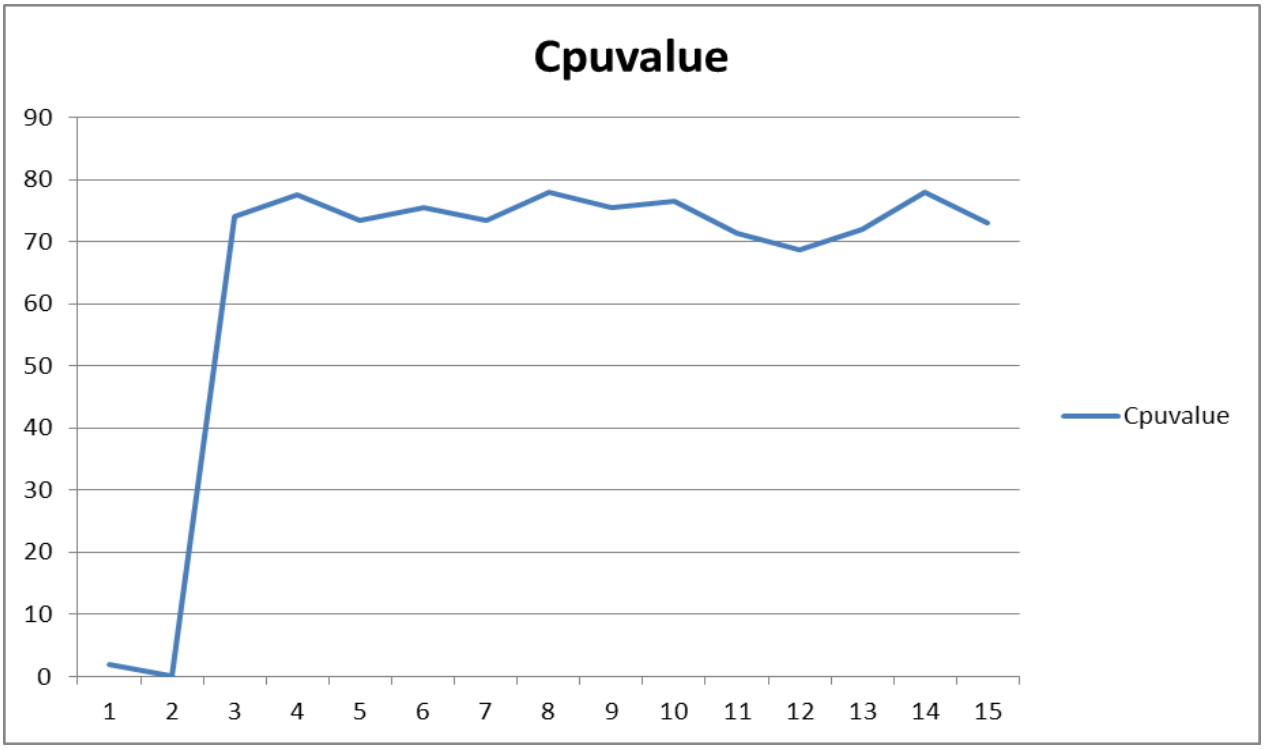


Figure 63 CPU Percentage Instace 1, 2-VMs, 80000 loop, 20 users

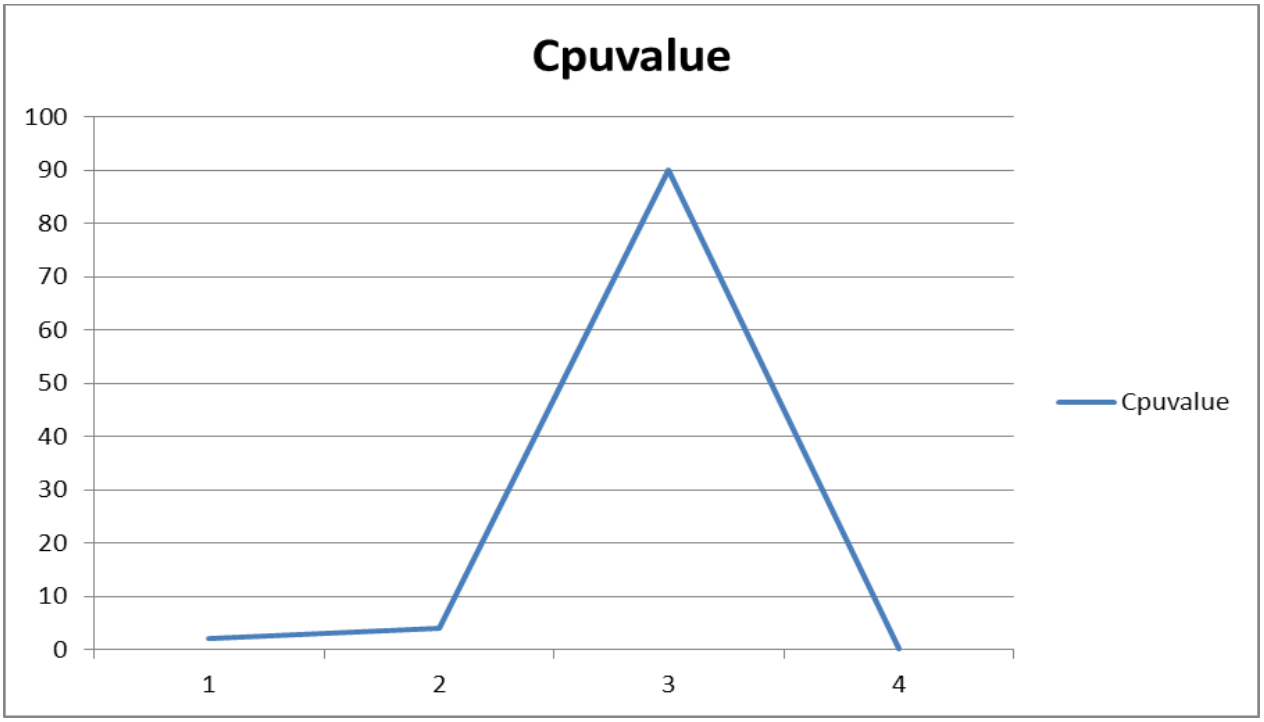


Figure 64 CPU Percentage Instace 2, 2-VMs, 80000 loop, 20 users

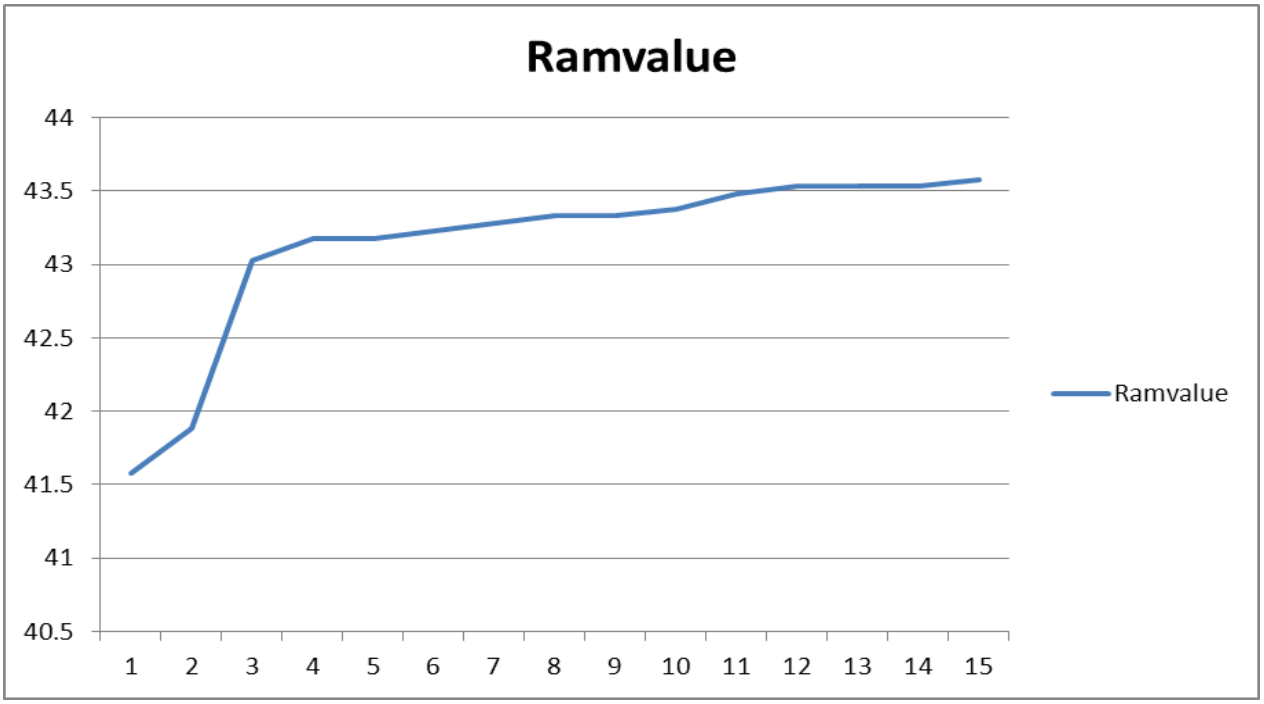


Figure 65 RAM Percentage Instace 1, 2-VMs, 80000 loop, 20 users

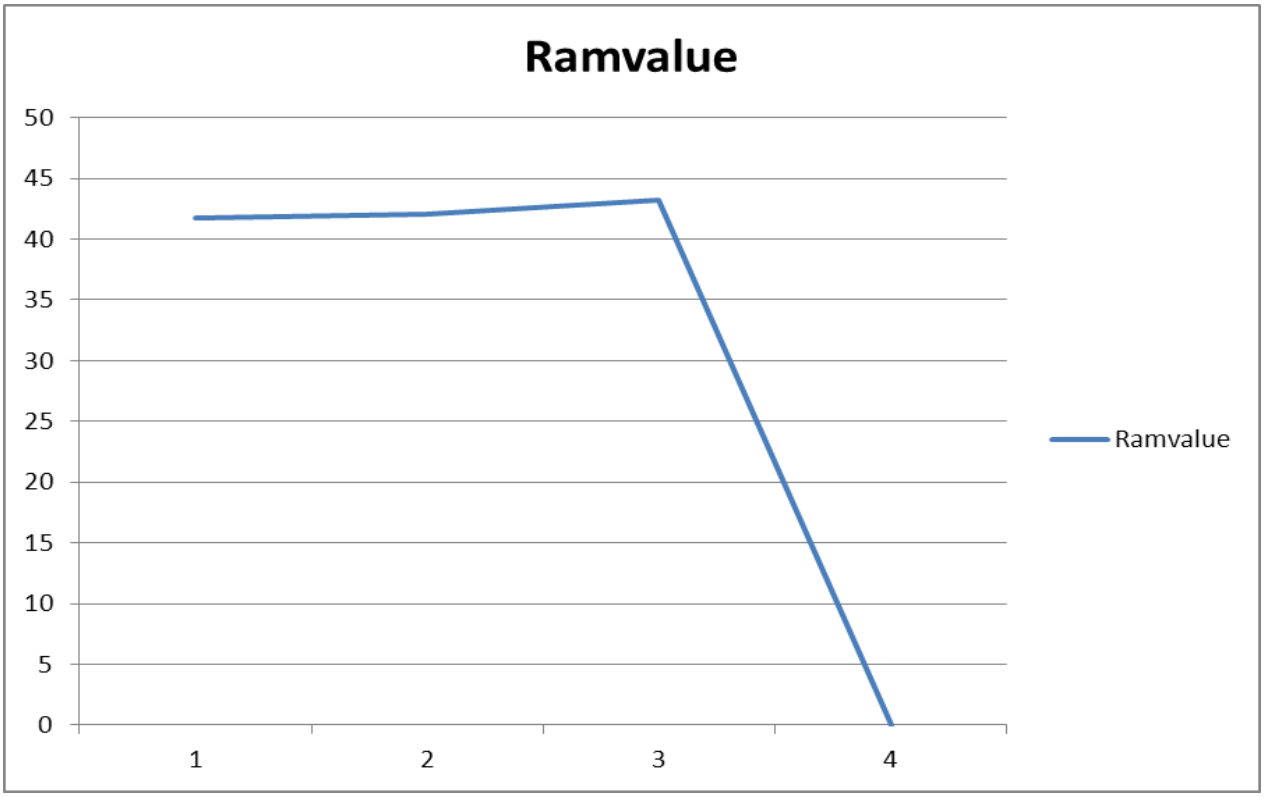


Figure 66 RAM Percentage Instace 2, 2-VMs, 80000 loop, 20 users

Virtual Machines = 4

Τα στοιχεία των εικονικών μηχανημάτων φαίνονται στον παρακάτω πίνακα.

Table 10 4-VM Instances

id	IP	CpuCores	AvailableRam	AvailableDisk	State	VMname
179	10.0.1.6	1	2001	19	DOWN	mon-inst-1
180	10.0.1.7	1	2001	19	DOWN	mon-inst-2
181	10.0.1.4	1	2001	19	DOWN	mon-inst-3
182	10.0.1.9	1	2001	19	DOWN	mon-inst-4

Τα ανώτατα όρια/thresholds που ακολουθούν τις πολιτικές monitoring που έχουμε εφαρμόσει φαίνονται στον παρακάτω πίνακα υπό την μορφή (ποσοστό/100).

Table 11 4-VM Instances Policy

id	CPUthres	RAMthres	VM
833	0.9	0.9	179
834	0.9	0.9	180
835	0.9	0.9	181
836	0.9	0.9	182

Loop = 80000

100 users

Table 12 4-VM Instances Monitoring, 80000 loop, 100 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2469	14.12.2014 20:08:04	2	41.58	10	187
2470	14.12.2014 20:08:05	6	41.58	10	188
2471	14.12.2014 20:08:06	2	41.83	10	189
2472	14.12.2014 20:08:07	3.92	41.63	10	190
2473	14.12.2014 20:08:23	0	41.78	10	187
2474	14.12.2014 20:08:24	3.85	41.78	10	188
2475	14.12.2014 20:08:25	0	42.03	10	189
2476	14.12.2014 20:08:25	0	41.78	10	190
2477	14.12.2014 20:08:43	51.92	44.93	10	187
2478	14.12.2014 20:08:44	57.14	44.88	10	188
2479	14.12.2014 20:08:45	53.06	45.08	10	189
2480	14.12.2014 20:08:45	60	44.78	10	190
2481	14.12.2014 20:09:03	54	45.08	10	187
2482	14.12.2014 20:09:04	62	45.03	10	188
2483	14.12.2014 20:09:05	56.86	45.18	10	189
2484	14.12.2014 20:09:05	46.94	44.88	10	190
2485	14.12.2014 20:09:23	48.98	45.33	10	187
2486	14.12.2014 20:09:24	72	45.18	10	188
2487	14.12.2014 20:09:24	58	45.33	10	189
2488	14.12.2014 20:09:25	50	45.13	10	190
2489	14.12.2014 20:09:43	52	45.03	10	187
2490	14.12.2014 20:09:44	67.35	44.98	10	188

2491	14.12.2014 20:09:45	53.06	45.13	10	189
2492	14.12.2014 20:09:46	54	44.88	10	190
2493	14.12.2014 20:10:03	54	44.88	10	187
2494	14.12.2014 20:10:04	71.15	44.83	10	188
2495	14.12.2014 20:10:04	59.18	45.08	10	189
2496	14.12.2014 20:10:05	57.14	44.83	10	190
2497	14.12.2014 20:10:23	54.9	44.88	10	187
2498	14.12.2014 20:10:24	65.31	44.93	10	188
2499	14.12.2014 20:10:24	60	45.03	10	189
2500	14.12.2014 20:10:25	50	44.78	10	190
2501	14.12.2014 20:10:43	52	45.03	10	187
2502	14.12.2014 20:10:44	62.75	44.98	10	188
2503	14.12.2014 20:10:45	54.9	45.18	10	189
2504	14.12.2014 20:10:45	54	44.98	10	190
2505	14.12.2014 20:11:03	52.94	44.98	10	187
2506	14.12.2014 20:11:04	63.27	44.93	10	188
2507	14.12.2014 20:11:04	57.14	45.08	10	189
2508	14.12.2014 20:11:05	52.94	44.83	10	190
2509	14.12.2014 20:11:23	53.06	45.03	10	187
2510	14.12.2014 20:11:24	65.38	45.03	10	188
2511	14.12.2014 20:11:24	57.14	45.18	10	189
2512	14.12.2014 20:11:25	54.17	45.03	10	190
2513	14.12.2014 20:11:43	54	45.08	10	187
2514	14.12.2014 20:11:44	66	45.03	10	188
2515	14.12.2014 20:11:44	59.18	45.18	10	189
2516	14.12.2014 20:11:45	56	44.93	10	190
2517	14.12.2014 20:12:03	56	44.93	10	187
2518	14.12.2014 20:12:04	62	44.93	10	188
2519	14.12.2014 20:12:05	54	44.98	10	189

