Ατομική Διπλωματική Εργασία

ΣΧΕΔΙΑΣΜΌΣ ΓΕΝΙΚΗΣ ΔΙΕΠΑΦΗΣ ΓΙΑ ΕΓΚΑΤΑΣΤΑΣΗ ΠΕΡΙΓΡΑΦΩΝ ΕΦΑΡΜΟΓΩΝ ΣΕ ΠΕΡΙΒΑΛΛΟΝΤΑ ΝΕΦΕΛΗΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΔΙΕΠΑΦΗΣ ΓΙΑ OPENSTACK

Ανδρέας Καστανάς

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδιασμός γενικής διεπαφής για εγκατάσταση περιγραφών εφαρμογών σε περιβάλλοντα νεφέλης και υλοποίηση διεπαφής για OpenStack

Ανδρέας Καστανάς

Επιβλέπων Καθηγητής
Μάριος Δ. Δικαιάκος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2015

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον καθηγητή κύριο Μάριο Δικαιάκο ο οποίος μου προσέφερε το συγκεκριμένο θέμα για την εκπόνηση της Διπλωματικής μου εργασίας και τις γνώσεις που αποκόμισα μέσω αυτής σε τομείς που δεν είχα τόση εμπειρία. Καθώς επίσης και τον καθηγητή κύριο Γιώργο Παλλή για την πολύτιμη υποστήριξή του.

Επίσης, θα ήθελα να ευχαριστήσω τον διδακτορικό φοιτητή Δημήτρη Τριχινά για την βοήθεια του και την καθοδήγηση στους κρίσιμους τομείς και στις δυσκολίες που συνάντησα κατά την διάρκεια του όλου εγχειρήματος. Επίσης, τους διδακτορικούς φοιτητές Στάλω Σοφοκλέους και Νικόλα Λουλλούδη για την πολύτιμη βοήθειά τους.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την πολύτιμη υποστήριξη τους σε όλους τους τομείς τόσο ψυχολογικά όσο και οικονομικά με σκοπό να καταφέρω να ολοκληρώσω επιτυχώς τις σπουδές μου.

Περίληψη

Η παρούσα διπλωματική εκπονήθηκε με σκοπό τη σχεδίαση και την ανάπτυξη μιας διεπαφής για εγκατάσταση περιγραφών εφαρμογών στο OpenStack. Αρχικά, έγινε έρευνα για το τι είναι ο όρος Cloud Computing, ποια είναι τα χαρακτηριστικά του, τις επιμέρους κατηγορίες και τα μοντέλα του καθώς το τι προσφέρει και τι το διαφοροποιεί από τις υπάρχουσες δομές. Στη συνέχεια, ορίστηκε ένα API για το ποιες λειτουργίες πρέπει να προσφέρει ο Deployer και ποια πρέπει να είναι τα χαρακτηριστικά του. Για τον σκοπό αυτό, κατασκευάστηκε αρχικά μία Standalone εφαρμογή για να διαπιστωθεί η λειτουργικότητά του και το κατά πόσο όσα ορίστηκαν στο στάδιο της σχεδίασης ήταν εφικτά. Ύστερα, έπρεπε η συγκεκριμένη υλοποίηση να μεταφερθεί στο περιβάλλον του CAMF. Να προστεθούν κάποια νέα χαρακτηριστικά και λειτουργίες που δεν υπήρχαν στην Standalone έκδοση με σκοπό να προσφέρει ακόμη περισσότερες δυνατότητες στον χρήστη και να τον βοηθήσει στο έργο του. Έπρεπε να δοθεί έμφαση στο να μην χαθεί κάποια από τις υπάρχουσες λειτουργικότητες που προσφέρει το CAMF καθώς και να είναι σε θέση να υποστηρίξει το ίδιο σύνολο ενεργειών. Αφού ολοκληρώθηκε το πρώτο στάδιο της ενσωμάτωσης, άρχισαν οι βελτιώσεις όπως παρουσιάζονται στο 3^{ο} κεφάλαιο. Μερικές από αυτές επιγραμματικά είναι η διαχείριση λαθών κατά την διαδικασία της ανάπτυξης κάποιας εφαρμογής και η αξιοποίηση του παραγόμενου JSON αρχείου που περιέχει κάποιες μετά-πληροφορίες. Μετά την ολοκλήρωση του σταδίου αυτού, έγινε κάποια αξιολόγηση της εφαρμογής με κάποιες τυχαίες περιγραφές με σκοπό να διαπιστωθεί αν το παραγόμενο αποτέλεσμα ήταν το αναμενόμενο στις διάφορες καταστάσεις που προκύπτουν. Τέλος, καθ' όλη τη διαδικασία της ανάπτυξης του Deployer, υπήρξε η πρόνοια να έχει δυνατότητες επέκτασης ως προς την λειτουργία του. Μία από αυτές είναι η υποστήριξη ελαστικών απαιτήσεων.