2520	14.12.2014 20:12:05	52	44.93	10	190
2521	14.12.2014 20:12:23	53.06	45.03	10	187
2522	14.12.2014 20:12:24	66.67	45.03	10	188
2523	14.12.2014 20:12:24	60	45.13	10	189
2524	14.12.2014 20:12:25	52.94	44.93	10	190
2525	14.12.2014 20:12:43	54	45.08	10	187
2526	14.12.2014 20:12:44	65.31	45.08	10	188
2527	14.12.2014 20:12:45	56.86	45.18	10	189
2528	14.12.2014 20:12:45	54	44.98	10	190

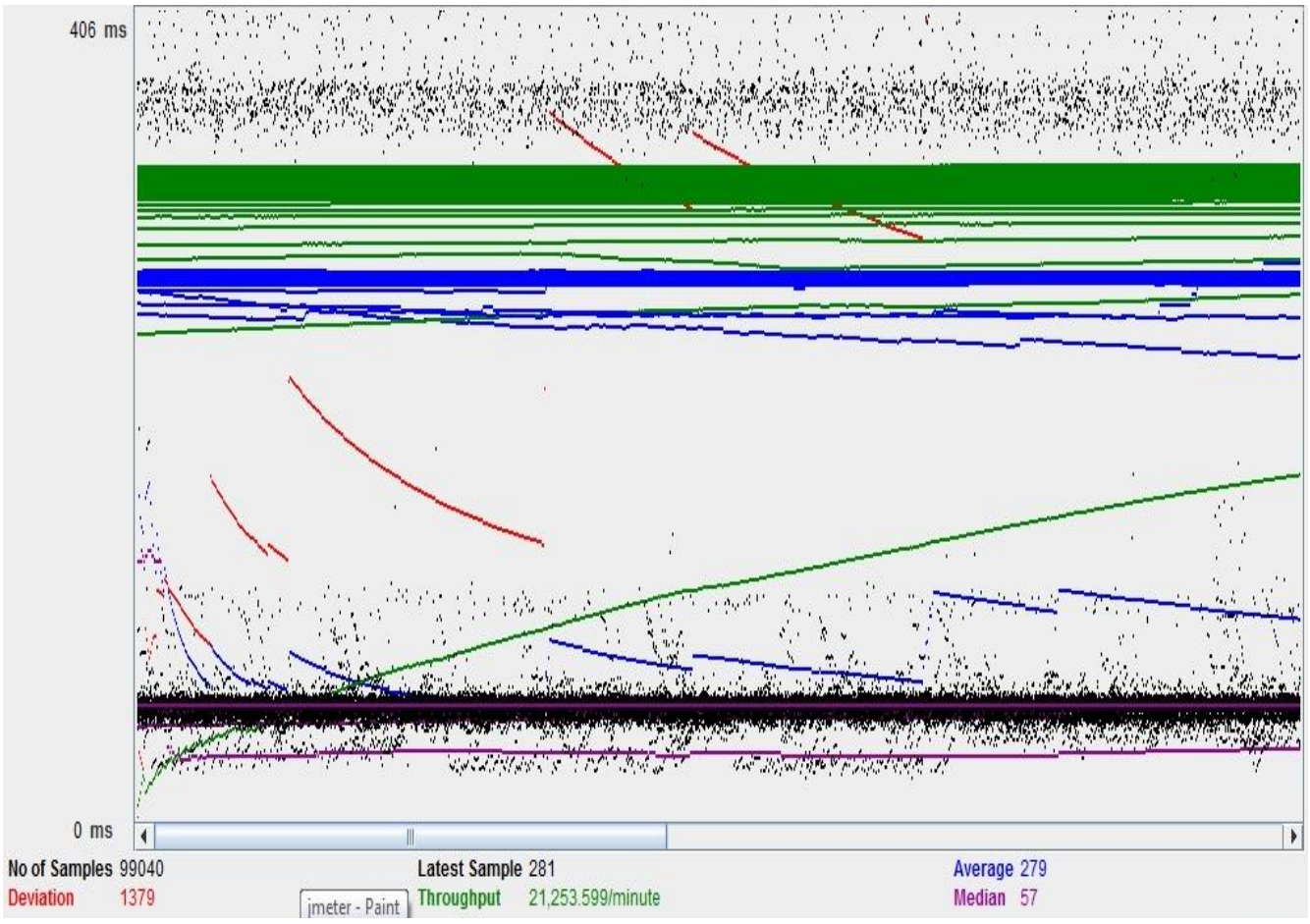


Figure 67 JMeter Graph 4-VMs, 80000 loop, 100 users

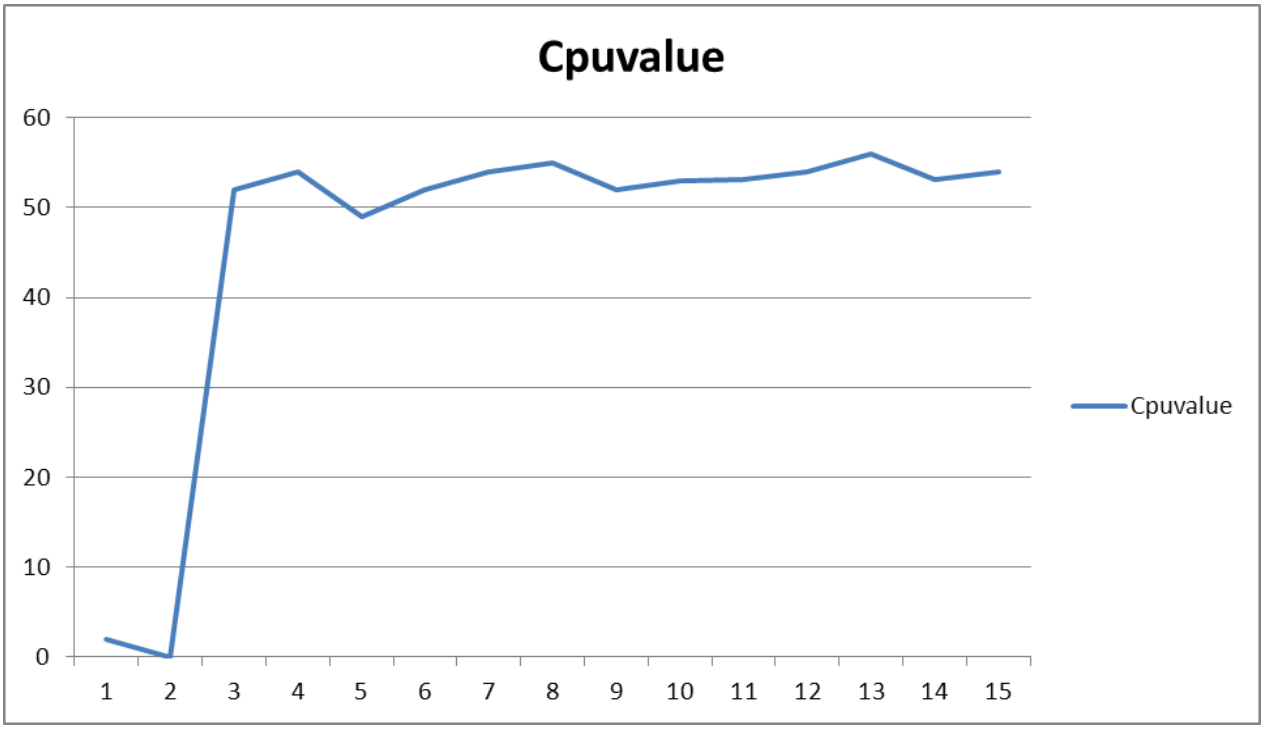


Figure 68 CPU Percentage Instace 1, 4-VMs, 80000 loop, 100 users

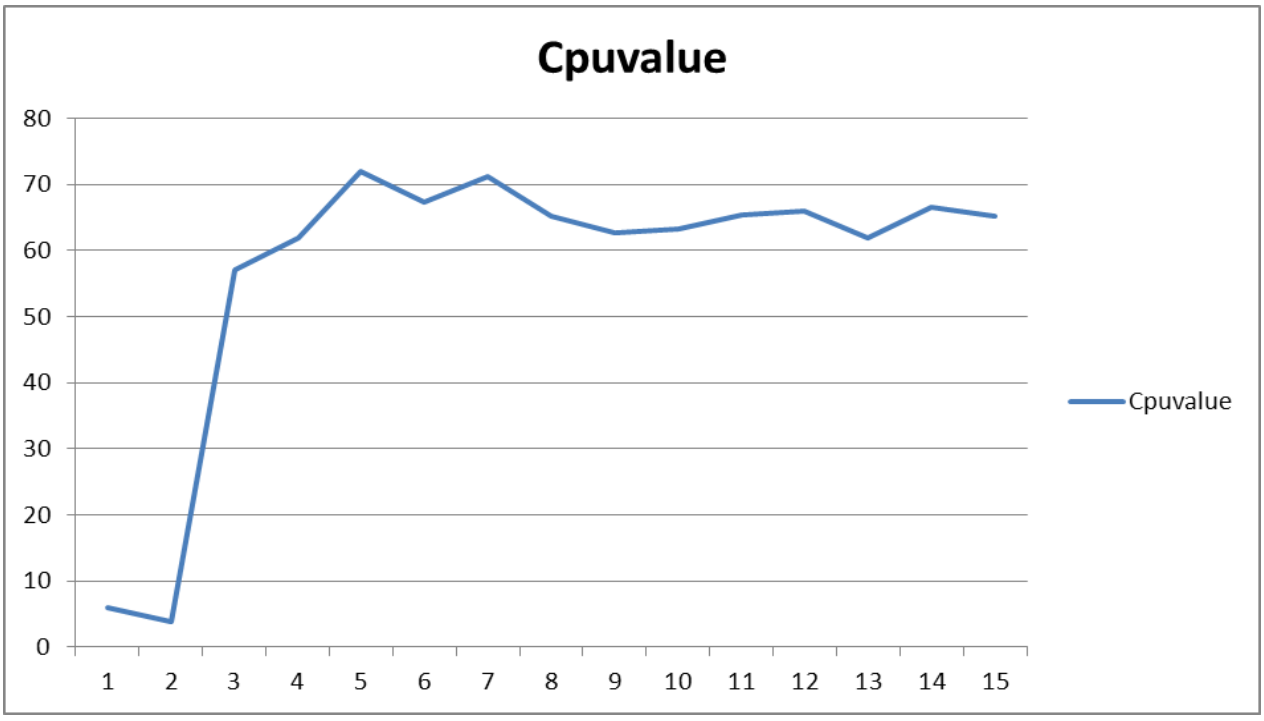


Figure 69 CPU Percentage Instace 2, 4-VMs, 80000 loop, 100 users

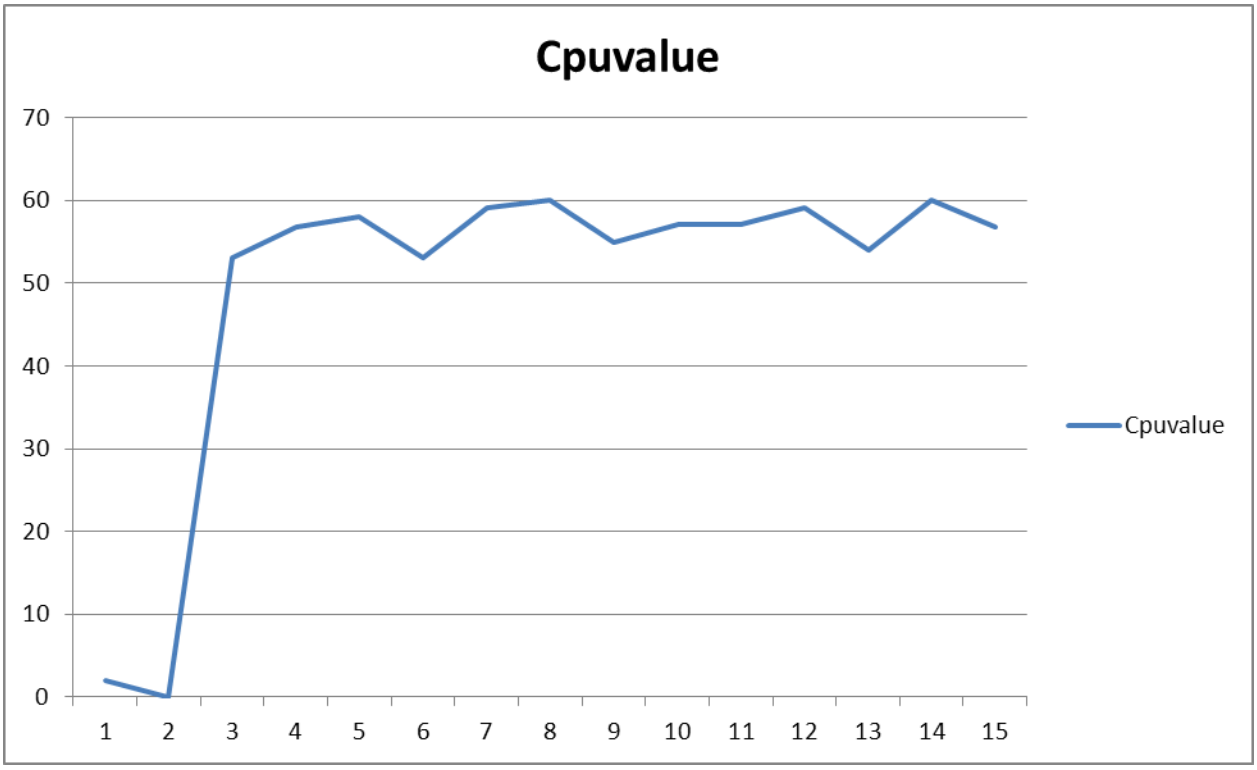


Figure 70 CPU Percentage Instace 3, 4-VMs, 80000 loop, 100 users

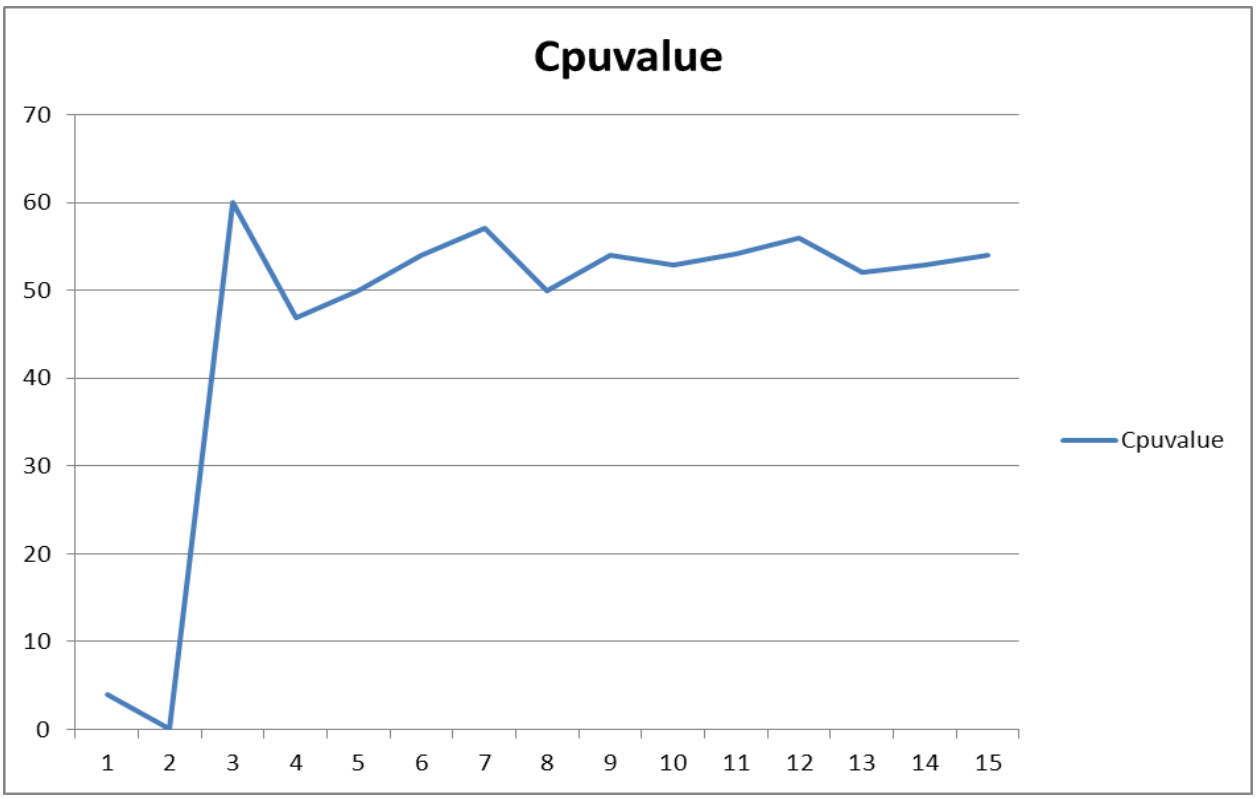


Figure 71 CPU Percentage Instace 4, 4-VMs, 80000 loop, 100 users

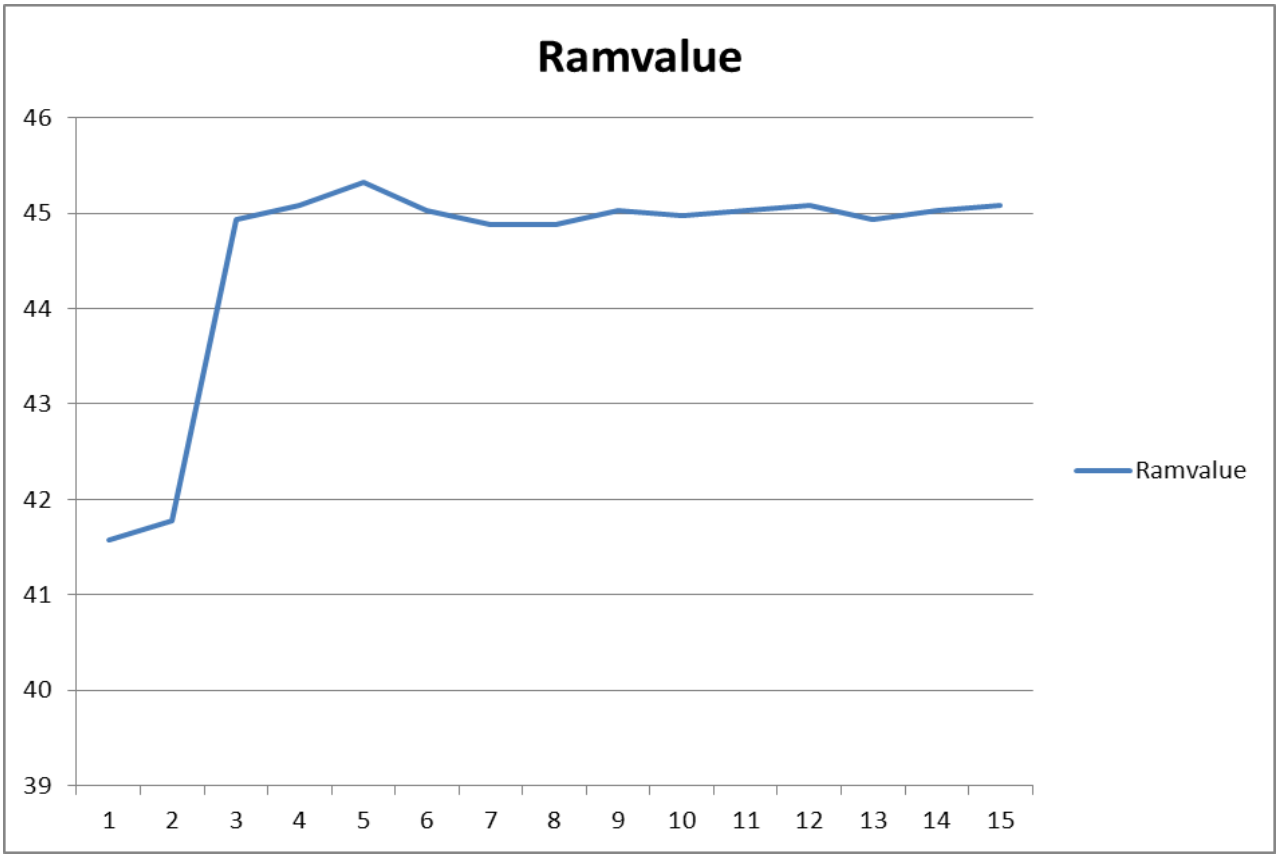


Figure 72 RAM Percentage Instace 1, 4-VMs, 80000 loop, 100 users

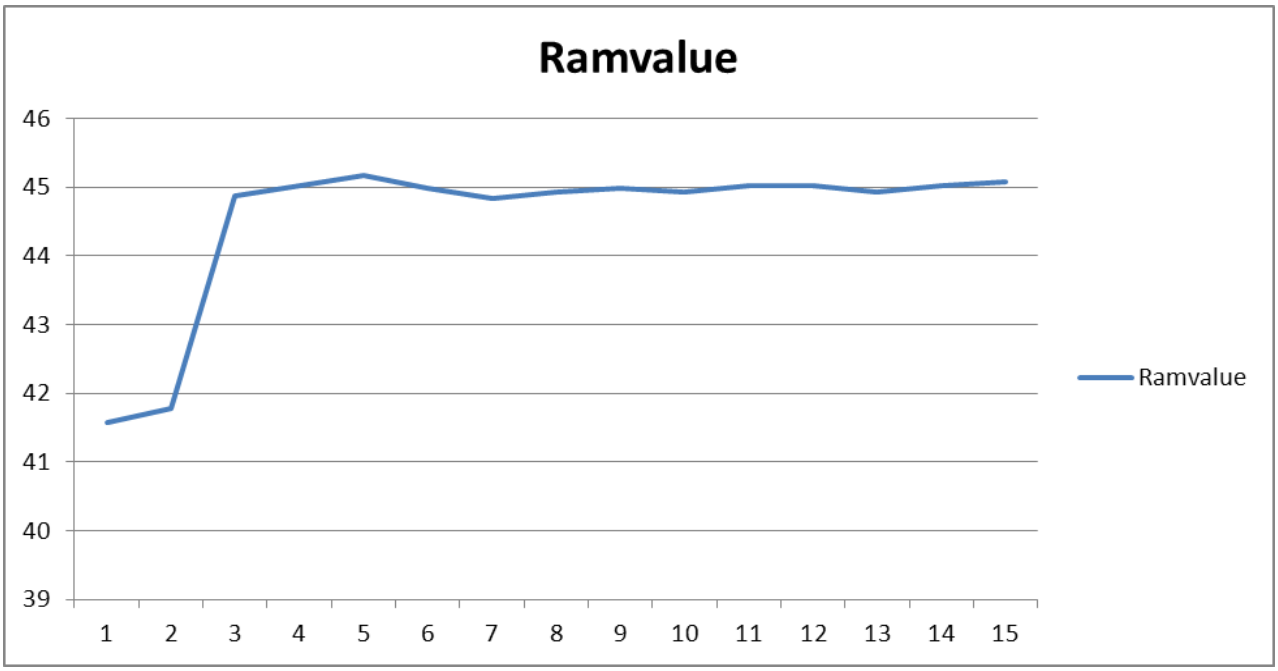


Figure 73 RAM Percentage Instace 2, 4-VMs, 80000 loop, 100 users

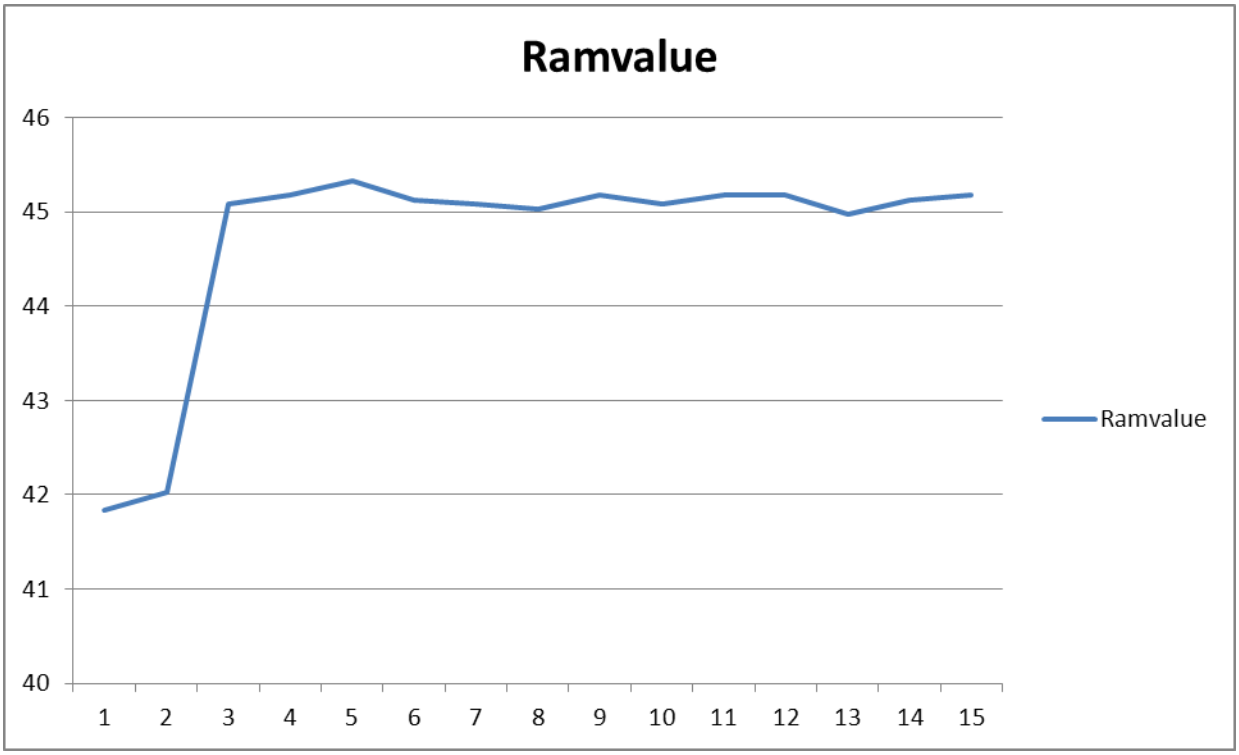


Figure 74 RAM Percentage Instace 3, 4-VMs, 80000 loop, 100 users

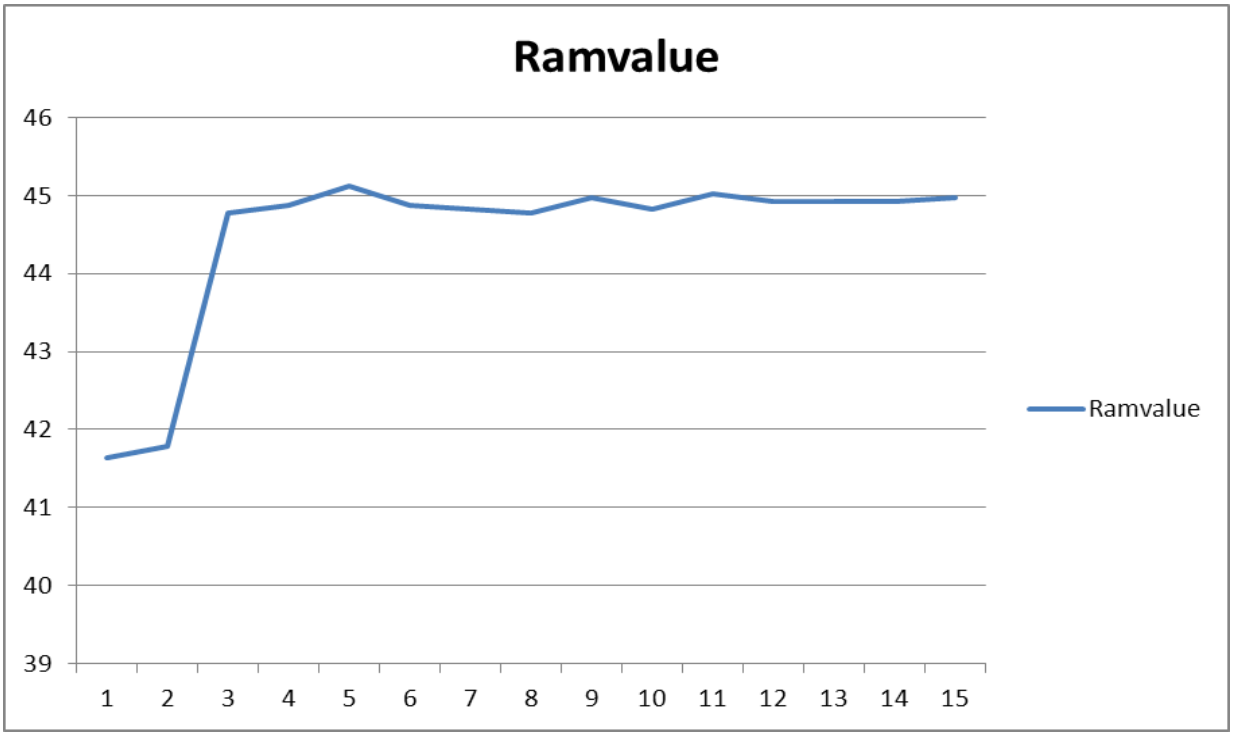


Figure 75 RAM Percentage Instace 4, 4-VMs, 80000 loop, 100 users

Loop = 160000

100 users

Table 13 4-VM Instances Monitoring, 160000 loop, 100 users

id	timestamp	Cpuvalue	Ramvalue	Diskvalue	VM
2553	14.12.2014 21:01:26	2	41.48	10	195
2554	14.12.2014 21:01:27	0	41.33	10	196
2555	14.12.2014 21:01:28	0	41.83	10	197
2556	14.12.2014 21:01:29	2	41.53	10	198
2557	14.12.2014 21:01:46	0	41.73	10	195
2558	14.12.2014 21:01:47	2	41.53	10	196