Ανδρέας Καστανάς – Πανεπιστήμιο Κύπρου, Μάιος 2015

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Ο όρος *Cloud Computing* χρησιμοποιείται όλο και πιο πολύ στις μέρες μας τόσο από τους καθημερινούς χρήστες του διαδικτύου όσο και από μικρές ή μεγάλες εταιρείες με σκοπό την αύξηση της αποδοτικότητά τους και την ελαχιστοποίηση του κόστους λειτουργίας της υπολογιστικής δύναμης που διαθέτουν.

Όπως παρουσιάζεται και στα υπόλοιπα κεφάλαια, ο όρος αυτός, είναι πολύ γενικός ως προς το περιεχόμενο του και τις λειτουργίες του. Δύο από τα βασικά χαρακτηριστικά του που έχουν συνδεθεί άρρηκτα με το όνομα του είναι η πρόσβαση από οποιαδήποτε συσκευή στης υπηρεσίες που προσφέρονται μέσω αυτού και το μοντέλο που χρησιμοποιεί *pay-as-you-use* το οποίο προσφέρει πρόσβαση σε μία εικονική υπολογιστική δύναμη και η χρέωση των υπηρεσιών αυτών γίνονται ανάλογα με τη χρήση τους. Επιπλέον, το Cloud, αποτελείται από τρία είδη διαφορετικών μοντέλων που το καθένα προσφέρεται και για διαφορετικούς σκοπούς το Public, Private και Hybrid. Καθώς επίσης, οι υπηρεσίες που προσφέρει χωρίζονται σε τρεις βασικές κατηγορίες που μπορούν να έχουν την μορφή μιας πυραμίδας (σχ. 2.2). Στην βάση της βρίσκουμε το Infrastructure-as-a-Service, ανεβαίνοντας το Platform-as-a-Service και στην κορυφή της, το Software-as-a-Service. Όπου η κάθε μία από αυτές, προσφέρει διαφορετικά είδη υπηρεσιών που απευθύνονται σε διαφορετικό κοινό. Ο μέσος χρήστης εμπλέκεται κυρίως με το Software-as-a-Service όπου προσφέρει κάποιες έτοιμες προς χρήση εφαρμογές. Εμείς, θα ασχοληθούμε με τη βάση αυτών των τριών κατηγοριών το Infrastructure-as-a-Service, όπου δίνεται πρόσβαση σε μία εικονική υπολογιστική νεφέλη για

περαιτέρω αξιοποίηση από τον χρήστη με σκοπό την ανάπτυξη των εφαρμογών του και πιο συγκεκριμένα με την υποδομή που προσφέρει το OpenStack.

Πιο συγκεκριμένα, για να είναι ο χρήστης σε θέση να προβεί σε αυτές τις ενέργειες και για σκοπό ευκολίας, υπάρχουν τα λεγόμενα *Application Management Framework* εργαλεία, όπου ο χρήστης τα αξιοποιεί με σκοπό να περιγράψει την εφαρμογή που θέλει. Τα εργαλεία αυτά, συλλέγουν όλα τα απαραίτητα συστατικά που χρειάζεται για τη λειτουργία της εφαρμογής όπως είναι τα Virtual Images, script για εκτέλεση κλπ. και τα αποστέλλουν στον πάροχο για ανάπτυξη. Μέσα στο κείμενο αυτό, τα AMF, έχουν χωριστεί σε δύο διακριτές κατηγορίες για σκοπούς σύγκρισης και υποβολής συμπερασμάτων. Τα (i) εργαλεία εμπορικής χρήσης και τα (ii) ακαδημαϊκά εργαλεία. Πολλά από αυτά προσφέρουν διάφορες λειτουργίες που θα μπορούσαν να ικανοποιήσουν τον μέσο χρήστη όπως προβάλλονται στο $2^{ο}$ κεφάλαιο. Συγχρόνως όλα τα υπάρχων εργαλεία έχουν ένα κοινό σημαντικό μειονέκτημα - πέρα από τις υπόλοιπες διαφορές που έχουν μεταξύ τους - και αυτό είναι η φορητότητα μεταξύ των παρόχων. Κανένα από τα ήδη υπάρχων εργαλεία δεν προσφέρει αυτή τη δυνατότητα με σκοπό να απαιτείται από τον χρήστη να προβεί σε περαιτέρω ενέργειες όταν θέλει να αλλάξει πάροχο. Σε αυτό το σημείο, έρχεται το CAMF να διορθώσει το πρόβλημα αυτό. Το ίδιο το εργαλείο, προσφέρει τη δυνατότητα επέκτασης του με σκοπό να επιτρέψει στον χρήστη του να το προσαρμόσει στις ανάγκες του. Μία από αυτές τις επεκτάσεις είναι, η χρήση κάποιας γενικής βιβλιοθήκης που προσφέρει την άμεση επικοινωνία με τους παρόχους βγάζοντας από τη μέση το στάδιο της αποστολής του CSAR αρχείου που περιέχει τα ζωτικής σημασίας αρχεία για την ανάπτυξη, με σκοπό την επεξεργασία του από τον πάροχο και την ανάπτυξη της εφαρμογής. Μια διαδικασία που απαιτείται μέχρι στιγμής να γίνεται από όλα τα διαθέσιμα εργαλεία. Με τον τρόπο αυτό, καθιστούμε το CAMF ικανό να λειτουργεί ανεξάρτητα του παρόχου και να είναι σε θέση να στείλει την ίδια περιγραφή σε διαφορετικούς παρόχους με ελάχιστες ή καθόλου διορθώσεις από την πλευρά του χρήστη.

1.2 Συνεισφορά

Σε αυτή την ΑΔΕ επιλύουμε το πρόβλημα που αναφέρθηκε πιο πάνω με την χρήση της εξωτερικής βιβλιοθήκης jClouds που προσφέρει την επικοινωνία με ένα πλήθος παρόχων. Έχουμε κατασκευάσει έναν Deployer για το OpenStack ο οποίος προσφέρει τις ακόλουθες λειτουργίες:

- Ανάπτυξη οποιασδήποτε περιγραφής ανεξαρτήτου βάθους και αριθμού των Components
- Χρήση πολιτικών ανάπτυξης σε περίπτωση σφάλματος
- Δημιουργία μετά-πληροφοριών για περαιτέρω αξιοποίηση τόσο από τον χρήστη όσο και από την ίδια την εφαρμογή
- Μηνύματα σφαλμάτων τα οποία είναι φιλικά προς τον χρήστη χωρίς να χάνεται το αρχικό σφάλμα για σκοπούς αποσφαλμάτωσης
- Δυνατότητα χρήσης τόσο της υπάρχουσας λειτουργικότητας ως προς την ανάπτυξη όσο και της νέας

1.3 Δομή εγγράφου

Τα υπόλοιπα κεφάλαια ακολουθούν την εξής δομή.
Στο 2^{ο} κεφάλαιο προβάλλεται μία μελέτη του Cloud, των υπηρεσιών του, των μοντέλων του και των λειτουργιών του.
Στο 3^{ο} κεφάλαιο η σχεδίαση που έγινε για την ανάπτυξη του Deployer καθώς και η ανάπτυξη αυτή καθ’ αυτή.
Στο 4^{ο} κεφάλαιο γίνεται μία αξιολόγηση της λειτουργικότητας του.
Τέλος, στο 5^{ο} κεφάλαιο προβάλλονται τα συμπεράσματα και η μελλοντική δουλειά που μπορεί να γίνει για την περαιτέρω ανάπτυξή του.

Κεφάλαιο 2

Background and Related Work

Στο συγκεκριμένο κεφάλαιο, γίνεται μια λεπτομερής περιγραφή του όρου Cloud Computing καθώς και των επί μέρους υπηρεσιών που περιβάλλει, αλλά και παραδείγματα που χρησιμοποιούνται καθημερινά από όλους τους χρήστες. Επίσης, αναλύονται τα τρία μοντέλα του Cloud και οι συνηθέστεροι λόγοι για τους οποίους χρησιμοποιούνται. Επιπλέον, αναφέρεται η έννοια του Application Management Framework (AMF) καθώς και τα κριτήρια που πρέπει να πληροί ένα AMF για να είναι λειτουργικό. Τα AMF έχουν χωριστεί, σε δύο βασικές κατηγορίες: (i) εμπορικά, και (ii) ακαδημαϊκά. Τέλος, γίνεται εκτενής αναφορά στο CAMF και τις λειτουργίες αυτές που το διαφοροποιούν τόσο από τα αντίστοιχα εμπορικά όσο και από τα ακαδημαϊκά προϊόντα καθώς και τις δυνατότητες επέκτασης του ώστε αυτό να μπορεί να βελτιστοποιηθεί.