2559	14.12.2014 21:01:47	0	42.08	10	197
2560	14.12.2014 21:01:48	1.96	41.73	10	198
2561	14.12.2014 21:02:06	66	45.83	10	195
2562	14.12.2014 21:02:06	88.24	45.18	10	196
2563	14.12.2014 21:02:07	79.59	46.18	10	197
2564	14.12.2014 21:02:08	79.59	45.83	10	198
2565	14.12.2014 21:02:26	68	45.53	10	195
2566	14.12.2014 21:02:26	92	45.18	10	196
2567	14.12.2014 21:02:27	80	45.73	10	197
2568	14.12.2014 21:02:28	81.63	45.48	10	198
2569	14.12.2014 21:02:46	66	45.28	10	195
2570	14.12.2014 21:02:47	80.39	45.63	10	197
2571	14.12.2014 21:02:47	76	45.28	10	198

2572	14.12.2014 21:03:06	62.75	44.98	10	195
2573	14.12.2014 21:03:07	79.59	45.38	10	197
2574	14.12.2014 21:03:08	75.51	45.03	10	198
2575	14.12.2014 21:03:26	65.31	45.08	10	195
2576	14.12.2014 21:03:27	75.51	45.38	10	197
2577	14.12.2014 21:03:27	68.63	45.13	10	198
2578	14.12.2014 21:03:46	64.71	45.03	10	195
2579	14.12.2014 21:03:47	72	45.33	10	197
2580	14.12.2014 21:03:47	78	45.08	10	198
2581	14.12.2014 21:04:06	69.39	44.98	10	195
2582	14.12.2014 21:04:06	78	45.33	10	197
2583	14.12.2014 21:04:07	77.55	45.03	10	198
2584	14.12.2014 21:04:26	68.63	45.13	10	195
2585	14.12.2014 21:04:26	76	45.48	10	197
2586	14.12.2014 21:04:27	80	45.13	10	198
2587	14.12.2014 21:04:46	78.43	45.03	10	195
2588	14.12.2014 21:04:47	82	45.33	10	197
2589	14.12.2014 21:04:48	82	44.98	10	198
2590	14.12.2014 21:05:06	67.35	45.08	10	195
2591	14.12.2014 21:05:06	65.31	45.33	10	197
2592	14.12.2014 21:05:07	80	45.08	10	198
2593	14.12.2014 21:05:26	66	45.08	10	195
2594	14.12.2014 21:05:26	80	45.43	10	197
2595	14.12.2014 21:05:27	79.59	45.08	10	198

2596	14.12.2014 21:05:46	66	45.08	10	195
2597	14.12.2014 21:05:47	80	45.43	10	197
2598	14.12.2014 21:05:47	78	45.08	10	198

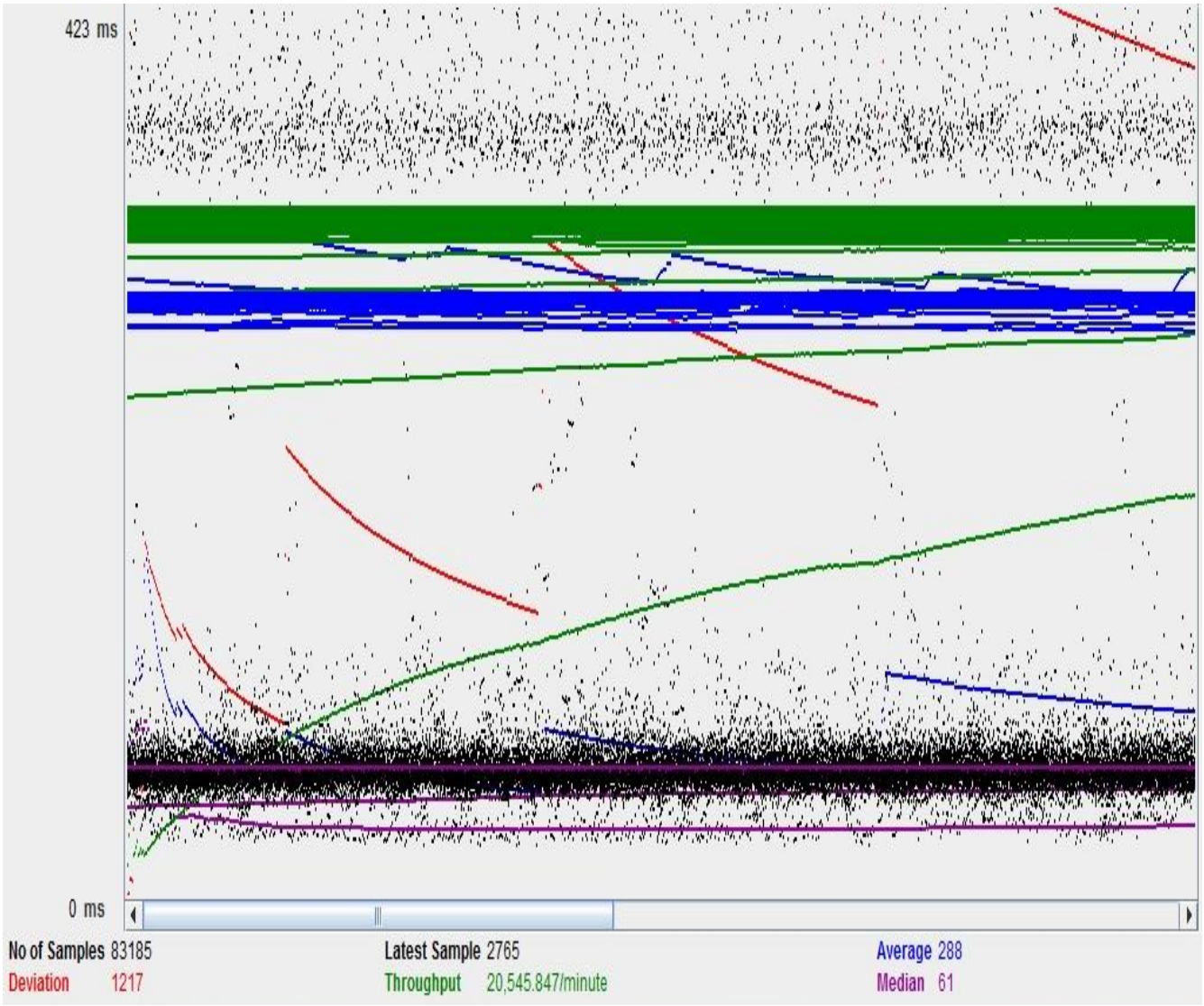


Figure 76 JMeter Graph 4-VMs, 160000 loop, 100 users

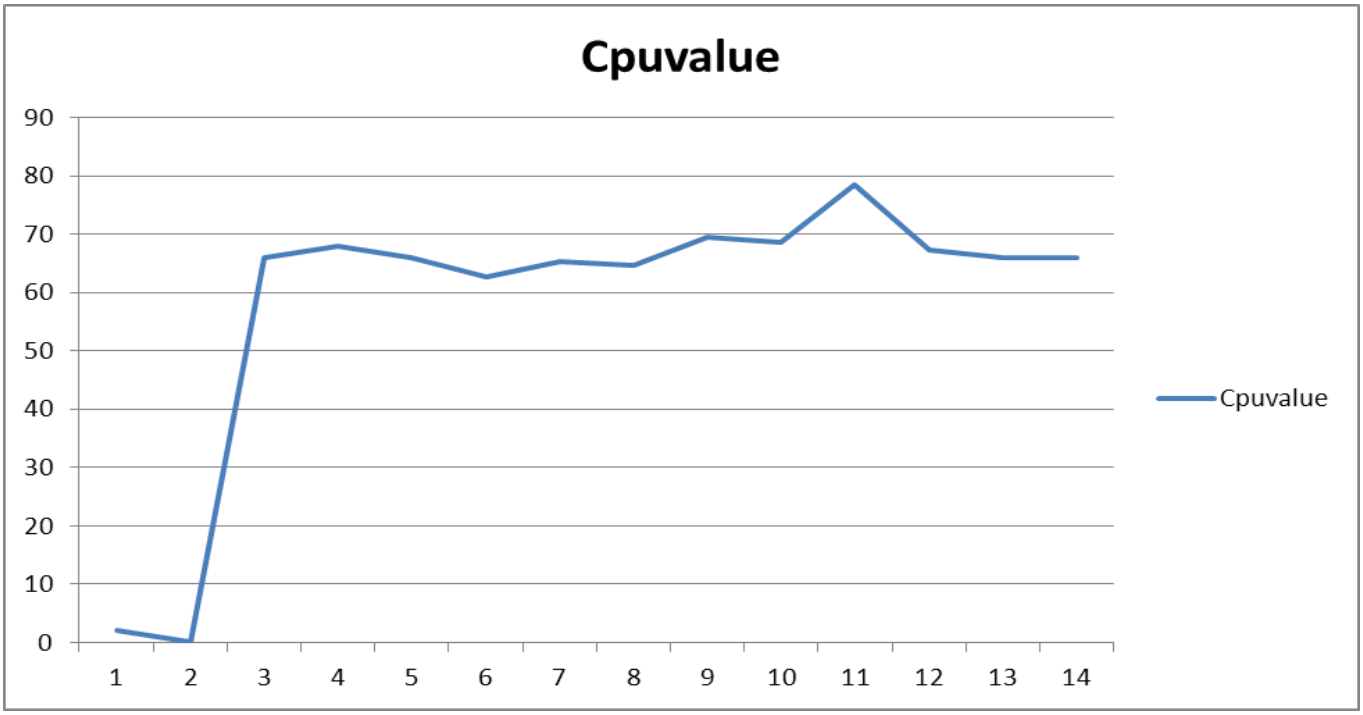


Figure 77 CPU Percentage Instace 1, 4-VMs, 160000 loop, 100 users

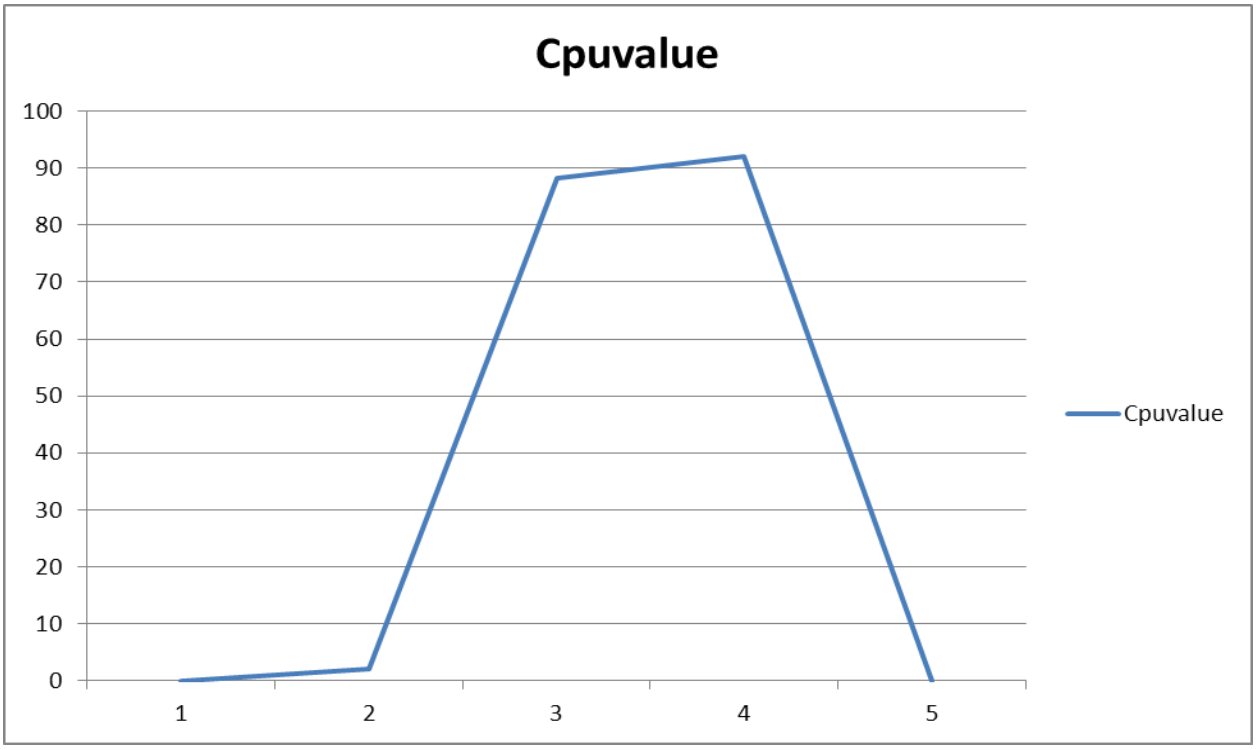


Figure 78 CPU Percentage Instace 2, 4-VMs, 160000 loop, 100 users

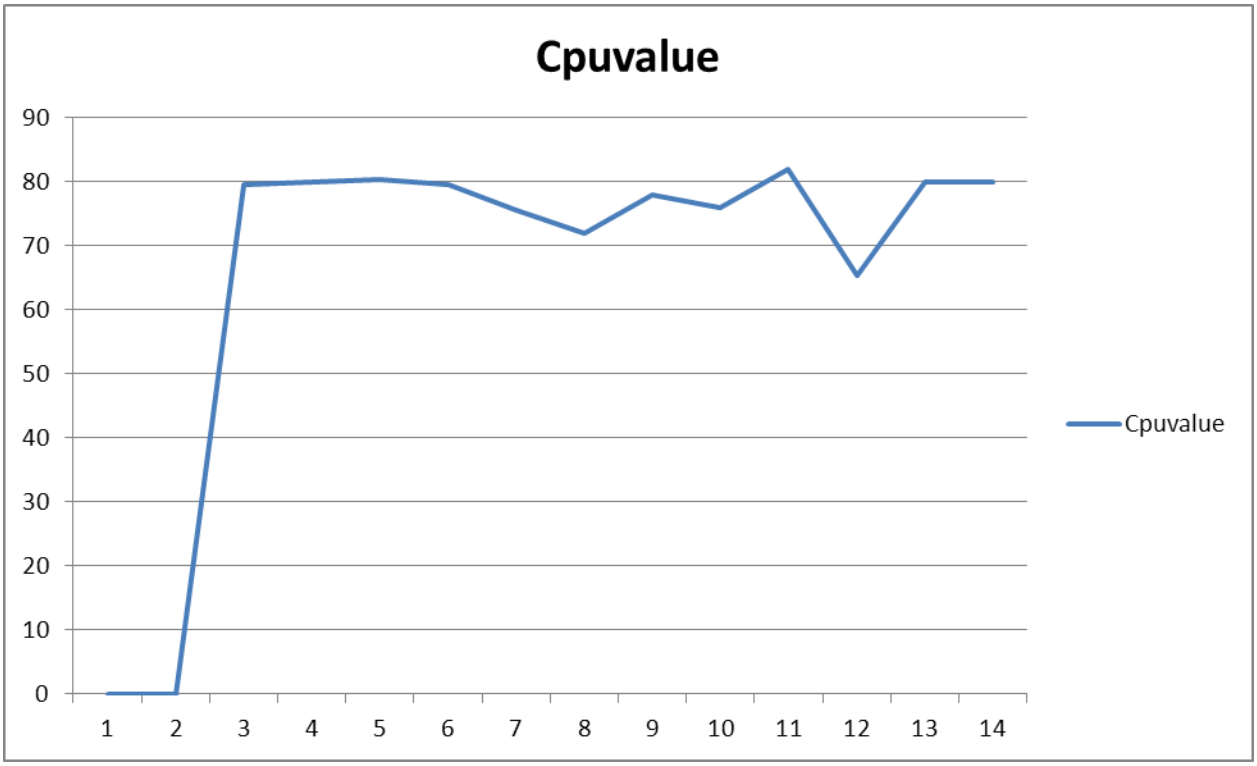


Figure 79 CPU Percentage Instace 3, 4-VMs, 160000 loop, 100 users

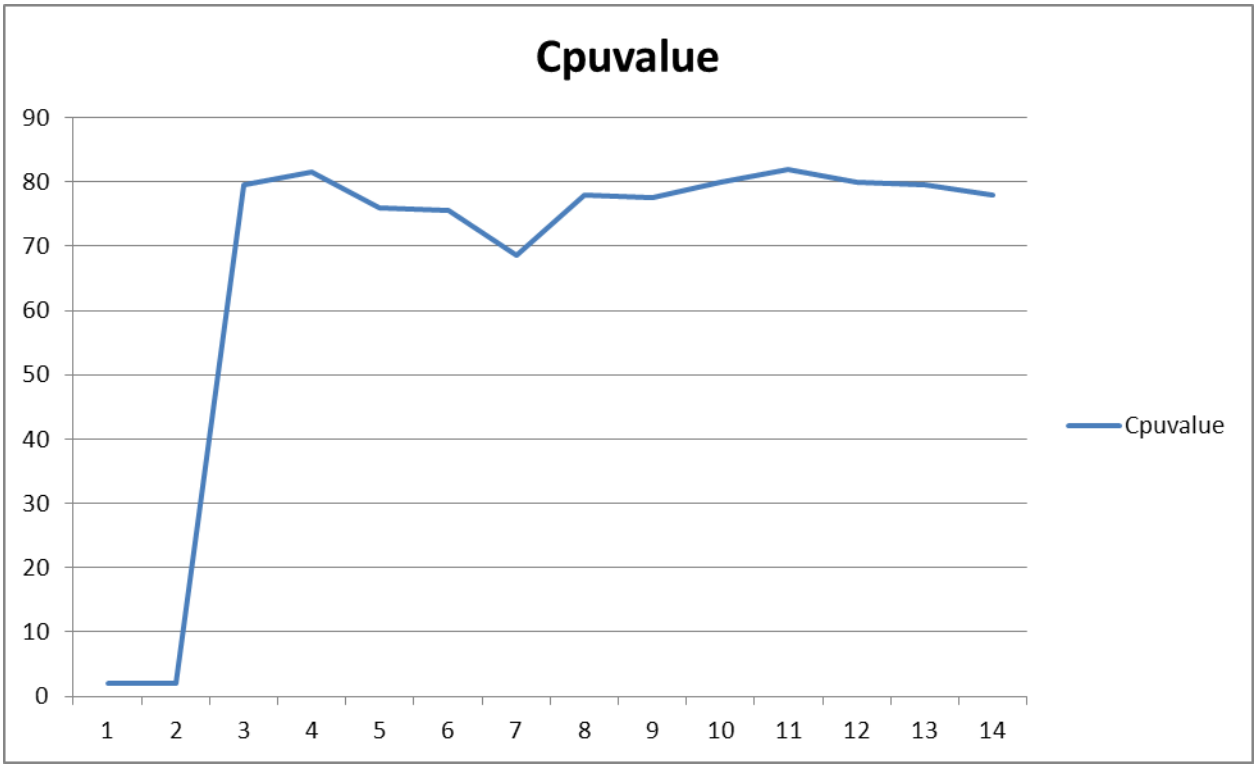


Figure 80 CPU Percentage Instace 4, 4-VMs, 160000 loop, 100 users

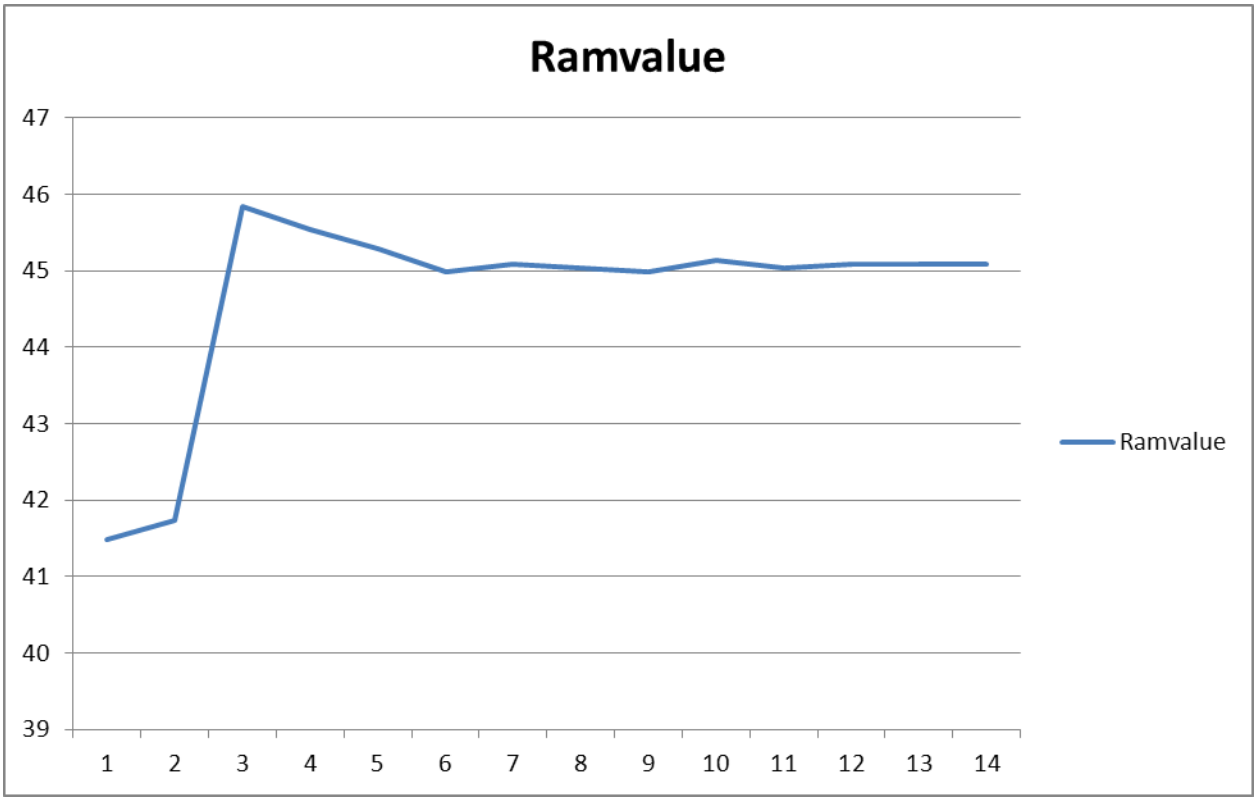


Figure 81 RAM Percentage Instace 1, 4-VMs, 160000 loop, 100 users

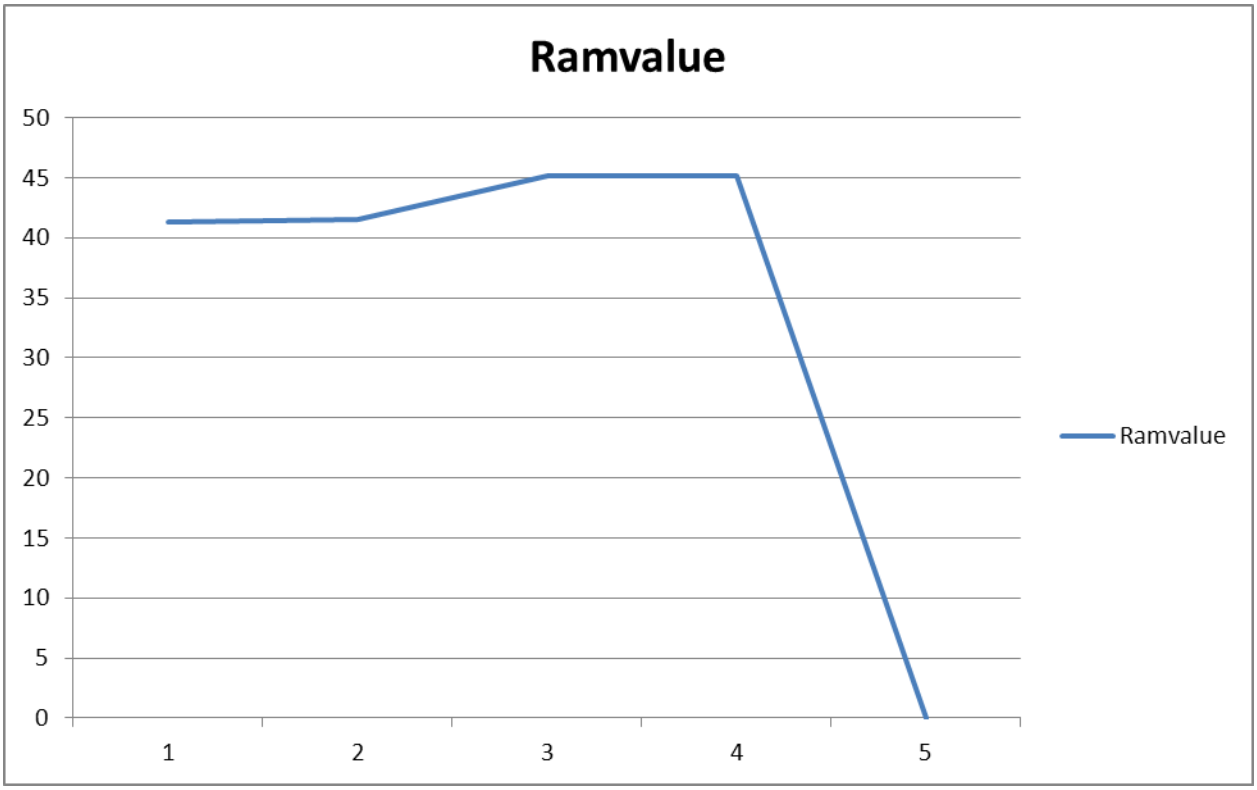


Figure 82 RAM Percentage Instace 2, 4-VMs, 160000 loop, 100 user

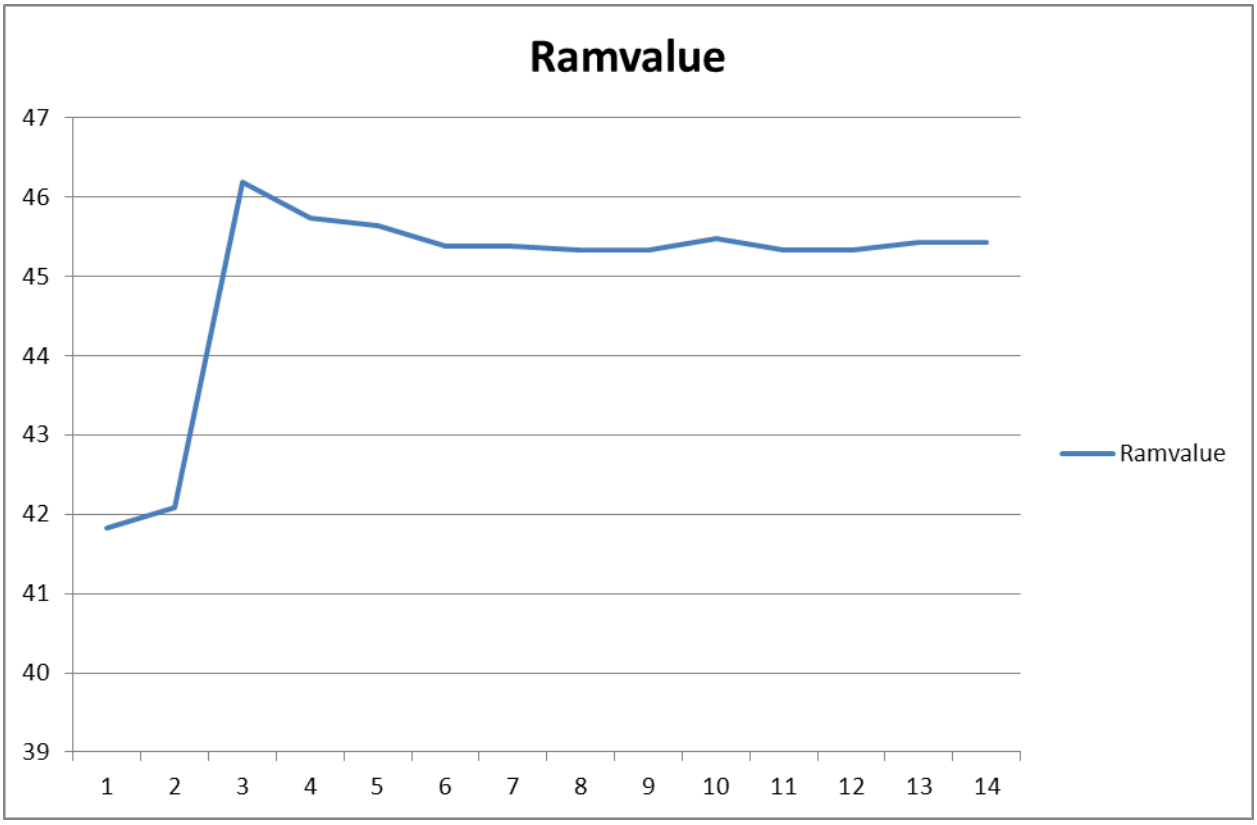


Figure 83 RAM Percentage Instace 3, 4-VMs, 160000 loop, 100 users

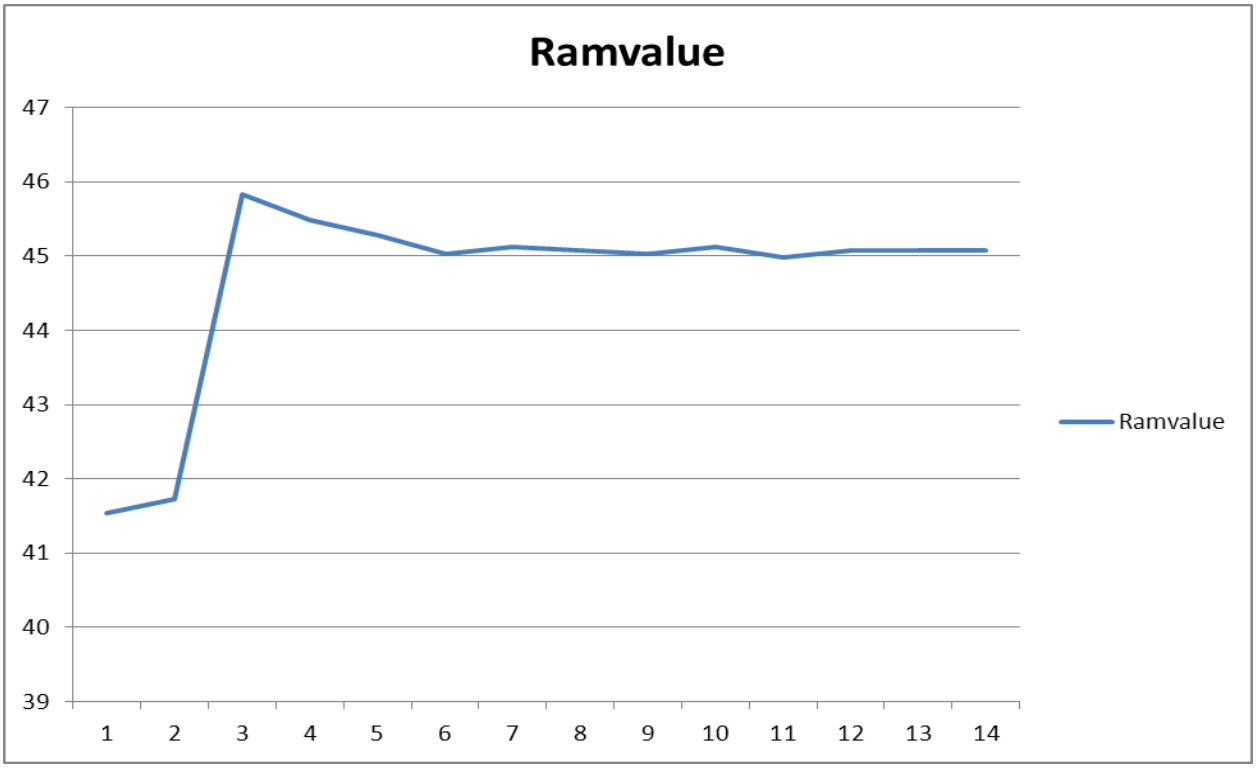


Figure 84 RAM Percentage Instace 4, 4-VMs, 160000 loop, 100 users

Συμπεράσματα

Με βάση τα παραπάνω αποτελέσματα, πίνακες και γραφήματα, μπορούν να εξαχθούν σημαντικά συμπεράσματα.

Έχοντας ένα συγκεκριμένο αριθμό χρηστών που «χτυπάνε» συνεχόμενα HTTP requests τύπου GET σε κάποιο virtual machine, αυτό έχει ως αποτέλεσμα την κατανάλωση πόρων του συστήματος, τόσο σε CPU όσο σε μνήμη RAM και την καταπόνησή του σε σχέση με την κατάσταση IDLE. Χρησιμοποιώντας loadbalancer και αλγόριθμο round-robin ισομοιράζεται ο φόρτος στα μηχανήματα.

Θεωρούμε ως κριτήρια-παραμέτρους τα εξης:

- Αριθμός χρηστών/threads
- Απαιτητικότητα php κώδικα
- Αριθμός μηχανημάτων

Αρχικά με 2 μηχανήματα και λιγότερο απαιτητικό φορτίο , αυξάνοντας τους χρήστες παρατηρήσαμε μια ελαφρά αλλα σημαντική σταδιακή αύξηση της κατανάλωσης των πόρων, ιδίως της CPU, χωρις να διαταράσσεται όμως η ομαλή λειτουργία κάποιου VM ή να ξεπερνάμε κάποιο όριο.
Στη συνέχεια όμως διπλασιάζοντας το φόρτο κάθε request και από 20 χρήστες και άνω, η CPU σε ένα από τα 2 μηχανήματα ξεπερνά το όριο 90% με αποτέλεσμα να τίθεται σε κατάσταση DOWN. Το monitoring πέτυχε.
Ακριβώς με τις ίδιες παραμέτρους του προηγούμενου σεναρίου, αυξάνοντας τον αριθμό των VMs από 2 σε 4 , ισομοιράζεται καλύτερα ο φόρτος οπότε οι πόροι καταναλώνονται σε επιτρεπτά όρια και δεν εμφανίζεται critical περίπτωση.
Θέλοντας να στρεσάρουμε ακόμα περισσότερο την υποδομή μας, διπλασιάζουμε για ακόμα μια φορά το φόρτο κάθε request με αποτέλεσμα ένα από τα μηχανήματα να πέφτει σε κατάσταση DOWN ενώ τα υπόλοιπα συνεχίζουν να είναι UP και με σημαντική κατανάλωση των πόρων τους.