2.1 Cloud Computing

Το Cloud Computing είναι ένας γενικός όρος που περιλαμβάνει, μεταξύ άλλων, την παροχή υπηρεσιών που φιλοξενούνται στο διαδίκτυο και επιτρέπει στον χρήστη να καταναλώνει υπολογιστικούς πόρους σαν υπηρεσία, εν' αντιθέσει με το να πρέπει να δημιουργήσει και να συντηρεί μία υπολογιστική υποδομή στο χώρο του. Υπάρχουν πολλοί ορισμοί αλλά ο πιο κοινά αποδεκτός είναι αυτός που ορίστηκε από το National Institute of Standards and Technology (NIST)[1], και συγκεκριμένα, αναφέρει ότι είναι "*ένα μοντέλο που καθιστά εφικτή*

[1] http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

την πρόσβαση σε ένα δίκτυο από κοινόχρηστους υπολογιστικούς πόρους (δίκτυα, server, αποθήκευση, εφαρμογές, υπηρεσίες) που μπορούν γρήγορα να τεθούν σε λειτουργία με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδρασης με τον πάροχο."

Μερικά από τα διακριτά χαρακτηριστικά του, που το διαφοροποιούν από το παραδοσιακό hosting:

- Πρόσβαση από οπουδήποτε, οποιαδήποτε στιγμή και από οποιαδήποτε συσκευή που έχει διαδικτυακή πρόσβαση (πχ. κινητό, υπολογιστής, κ.α.)
- Κοστολογείται με βάση τη ζήτηση (πχ. συνήθως με το λεπτό ή την ώρα) και το επιχειρηματικό μοντέλο αυτό αναφέρεται συχνά ως "pay-as-you-use"
- Ελαστικότητα που σημαίνει ότι ένας χρήστης μπορεί να έχει όσο πολύ ή όσο λίγο χρειάζεται μια υπηρεσία μία δεδομένη χρονική στιγμή
- Η υπηρεσία τυγχάνει διαχείρισης από τον πάροχο

Σχήμα 2.1 Χαρακτηριστικά του Cloud

2.1.1 Μοντέλα του Cloud

Το Cloud μπορεί να είναι είτε Private είτε Public καθώς και Hybrid.

Ένα Public Cloud παρέχει υπηρεσίες σε όλους στο Internet. Προς το παρόν, το Amazon Web Services[2] είναι το μεγαλύτερο και το πιο διαδεδομένο ως προς τη χρήση και τις υπηρεσίες που προσφέρει.

Ένα Private Cloud είναι ένα ιδιόκτητο δίκτυο ή datacenter που παρέχει τις υπηρεσίες του σε περιορισμένο αριθμό ατόμων.

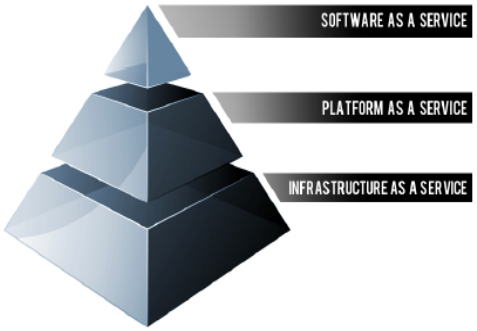
[2] http://www.geekwire.com/2015/aws-remains-top-dog-in-cloud-survey-but-microsoft-azure-gains-traction/

Στο Hybrid, γίνεται η χρήση τόσο μιας εσωτερικής (private cloud) δομής όσο και μια εξωτερικής (public cloud). Για παράδειγμα κάποια ευαίσθητα δεδομένα που ενδεχομένως να έχει μία επιχείρηση μπορεί να βρίσκονται στο εσωτερικό της δίκτυο και κάποια άλλα στο Public. Ανεξαρτήτως του είδους του cloud, στόχος είναι η παροχή εύκολης και επεκτάσιμης πρόσβασης σε υπολογιστικούς πόρους και υπηρεσίες διαδικτύου.