Καταληκτικά , γίνεται εμφανές ότι οι πόροι ενός μηχανήματος καταναλώνονται τόσο με την αύξηση των συνδεδεμένων χρηστών όσο και με την αύξηση του φόρτου κάθε request. Στην δεύτερη περίπτωση το φαινόμενο αυτό γίνεται πιο εμφανές αφού καταφέρνει να φέρει στα όρια τους τα μηχανήματα. Τότε γίνεται σωστή αντιμετώπιση της κατάστασης μέσω της πλατφόρμας monitoring που υλοποιήθηκε.

Παράρτημα

Παρακάτω θα περιγραφούν εκτενώς διάφορες τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της υπηρεσιοστρεφούς πλατφόρμας.

6.1 Java

Γενικά:

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η Java. Πρόκειται για μια αντικειμενοστραφής, βασισμένη-σε-κλάσεις γλώσσα η οποία σχεδιάστηκε με σκοπό να έχει όσο το δυνατόν λιγότερες εξαρτήσεις εφαρμογής. Σημαντικό χαρακτηριστικό της είναι η φορατότητα, δηλαδή τα προγράμμματα σε γλώσσα Java τρέχουν παρόμοια σε οποιοδήποτε hardware ή πλατφόρμα λειτουργικού συστήματος. [17]

Χαρακτηριστικά:

Παρακάτω γίνεται αναφορά σε κάποια ιδιαίτερα χαρακτηριστικά της γλώσσας Java που γίνονται εμφανή στο project και τα οποία έχουν απόλυτη σχέση με την ιδιότητα του αντικειμενοστραφούς προγραμματισμού.

- Κληρονομικότητα (Inheritance)

Η αρχή της κληρονομικότητας βασίζεται στη δημιουργία μιας νέας κλάσης από μια υπάρχουσα με αποτέλεσμα να επιτυγχάνεται επαναχρησιμοποίηση κώδικα. Με αυτό τον τρόπο, τα χαρακτηριστικά και η συμπεριφορά της αρχικής κλάσης κληροδοτούνται στην νέα κλάση και είναι δυνατή η επέκταση με νέες μεταβλητές και μεθόδους.

- Υπερκάλυψη Μεθόδων(Method Overriding)

Πρόκειται για μια διαδικασία όπου μια κλάση επανα-υλοποιεί μία μέθοδο που κληρονόμησε απο κάποια αρχική κλάση.

- Πολυμορφισμός(Polymorphism)

Στον object-oriented προγραμματισμό ο πολυμορφισμός αναφέρεται στη δυνατότητα να χειριζόμαστε αντικείμενα που ανήκουν στην ίδια ιεραρχία κλάσεων σαν να ήταν αντικείμενα της κλάσης βάσης.

- Εξαιρέσεις(Exceptions)

Το exception είναι ένα γεγονός που συμβαίνει κατά τη διάρκεια εκτέλεσης ενός προγράμματος και το οποίο αλλάζει την φυσιολογική ροή των εντολών. Όταν λαμβάνει χώρα κάποιο λάθος μέσα σε μια μέθοδο, δημιουργείται ένα exception αντικείμενο το οποίο περιέχει πληροφορία για τον τύπο του λάθος και τη στιγμή που αυτό εμφανίστηκε. Σκοπός του exception handling είναι να «πιάνει» τις εξαιρέσεις, να τις χειρίζεται αναλόγως και να συνεχίζει την εκτέλεση του προγράμματος.

JEE:

Το Java EE είναι η Java υπολογιστική πλατφόρμα της Oracle για οργανισμούς και επιχειρήσεις. Παρέχει μια διεπαφή/API και runtime περιβάλλον για την ανάπτυξη και λειτουργία λογισμικού περιλαμβανομένου δικτυακών υπηρεσιών. Το Java EE περιλαμβάνει αρκετές API προδιαγραφές όπως e-mail, xml, javabeans, servlets κτλ. και ορίζει πώς συντονίζονται. Αυτό επιτρέπει στους

προγραμματιστές να δημιουργούν τις δικές τους εφαρμογές, φορητές και επεκτάσιμες. [18]

SIGAR:

Το SIGAR (System Information Gatherer) της Hyperic αποτελεί ένα API πλατφόρμας για τη συλλογή πληροφορίας συστήματος. Χρήστες του SIGAR API. Οι χρήστες αποκτούν "φορητή" πρόσβαση σε monitoring data όπως για παράδειγμα cpu, μέσο φορτίο, δισκος, ram, ανίχνευση δικτυακών διεπαφών, πληροφορίες εγκατάστασης και διάφορες μετρικές. [19]

6.2 Eclipse

Πρόκειται για ένα ενοποιημένο περιβάλλον ανάπτυξης. Περιλαμβάνει ένα βασικό χώρο εργασίας (workspace) και ένα επεκτάσιμο σύστημα από plug-in.
Τα plug-in αποτελούν κομμάτια λογισμικού που προσθέτουν ένα συγκεκριμένο χαρακτηριστικό σε μια ήδη υπάρχουσα εφαρμογή λογισμικού.
Χρησιμοποιήθηκε το Eclipse Software Development Kit (SDK) για developing σε γλώσσα Java. Συγκεκριμένα η έκδοση Eclipse Java EE Kepler.

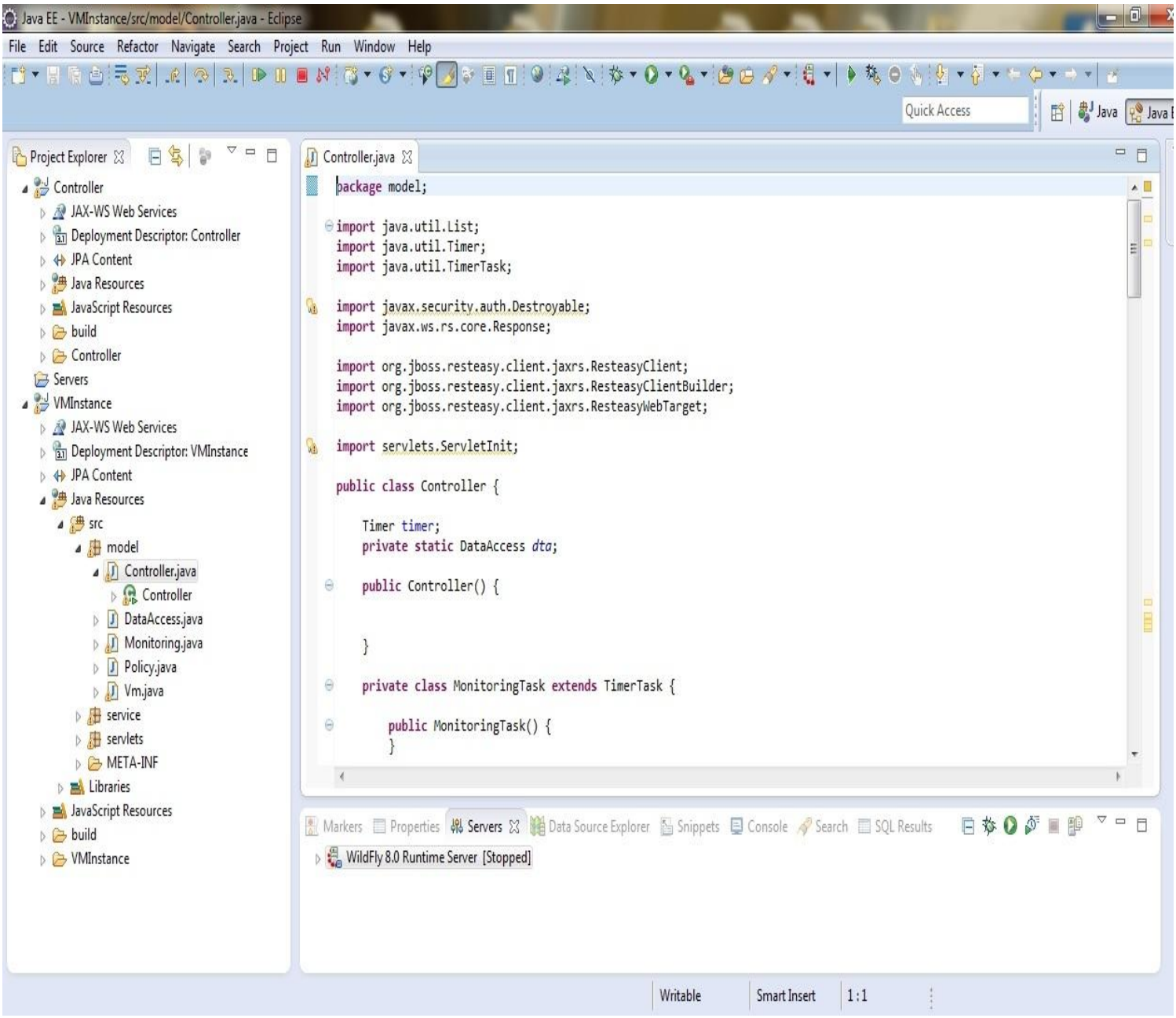


Figure 91 Snapshot από Eclipse

Το Eclipse SDK περιλαμβάνει εργαλεία ανάπτυξης Java (JDT), προσφέροντας Java μεταγλωττιστή (compiler) μαζί με ένα πλήρες μοντέλο από Java αρχεία. Αυτό επιτρέπει προχωρημένες τεχνικές refactoring και ανάλυσης κώδικα.

Το ιδιαίτερο χαρακτηριστικό του Eclipse είναι η δυνατότητα υποστήριξης servers, όπως ο Apache Tomcat και ο WildFly Server (που χρησιμοποιήθηκε στο project). Ως αποτέλεσμα επιτρέπεται στο χρήστη η εύρεση λαθών (debugging) και η παρακολούθηση βήμα-βήμα της εκτέλεσης της εφαρμογής καθώς «τρέχει» στον εκάστοτε server.

Τέλος ανάλογο χαρακτηριστικό του Eclipse είναι η ύπαρξη συστημάτων διαχείρισης βάσεων δεδομένων. Πιο συγκεκριμένα χρησιμοποιήθηκε το MySQL Workbench για τη δημιουργία των databases που ήταν απαραίτητες για την υλοποίηση του project.

6.3 RESTful Web Services

Το REST (Representational state transfer) μπορεί να θεωρηθεί ως μια αφηρημένη έννοια του Παγκόσμιου Ιστού. Το REST αρχιτεκτονικό μοντέλο εφαρμόζεται κυρίως για την ανάπτυξη web υπηρεσιών ως εναλλακτική λύση σε τεχνολογίες υπολογιστικής διανομής όπως το SOAP. Οι web υπηρεσίες που υπόκεινται σε κανόνες που βρίσκονται στα όρια της αρχιτεκτονικής του REST συχνά χαρακτηρίζονται ως “RESTful”. [20]

Το REST ενσαρκώνει μια stateless client-server αρχιτεκτονική στην οποία οι web υπηρεσίες θεωρούνται πόροι που ταυτοποιούνται μέσω των URL τους. Οι πελάτες web υπηρεσιών που θέλουν να χρησιμοποιήσουν αυτούς τους πόρους έχουν πρόσβαση σε ένα συγκεκριμένο είδος αναπαράστασης μεταφέροντας περιεχόμενο εφαρμογής μέσω της χρήσης ενός μικρού ορισμένου συνόλου μεθόδων που περιγράφουν την ενέργεια που θα εκτελεστεί πάνω στον πόρο. [21]

Ένα από τα χαρακτηριστικά-κλειδιά μιας RESTful Web υπηρεσίας είναι η

ξεκάθαρη χρήση HTTP μεθόδων με τέτοιο τρόπο ώστε να ακολουθείται το πρωτόκολλο όπως ορίζεται στο RFC 2616.

HTTP βασισμένα RESTful APIs ορίζονται με τα παρακάτω χαρακτηριστικά:

- ένα URI-βάση , πχ. http://localhost:8080/project/service/vm
- κάποιο media type όπως πχ. JSON ή XML που θα αναλυθούν παρακάτω
- βασικές HTTP μεθόδους όπως
    1. GET: ανάκτηση κάποιου πόρου
    2. POST: δημιουργία πόρου στον server
    3. PUT: τροποποίηση/ενημέρωση πόρου
    4. DELETE: διαγραφή πόρου

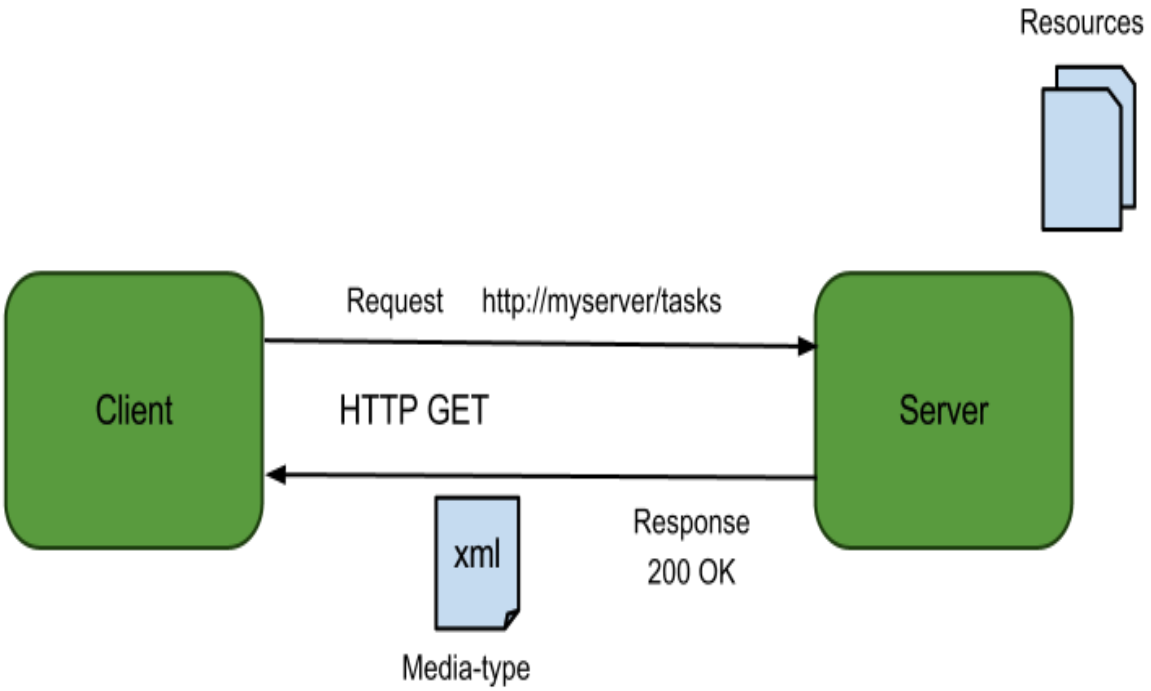


Figure 92 Παράδειγμα GET http μεθόδου (25)

Χαρακτηριστικά του REST: [22]

I. Κλιμάκωση με σκοπό την υποστήριξη μεγάλου αριθμού στοιχείων αλλά και αλληλεπιδράσεων μεταξύ αυτών των στοιχείων
II. Απλότητα των διεπαφών
III. Δυνατότητα τροποποίησης των στοιχείων με βάση τις ανάγκες
IV. Φορητότητα
V. Ορατότητα της επικοινωνίας μεταξύ των στοιχείων
VI. Αξιοπιστία λόγω αντίστασης σε λάθη σε επίπεδο συστήματος

6.3.1 Jboss/Wildfly

Το Wildfly, γνωστό μέχρι πρότινος και ως Jboss, είναι ένας application server , δημιουργία της εταιρίας Jboss, υπό ανάπτυξη από την Red Hat.

Figure 93 WildFly logo (26)

Είναι γραμμένο σε Java,εφαρμόζει την Java EE προδιαγραφή και μπορεί να τρέξει σε πολλαπλές πλατφόρμες.

6.3.2 RESTEasy

Το RESTEasy είναι ένα JBoss project που παρέχει ποικίλα πλαίσια (frameworks) με σκοπό το "χτίσιμο" RESTful Web υπηρεσιών και RESTful Java εφαρμογών. Είναι μια πλήρως πιστοποιημένη και φορητή εφαρμογή της JAX-RS προδιαγραφής. Το JAX-RS είναι μία νέα JCP προδιαγραφή η οποία παρέχει ένα Java API για RESTful Web υπηρεσίες πάνω στο HTTP πρωτόκολλο. [23]

Το RESTEasy μπορεί να τρέξει σε οποιοδήποτε Servlet container αλλά καλύτερη ενσωμάτωση με τον Jboss Application Server είναι διαθέσιμη για καλύτερη αλληλεπίδραση χρήστη και περιβάλλοντος.

6.4 JSON

Το JSON (JavaScript Object Notation) είναι μια ανοιχτού προτύπου μορφή ανταλλαγής πληροφορίας η οποία χρησιμοποιεί καταννοητό από τον άνθρωπο κείμενο για τη μετάδοση αντικειμένων πληροφορίας τα οποία αποτελούνται από ζευγάρια attribute-value (χαρακτηριστικό-τιμή). Χρησιμοποείται κυρίως για μεταδοση δεδομένων μεταξύ ενός server και μιας web εφαρμογής, ως εναλλακτική λύση στην χρήση XML. [24]

Το JSON είναι μια μορφή δεδομένων ανεξάρτητη από γλώσσες, δεδομένου ότι κώδικας για parsing και δημιουργία JSON είναι διαθέσιμος σε όλες τις προγραμματιστικές γλώσσες. [25]

Χτίζεται πάνω σε δύο δομές:

-μια συλλογή από ζευγάρια ονόματος-τιμής , που σε μια προγραμματιστική γλώσσα μπορεί να χαρακτηριστεί ως object, struct, record κτλ.

-μια ταξινομημένη λίστα τιμών (πίνακας, λίστα)

Παράδειγμα JSON

```
{
	"Name": "Bill",
	"LastName": "Papadopoulos",
	"Age": 24,
	"Gender": {
		"Type": "Male"
	},
	"Address": {
		"StreetAddress": "Venizelou 11",
		"City": "Athens",
		"Country": "Greece",
		"PostalCode": "17122"
	},
	"PhoneNumber": [
		{
			"Type": "Home",
			"Number": "210 8888888"
		},
		{
			"Type": "Mobile",
			"Number": "6987766555"
		}
	],
	"Occupation": {
		"Job": "Software Engineer"
		"Years": "11"
	}
}
```

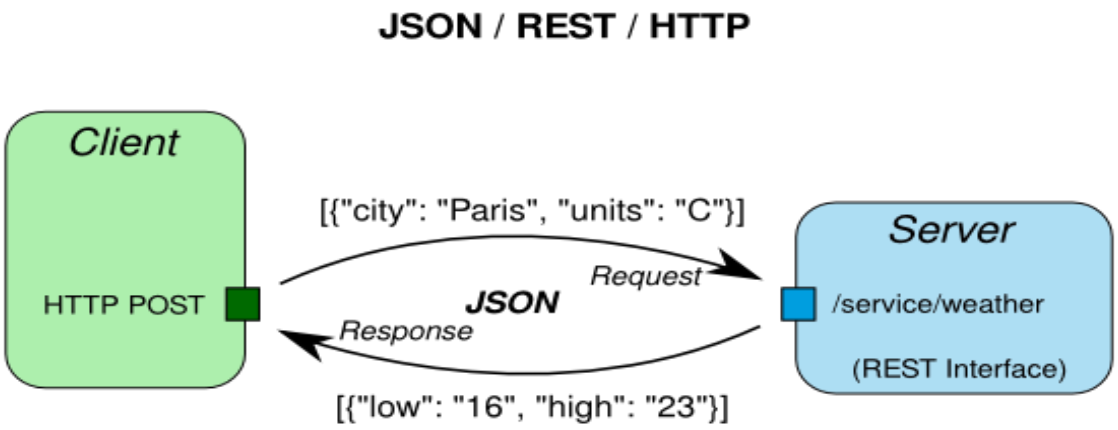


Figure 94 JSON and RESTful API (27)

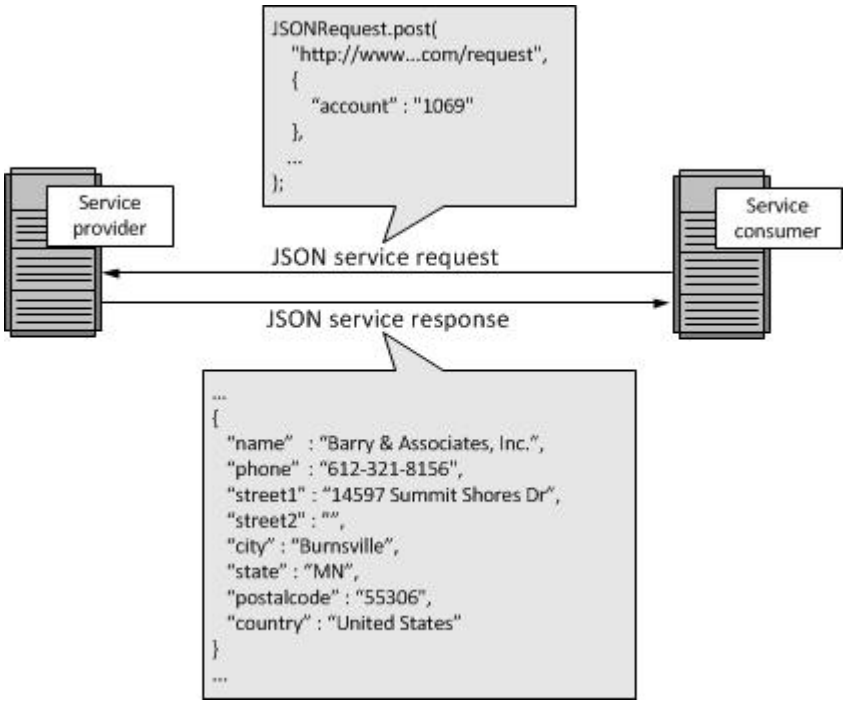


Figure 95 JSON structure for Web Services (28)

6.5 MySQL Workbench

Το MySQL Workbench αποτελεί εργαλείο σχεδιασμού βάσεων δεδομένων ενσωματώνοντας ανάπτυξη σε γλώσσα SQL , δημιουργία, διαχείριση, σχεδιασμό και συντήρηση databases σε ένα ενιαίο περιβάλλον ανάπτυξης για MySQL. Το λογισμικό MySQL προσφέρει ένα ταχύ, πολυ-νηματικό και αξιόπιστο SQL (Structured Query Language) database server.

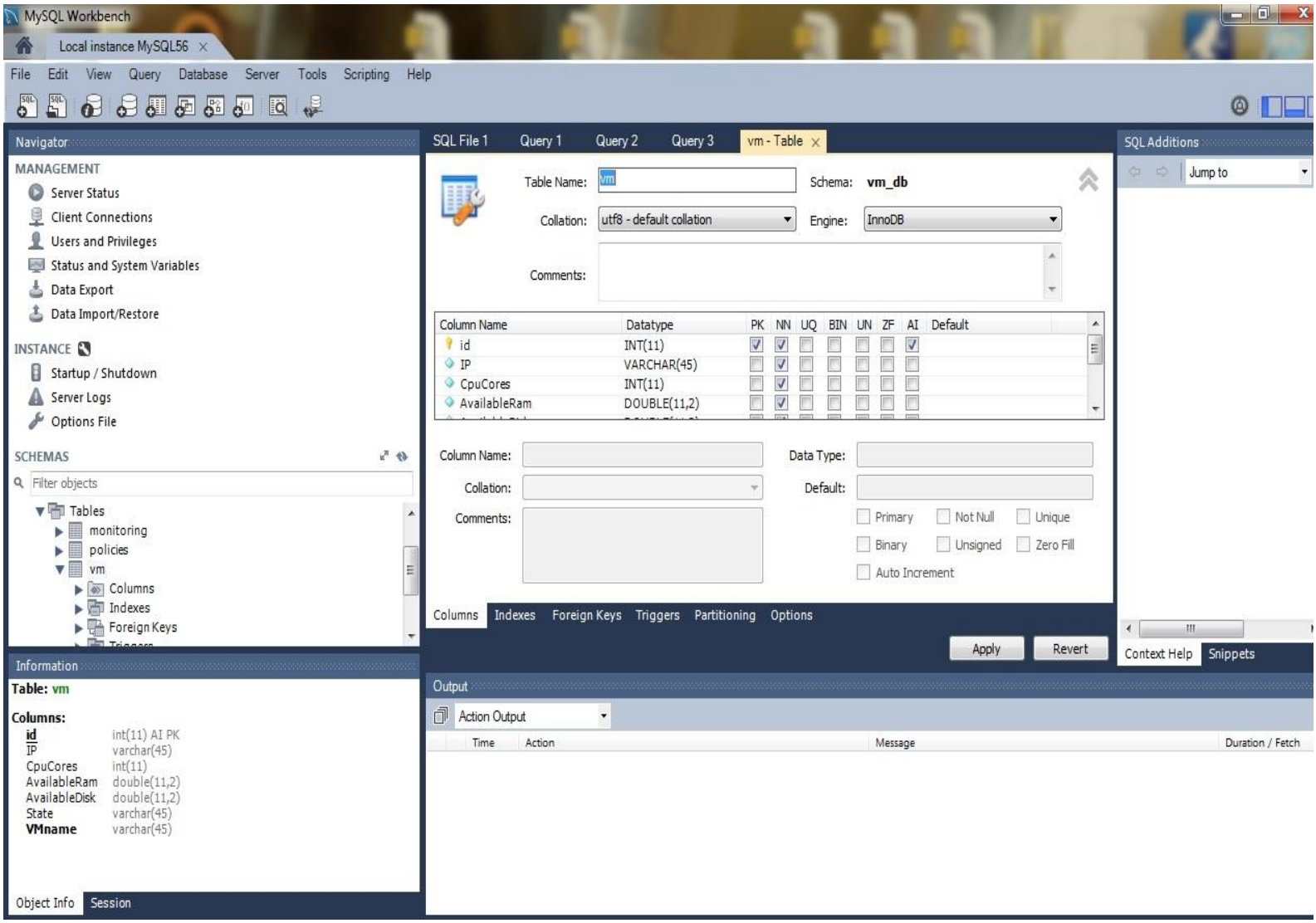


Figure 96 Snapshot from MySQL Workbench

6.6 XML

Η Extensible Markup Language (XML) είναι μια γλώσσα σήμανσης που ορίζει ένα σύνολο κανόνων για την κωδικοποίηση σε μορφή που είναι κατανοητή τόσο από τον άνθρωπο όσο και από τη μηχανή. [26]

Ένα απλό γενικό παράδειγμα XML αρχείου ακολουθεί:

```
<Belgian_menu>
      <food>
            <name>Belgian Waffles</name>
            <price>$5</price>
            <description>
                  Two of our famous Belgian Waffles with plenty of real
                  maple syrup
            </description>
            <calories>650</calories>
      </food>
      <food>
            <name>Strawberry Belgian Waffles</name>
            <price>$7</price>
            <description>
                  Light Belgian waffles covered with strawberries and
                  whipped cream
            </description>
            <calories>900</calories>
      </food>
      <food>
            <name>Berry-Berry Belgian Waffles</name>
            <price>$8</price>
            <description>
                  Light Belgian waffles covered with an assortment of
                  fresh berries and whipped cream
            </description>
            <calories>900</calories>
      </food>
</Belgian_menu>
```

Άξιο αναφοράς είναι το XML schema το οποίο ορίζει πως να περιγράψεις τυπικά τα στοιχεία σε ένα αρχείο XML και χρησιμοποιείται για την επαλήθευση και εγκυρότητα κάθε "αντικειμένου" σε ένα document. Τεχνικά αποτελεί μια αφηρημένη συλλογή από metadata.

6.7 Servlet programming

Το servlet είναι μια κλάση της γλώσσας Java που χρησιμοποιείται για την επέκταση των δυνατοτήτων ενός server. Χρησιμοποείται συνήθως για την επέκταση των εφαρμογών που "φιλοξενούνται" από web servers ώστε να θεωρηθούν ως Java applets που τρέχουν σε servers. [27]

Χρήση:

- επεξεργασία/αποθήκευση πληροφορίας από μια html φόρμα.
- παροχή δυναμικού περιεχομένου όπως τα αποτελέσματα μιας βάσης δεδομένων.

Ένα servlet συμβαδίζει με το Java Servlet API , ένα πρότυπο για την εφαρμογή κλάσεων που απαντούν σε requests και το οποίο περιέχεται στο Java package javax.servlet. Το servlet είναι ένα object που δέχεται ένα request και δημιουργεί ένα αντίστοιχο response. Μέσω της κλήσης των μεθόδων init() και destroy() επιτυγχάνεται η αρχικοποίηση του servlet instance και ο τερματισμός της υπηρεσίας αντίστοιχα. [28]

Source Code

Controller

VM.java

```
package controller.model;

import java.io.Serializable;
import javax.persistence.*;


/**
 * The persistent class for the vm database table.
 *
 */
@Entity
@Table(name="vm")
@NamedQuery(name="Vm.findAll", query="SELECT v FROM Vm v")
public class Vm implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;

	private double availableDisk;

	private double availableRam;

	private int cpuCores;

	private String ip;

	private String state;

	private String VMname;


	public Vm() {
	}

	public int getId() {
```

```
        return this.id;
}

public void setId(int id) {
        this.id = id;
}

public double getAvailableDisk() {
        return this.availableDisk;
}

public void setAvailableDisk(double availableDisk) {
        this.availableDisk = availableDisk;
}

public double getAvailableRam() {
        return this.availableRam;
}

public void setAvailableRam(double availableRam) {
        this.availableRam = availableRam;
}

public int getCpuCores() {
        return this.cpuCores;
}

public void setCpuCores(int cpuCores) {
        this.cpuCores = cpuCores;
}

public String getIp() {
        return this.ip;
}

public void setIp(String ip) {
        this.ip = ip;
}

public String getState() {
        return this.state;
}

public void setState(String state) {
        this.state = state;
}

public String getVMname() {
```

```
		return this.VMname;
	}

	public void setVMname(String VMname) {
		this.VMname = VMname;
	}

}
```

Monitoring.java

```
package controller.model;

import java.io.Serializable;

import javax.persistence.*;


/**
 * The persistent class for the monitoring database table.
 *
 */
@Entity
@Table(name="monitoring")
//@NamedQuery(name="Monitoring.findAll", query="SELECT m FROM Monitoring m")
public class Monitoring implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;

	private double cpuvalue;

	private double diskvalue;

	private double ramvalue;

	private String timestamp;

	//bi-directional many-to-one association to Vm
	@ManyToOne
	@JoinColumn(name="VM")
	private Vm vmBean;

	public Monitoring() {
```

```
}

public int getId() {
	return this.id;
}

public void setId(int id) {
	this.id = id;
}

public double getCpuvalue() {
	return this.cpuvalue;
}

public void setCpuvalue(double cpuvalue) {
	this.cpuvalue = cpuvalue;
}

public double getDiskvalue() {
	return this.diskvalue;
}

public void setDiskvalue(double diskvalue) {
	this.diskvalue = diskvalue;
}

public double getRamvalue() {
	return this.ramvalue;
}

public void setRamvalue(double ramvalue) {
	this.ramvalue = ramvalue;
}

public String getTimestamp() {
	return this.timestamp;
}

public void setTimestamp(String timestamp) {
	this.timestamp = timestamp;
}

public Vm getVmBean() {
	return this.vmBean;
}

public void setVmBean(Vm vmBean) {
	this.vmBean = vmBean;
```

```
	}
}
```

Policy.java

```
package controller.model;

import java.io.Serializable;
import javax.persistence.*;


/**
 * The persistent class for the policies database table.
 *
 */
@Entity
@Table(name="policies")
@NamedQuery(name="Policy.findAll", query="SELECT p FROM Policy p")
public class Policy implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private int id;

	private double CPUthres;

	private double RAMthres;

	//bi-directional many-to-one association to Vm
	@ManyToOne
	@JoinColumn(name="VM")
	private Vm vmBean;

	public Policy() {
	}

	public int getId() {
		return this.id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public double getCPUthres() {
```

```
		return this.CPUthres;
	}

	public void setCPUthres(double CPUthres) {
		this.CPUthres = CPUthres;
	}

	public double getRAMthres() {
		return this.RAMthres;
	}

	public void setRAMthres(double RAMthres) {
		this.RAMthres = RAMthres;
	}

	public Vm getVmBean() {
		return this.vmBean;
	}

	public void setVmBean(Vm vmBean) {
		this.vmBean = vmBean;
	}

}
```

DataAccess.java

```
package controller.model;

import java.util.List;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

@Stateless
public class DataAccess {

	@PersistenceContext(unitName="Controller")
	private EntityManager em;

	@Transactional
	public void createVm(Vm vm){
		em.persist(vm);
		em.flush();
```

```
}

@Transactional
public void updateVm(Vm vm){
        vm = em.merge(vm);
        em.flush();
}

@Transactional
public void createMonitoring(Monitoring monitoring){
        em.persist(monitoring);
        em.flush();
}

@Transactional
public void Monitoring(Monitoring monitoring){
        monitoring = em.merge(monitoring);
        em.flush();
}

@Transactional
public void createPolicy(Policy policy){
        em.persist(policy);
        em.flush();
}

@Transactional
public void Policy(Policy policy){
        policy = em.merge(policy);
        em.flush();
}

public List<Vm> getVms(){
        return (List<Vm>)em.createQuery("select v from Vm v where
        v.state='UP'",Vm.class).getResultList();
}

public Vm getVm(Integer id){
        return (Vm)em.find(Vm.class, id);
}

public void saveMonitoringData(Monitoring mon){
        em.persist(mon);
        em.flush();
}

public void savePolicyData(Policy pol){
        em.persist(pol);
```

```
            em.flush();
        }

}
```

ControllerServlet.java

```
package controller;

import java.util.List;
import java.util.Timer;
import java.util.TimerTask;

import javax.ejb.EJB;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.ws.rs.core.Response;

import org.jboss.resteasy.client.jaxrs.ResteasyClient;
import org.jboss.resteasy.client.jaxrs.ResteasyClientBuilder;
import org.jboss.resteasy.client.jaxrs.ResteasyWebTarget;

import controller.model.DataAccess;
import controller.model.Monitoring;
import controller.model.Policy;
import controller.model.Vm;

public class ControllerServlet extends HttpServlet{


        private static final long serialVersionUID = 1L;
        Timer timer;
        @EJB
        private DataAccess dta;

        public ControllerServlet() {

        }

        @Override
        public void init(ServletConfig config) throws ServletException{
            super.init(config);
            timer = new Timer();
            timer.schedule(new MonitoringTask(), 0, 20*1000);
        }
```

```
private class MonitoringTask extends TimerTask {

	public MonitoringTask() {
	}

	public void run() {
		List<Vm> vmList = (List<Vm>) dta.getVms();
		for (Vm vm : vmList) {

			Monitoring monit = getMonitoringData(vm);

			boolean critical = false;
			critical = CompareModule(monit);
			if (critical) {

				destroyVm(vm);
			}
		}
		System.out.println("Waiting 20 seconds...");

	}

}


private boolean CompareModule(Monitoring mon) {

	Vm vm = mon.getVmBean();
	Policy policy = getPolicyData(vm);

	double CpuLoad = mon.getCpuvalue();
	double CpuThres = policy.getCPUthres();
	if (CpuLoad >= (CpuThres * 100)) {

		vm.setState("DOWN");
		System.out.println("CPU status is critical");
		System.out.println("VM state is DOWN");
		return true;

	}

	double UsedDiskSpacePerc = mon.getDiskvalue();
	double SystemDisk = vm.getAvailableDisk();
	if (UsedDiskSpacePerc >= (0.95 * 100)) {

		vm.setState("DOWN");
		System.out.println("Disk status is critical");
```

```
                System.out.println("VM state is DOWN");
                return true;

        }

        double UsedRAM = mon.getRamvalue();
        double RAMthres = policy.getRAMthres();
        double SystemRAM = vm.getAvailableRam();
        if (UsedRAM >= (RAMthres * 100)) {

                vm.setState("DOWN");
                System.out.println("RAM status is critical");
                System.out.println("VM state is DOWN");
                return true;

        }

        return false;

}

private Monitoring getMonitoringData(Vm vm) {

        // Monitoring mon = dta.getMonitoring(vm);
        // 1) IMPLEMENT CLIENT TO RETRIEVE MONITORING DATA
        // 2) USE dta TO SAVE MONITORING DATA TO DATABASE
        ResteasyClient client = new ResteasyClientBuilder().build();
        ResteasyWebTarget target = client.target("http://" + vm.getIp()
                        + ":8080/VMInstance/service/monitoring");
        Response response = target.request().get();
        Monitoring mon = (Monitoring) response.readEntity(Monitoring.class);
        mon.setVmBean(vm);
        dta.saveMonitoringData(mon);

        return mon;
}


private Policy getPolicyData(Vm vm) {

        // Policy policy = getPolicyData(vm);
        // 1) IMPLEMENT CLIENT TO RETRIEVE POLICY DATA
        // 2) USE dta TO SAVE POLICY DATA TO DATABASE

        ResteasyClient client = new ResteasyClientBuilder().build();
        ResteasyWebTarget target = client.target("http://" + vm.getIp()
                        + ":8080/VMInstance/service/policies");
        Response response = target.request().get();
```

```
		Policy pol = (Policy) response.readEntity(Policy.class);
		pol.setVmBean(vm);
		dta.savePolicyData(pol);

		return pol;
	}

	public void destroyVm(Vm vm) {
		// 1) RETRIEVE VM FROM DATABASE BY IP
		// 2) SET VM STATE TO 'DOWN'
		// 3) UPDATE VM DATABASE

		ResteasyClient client = new ResteasyClientBuilder().build();
		ResteasyWebTarget target = client.target("http://" + vm.getIp()
				+ ":8080/VMInstance/service/vms/" + vm.getId());
		Response response = target.request().get();
		Vm vM = (Vm) response.readEntity(Vm.class);

		vM.setState("DOWN");
		dta.updateVm(vM);

	}

}
```

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
	<persistence-unit name="Controller" transaction-type="JTA">
	<jta-data-source>java:jboss/RestDS</jta-data-source>
		<class>controller.model.Monitoring</class>
		<class>controller.model.Policy</class>
		<class>controller.model.Vm</class>
	</persistence-unit>
</persistence>
```

VM Instance

ServletInit.java

```
package servlets;

import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.Enumeration;

import javax.ejb.EJB;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

import org.hyperic.sigar.FileSystemUsage;
import org.hyperic.sigar.Mem;
import org.hyperic.sigar.Sigar;
import org.hyperic.sigar.SigarException;

import model.DataAccess;
import model.Vm;

public class ServletInit extends HttpServlet {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	@EJB
	private DataAccess dta;
	private String ip;

	@Override
	public void destroy() {
		// 1) RETRIEVE VM FROM DATABASE BY IP
		// 2) SET VM STATE TO 'DOWN'
		System.out.println("I AM GOING DOWN");
		Vm vm = (Vm) dta.getVmByIp(ip);
		vm.setState("DOWN");
```

```
        dta.updateVm(vm);
}

@Override
public void init(ServletConfig config) throws ServletException {
        super.init(config);
        // 1) Retrieve IP from OS
        // 2) Retrieve CPU,RAM,DISK from OS
        // 3) Write all of the above to database

        Vm vm = new Vm();
        ip = null;

        try {
                Enumeration<NetworkInterface> interfaces = NetworkInterface
                                .getNetworkInterfaces();
                while (interfaces.hasMoreElements()) {
                        NetworkInterface iface = interfaces.nextElement();
                        // filters out 127.0.0.1 and inactive interfaces
                        if (iface.isLoopback() || !iface.isUp())
                                continue;

                        Enumeration<InetAddress> addresses = iface.getInetAddresses();
                        while (addresses.hasMoreElements()) {
                                InetAddress addr = addresses.nextElement();
                                ip = addr.getHostAddress();
                                System.out.println(iface.getDisplayName() + " " + ip);
                        }
                }
        } catch (SocketException e) {
                throw new RuntimeException(e);
        }

        vm.setIp(ip);

        Sigar sigar = new Sigar();

        // double disk=system.getTotalDiskCapacity();
        FileSystemUsage diskvalue = null;
        try {
                diskvalue = sigar.getFileSystemUsage("/");
        } catch (SigarException e2) {
                // TODO Auto-generated catch block
                e2.printStackTrace();
        }
        double disk = (diskvalue.getTotal()/1024)/1024;
        System.out.print("Disk Capacity= ");
        System.out.println(disk);
```

```
                vm.setAvailableDisk(disk);

                // double ram=system.getRAM();
                Mem ramvalue = null;
                try {
                        ramvalue = sigar.getMem();
                } catch (SigarException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                }
                double ram = (ramvalue.getTotal()/1024)/1024;
                System.out.print("RAM= ");
                System.out.println(ram);
                vm.setAvailableRam(ram);

                try {
                        vm.setVMname(InetAddress.getLocalHost().getHostName());
                } catch (UnknownHostException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                vm.setState("UP");

                int cores = Runtime.getRuntime().availableProcessors();
                vm.setCpuCores(cores);

                if (dta.getVmByIp(ip) == null) {
                        dta.createVm(vm);
                }else {
                        vm = dta.getVmByIp(ip);
                        vm.setState("UP");
                        dta.updateVm(vm);
                }
        }

}
```

ServiceRegistry.java

```
package servlets;


import java.util.Arrays;
import java.util.HashSet;
```

```
import java.util.Set;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

import service.InstanceService;

@ApplicationPath("/")
public class ServiceRegistry extends Application {

        @SuppressWarnings("unchecked")
    public Set<Class<?>> getClasses() {
        return new HashSet<Class<?>>(Arrays.asList(InstanceService.class));
    }

}
```

InstanceService.java

```
package service;

import javax.ejb.EJB;
import javax.ejb.Singleton;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
```

```
import model.Monitoring;
import model.Policy;
import org.hyperic.sigar.CpuPerc;
import org.hyperic.sigar.FileSystemUsage;
import org.hyperic.sigar.Mem;
import org.hyperic.sigar.Sigar;
import org.hyperic.sigar.SigarException;

import java.text.SimpleDateFormat;

@Path("/service")
@Singleton
public class InstanceService {
	@EJB
	private model.DataAccess dao;

	@GET
	@Path("/policies")
	@Produces(MediaType.APPLICATION_JSON)
	public Response getPolicy() {
		Policy pol = new Policy();

		pol.setCPUthres(0.9);
		pol.setRAMthres(0.9);

		return Response.accepted(pol).build();
	}

	@GET
	@Path("/monitoring")
	@Produces(MediaType.APPLICATION_JSON)
	public Response newMonitoring() throws SigarException{
```

```
Sigar sigar = new Sigar();
Monitoring mon = new Monitoring();
System.out.println("DEBUG");
CpuPerc perc = sigar.getCpuPerc();
double CPUusage = perc.getCombined() * 100; //get current CPU usage percentage
System.out.print("CPU usage = ");
System.out.print(CPUusage);
System.out.println("%");
FileSystemUsage disk = sigar.getFileSystemUsage("/");
double diskvalue = ((disk.getTotal()-disk.getFree())/1024)/1024; // get used disk volume
System.out.print("Disk used volume = ");
System.out.println(diskvalue);
double UsedDiskPerc = disk.getUsePercent() * 100; // get percent of disk used
System.out.print("Disk used percent = ");
System.out.print(UsedDiskPerc);
System.out.println("%");
Mem ram = sigar.getMem();
double UsedMemory = (ram.getUsed()/1024)/1024; // get used memory volume
double TotalMemory = ((ram.getTotal()/1024)/1024);
double ramvalue = (UsedMemory/TotalMemory) * 100; // get percent of memory used
System.out.print("Memory used volume = ");
System.out.println(UsedMemory);
System.out.print("Memory used percent = ");
System.out.print(ramvalue);
System.out.println("%");
mon.setCpuvalue(CPUusage);
mon.setDiskvalue(UsedDiskPerc);
mon.setRamvalue(ramvalue);
java.util.Date date = new java.util.Date(); //get timestamp
java.sql.Timestamp sq = new java.sql.Timestamp(date.getTime());
String S = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss").format(sq);
```

```
            mon.setTimestamp(S);

            //dao.createMonitoring(mon);
            //1. GET SYSTEM MONITORING WITH SIGAR API
            //2. BUILD MONITORING JAVA OBJECT
            //3. RETURN IT TO SERVICE CLIENT
            return Response.accepted(mon).build();
        }

        @GET
        @Path("/vms/{id}")
        @Produces(MediaType.APPLICATION_JSON)
        public Response getVm(@PathParam("id") Integer id){
            return Response.ok(dao.getVm(id)).build();
        }

}
```

DataAccess.java

```
package model;

import java.util.List;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import javax.transaction.Transactional;
```

```
@Stateless
public class DataAccess {

	@PersistenceContext(unitName = "VMInstance")
	private EntityManager em;

	@Transactional
	public void createVm(Vm vm) {
		em.persist(vm);
		em.flush();
	}

	@Transactional
	public void updateVm(Vm vm) {
		vm = em.merge(vm);
		em.flush();
	}

	@Transactional
	public void createMonitoring(Monitoring monitoring) {
		em.persist(monitoring);
		em.flush();
	}

	@Transactional
	public void Monitoring(Monitoring monitoring) {
		monitoring = em.merge(monitoring);
		em.flush();
	}

	@Transactional
	public void createPolicy(Policy policy) {
```

```
        em.persist(policy);
        em.flush();
}

@Transactional
public void Policy(Policy policy) {
        policy = em.merge(policy);
        em.flush();
}

public List<Vm> getVms() {
        return (List<Vm>) em.createQuery(
                        "select v from Vm v where v.state='UP'", Vm.class).getResultList();
}

public Vm getVm(Integer id) {
        return (Vm) em.find(Vm.class, id);
}

public void saveMonitoringData(Monitoring mon) {
        em.persist(mon);
        em.flush();
}

public Vm getVmByIp(String ip) {
        Query q = em.createQuery("select v from Vm v where v.ip='" + ip + "'",Vm.class);
        if (q.getResultList().isEmpty()) {
                return null;
        } else {
                return (Vm) q.getResultList().get(0);
        }
}
```

```
        public void savePolicyData(Policy pol) {
                em.persist(pol);
                em.flush();
        }
}
```

Table of figures

Table of Tables

Βιβλιογραφία

[1] P. Mell και T. Grace, «NIST Special Publication 800-145 The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,» September 2011. [Ηλεκτρονικό]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[2] Wikipedia, «Cloud computing,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Cloud_computing.

[3] B. Smye, «A Brief History of Cloud Computing,» April 2014. [Ηλεκτρονικό]. Available: http://www.business2community.com/cloud-computing/brief-history-cloud-computing-0858476.

[4] Ι. Σ. Βενιέρης, Δίκτυα ευρείας ζώνης, Τζιόλα, 2012.

[5] Oneserve, «A brief history of cloud computing,» 2014. [Ηλεκτρονικό]. Available: http://www.slideshare.net/Oneserve/a-brief-history-of-cloud-computing-33889054.

[6] P. Viswanathan, «Cloud Computing – Is it Really All That Beneficial?,» [Ηλεκτρονικό]. Available: http://mobiledevices.about.com/od/additionalresources/a/Cloud-Computing-Is-It-Really-All-That-Beneficial.htm.

[7] I. Tsagklis, «Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons,» April 2013. [Ηλεκτρονικό]. Available: http://www.javacodegeeks.com/2013/04/advantages-and-disadvantages-of-cloud-computing-cloud-computing-pros-and-cons.html.

[8] Ubuntu, «Metal as a Service,» 2014. [Ηλεκτρονικό]. Available: https://maas.ubuntu.com/.

[9] F. Stroud, «Metal-as-a-Service (MAAS),» [Ηλεκτρονικό]. Available: http://www.webopedia.com/TERM/M/metal-as-a-service_maas.html.

[10] Wikipedia, «Network as a service,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Network_as_a_service.

[11] P. Costa, M. Migliavacca, P. Pietzuch και A. L. Wolf, «NaaS: Network-as-a-Service in the Cloud,» 2012. [Ηλεκτρονικό]. Available: http://research.microsoft.com/en-us/um/people/pcosta/papers/costa12naas.pdf.

[12] B. Bardwell, «The Importance Of Monitoring, Even In A Managed Cloud Environment,» May 2012. [Ηλεκτρονικό]. Available: http://www.rackspace.com/blog/the-importance-of-monitoring-even-in-a-managed-cloud-environment/.

[13] E. Galstad, «Cloud Computing And Monitoring With Nagios,» [Ηλεκτρονικό]. Available: http://www.nagios.com/solutions/cloud-computing.

[14] Wikipedia, «Client–server model,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Client-server_model.

[15] Apprenda, «Cloud Middleware,» 2014. [Ηλεκτρονικό]. Available: http://apprenda.com/library/glossary/definition-cloud-middleware/.

[16] Openstack, «Openstack Cloud Administrator Guide,» 2014. [Ηλεκτρονικό]. Available: http://docs.openstack.org/admin-guide-cloud/content/ch_getting-started-with-

openstack.html.

[17] Wikipedia, «Java (programming language),» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Java_(programming_language).

[18] Wikipedia, «Java Platform, Enterprise Edition,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition.

[19] Hyperic, «Hyperic SIGAR API,» 2014. [Ηλεκτρονικό]. Available: http://www.hyperic.com/products/sigar.

[20] A. Rodriguez, «RESTful Web services: The basics,» November 2008. [Ηλεκτρονικό]. Available: http://www.ibm.com/developerworks/library/ws-restful/.

[21] S. Tyagi, «RESTful Web Services,» August 2006. [Ηλεκτρονικό]. Available: http://www.oracle.com/technetwork/articles/javase/index-137171.html.

[22] Wikipedia, «Representational state transfer,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Representational_state_transfer.

[23] RESTEasy, «RESTEasy,» 2014. [Ηλεκτρονικό]. Available: http://resteasy.jboss.org/.

[24] Wikipedia, «JSON,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/JSON.

[25] json.org, «Introducing JSON,» 2014. [Ηλεκτρονικό]. Available: http://json.org/.

[26] Wikipedia, «XML,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/XML.

[27] Wikipedia, «Java Servlet,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Java_Servlet.

[28] tomcat.apache.org, «javax.servlet,» [Ηλεκτρονικό]. Available: http://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/Servlet.html.

Βιβλιογραφία Εικόνων

1. Cloud Computing. *Wikidot.* [Ηλεκτρονικό] http://cs205sp14.wikidot.com/cloud-computing.
2. Cloud Computing From the Home. *tutorialspoint.* [Ηλεκτρονικό] http://www.tutorialspoint.com/shorttutorials/cloud-computing-from-the-home.
3. The essential characteristics of cloud computing. *flickr.* [Ηλεκτρονικό] www.flickr.com.
4. Historical Maps of Computer Networks. [Ηλεκτρονικό] http://personalpages.manchester.ac.uk/staff/m.dodge/cybergeography/atlas/historical.html.
5. VM (operating system). *Wikipedia.* [Ηλεκτρονικό] http://en.wikipedia.org/wiki/VM_(operating_system).
6. PK Switched Vs Circuit Switched. *Wikimedia.* [Ηλεκτρονικό] ommons.wikimedia.org/wiki/File:PK_Switched_Vs_Circuit_Switched.jpg.
7. Virtualization, Cloud Computing, Open Source and Beyond. [Ηλεκτρονικό] http://www.qyjohn.net/?p=2572.
8. EC2 Micro Instances. *Attune Infocom.* [Ηλεκτρονικό] http://attune-infocom.blogspot.gr/2013/02/EC2-Micro-Instances.html.
9. Public Cloud Virtualisation. *Core Technology Partners.* [Ηλεκτρονικό] http://www.coretechnologypartners.com.au/public-cloud.php.
10. What is Private Cloud? *Datamation.* [Ηλεκτρονικό] http://www.datamation.com/cloud-computing/what-is-private-cloud.html.
11. Community Cloud Model. *tutorialspoint.* [Ηλεκτρονικό] http://www.tutorialspoint.com/cloud_computing/cloud_computing_community_cloud_model.htm.
12. Hybrid Cloud. *Argowiki.* [Ηλεκτρονικό] http://argowiki.com/index.php?title=Hybrid_Cloud.
13. What To Consider Before Choosing An IaaS Provider. *CloudTimes.* [Ηλεκτρονικό] http://cloudtimes.org/2011/02/27/what-to-consider-before-choosing-an-iaas-provider/.
14. [Ηλεκτρονικό] https://www.zoho.com/creator/images/subpages/paas.gif.
15. Choosing SaaS as a Distribution Model - Pros and Cons. *DZone.* [Ηλεκτρονικό] http://java.dzone.com/articles/choosing-saas-distribution.
16. OpenStack or CloudStack ? *Linkedin.* [Ηλεκτρονικό] https://www.linkedin.com/pulse/article/20140925012324-246665791-openstack-or-cloudstack.
17. Metal as a Service. *Ubuntu.* [Ηλεκτρονικό] https://maas.ubuntu.com/.
18. Cloud Computing Network as a Service(NaaS). *tutorialspoint.* [Ηλεκτρονικό] http://www.tutorialspoint.com/cloud_computing/cloud_computing_network_as_a_service.htm.
19. cloud email. *fifosys.* [Ηλεκτρονικό] http://www.fifosys.com/the-cloud/cloud-email/.
20. CRM and the Cloud Platform – A Match Made in Heaven. *StrataBlue.* [Ηλεκτρονικό] http://stratablue.com/crm-cloud-platform-match-made-heaven/.
21. Nagios. *Wikipedia.* [Ηλεκτρονικό] http://en.wikipedia.org/wiki/Nagios.
22. The Client/Server Model. [Ηλεκτρονικό] http://samuraii.net/writings/notes_networksintro.htm.
23. OpenStack: The Open Source Cloud Operating System. *openstack.* [Ηλεκτρονικό] http://www.openstack.org/software/.

24. Chapter 1. Architecture. *openstack.* [Ηλεκτρονικό] http://docs.openstack.org/juno/install-guide/install/apt/content/ch_overview.html.
25. How to Create RESTful Java Client With Jersey Client – Example. *crunchify.* [Ηλεκτρονικό] http://crunchify.com/how-to-create-restful-java-client-with-jersey-client-example/.
26. Spring Batch as Wildfly Module. *Java Code Geeks.* [Ηλεκτρονικό] http://www.javacodegeeks.com/2014/08/spring-batch-as-wildfly-module.html.
27. Ingredients of OpenStack API. *Destiny to The Cloud.* [Ηλεκτρονικό] http://cloudn1n3.blogspot.gr/2014/10/ingredients-of-openstack-api.html.
28. JavaScript Object Notation (JSON). *Service Architecture.* [Ηλεκτρονικό] http://www.service-architecture.com/articles/web-services/javascript_object_notation_json.html.