2.1.2 κατηγορίες του Cloud

Οι υπηρεσίες που προσφέρει το Cloud, χωρίζονται σε τρεις διακριτές κατηγορίες:

- Software-as-a-Service (SaaS)
- Platform-as-a-Service (PaaS)
- Infrastructure-as-a-Service (IaaS)



Σχήμα 2.2 Κατηγορίες του Cloud

SaaS: Ο πάροχος παρέχει την δομή τόσο σε υλικό όσο και σε λογισμικό. Ο χρήστης αλληλεπιδρά μαζί του συνήθως μέσω κάποιας ιστοσελίδας διαχείρισης ή κάποιας εφαρμογής. Επίσης, δεν έχει κανέναν έλεγχο στην υποδομή ή στα εργαλεία ανάπτυξης. Η χρήση του μπορεί να γίνει για πολλούς λόγους. Μερικά παραδείγματα είναι τα Google Docs[3], Gmail[4], QuickBooks Online[5] κλπ. Επειδή τόσο η δομή όσο και τα δεδομένα είναι στη πλευρά του παρόχου, ο χρήστης μπορεί να αλληλεπιδράσει με την εφαρμογή από οποιοδήποτε σημείο.

PaaS: Είναι ένα σύνολο εργαλείων για την ανάπτυξη λογισμικού το οποίο βρίσκεται στη πλευρά του παρόχου τα οποία χρησιμοποιούνται από τους προγραμματιστές για την ανάπτυξη των εφαρμογών τους. Τέτοιο παράδειγμα εργαλείων είναι τα Google Apps[6], Openshift[7], Heroku[8], CloudFountry[9]. Υπάρχουν κάποιοι περιορισμοί ως προς την χρήση τους από κάποιους παρόχους οι οποίοι δεν επιτρέπουν στον χρήστη να μεταφέρει εκτός της πλατφόρμας τους την εφαρμογή του. Πάλι, ο χρήστης δεν έχει κάποιον έλεγχο στην εικονική

[3] https://docs.google.com/
[4] https://mail.google.com
[5] http://quickbooks.intuit.com/
[6] https://www.google.com/work/apps/business/
[7] https://www.openshift.com/
[8] https://www.heroku.com/
[9] http://cloudfoundry.org/index.html

υποδομή παρά μόνο σε μερικές περιπτώσεις έχει έλεγχο στις εφαρμογές που αναπτύσσονται και στις ρυθμίσεις μεταξύ των εφαρμογών και του περιβάλλοντος.

IaaS: Είναι η δομή που χρησιμοποιεί και το Amazon. Προσφέρει στους χρήστες ένα virtual server και χώρο αποθήκευσης καθώς και κάποια APIs που επιτρέπουν στον χρήστη την επικοινωνία και την διαχείριση αυτών. Η συγκεκριμένη δομή, δίνει τη δυνατότητα της πληρωμής ανάλογα με τη χρήση και να είναι διαθέσιμη ανά πάσα χρονική στιγμή. Λόγω του ότι είναι παρόμοιο με τα μοντέλα παροχής υπηρεσιών (ρεύμα, νερό) χρησιμοποιείται και ο όρος Utility Computing για τη περιγραφή του. Όπως και στις προηγούμενες περιπτώσεις ο χρήστης δεν έχει κάποιο έλεγχο στο υλικό, παρά μόνο στο λειτουργικό σύστημα, την αποθήκευση, τις εγκατεστημένες εφαρμογές και πιθανόν σε ρυθμίσεις δικτύου όπως το firewall.

2.2 Cloud Application Management Tools and Services

Η περιγραφή και η ανάπτυξη μιας περιγραφής στο Cloud, τις πιο πολλές φορές μπορεί να είναι μια πολύπλοκη και χρονοβόρα διαδικασία. Επίσης, ο κάθε πάροχος, έχει καθορισμένα δικά του πρότυπα και APIs, πράγμα που απαιτεί ακόμη περισσότερη δουλειά από τον χρήστη σε περίπτωση που επιθυμεί την μεταφορά μεταξύ διαφόρων παρόχων. Για αυτό το λόγο, υπάρχουν τα Application Management Framework (AMF), τα οποία είναι σε θέση να συγκεντρώνουν τις περιγραφές του χρήστη, τα τυχόν script και ότι άλλο χρειάζεται για να αναπτυχθεί η περιγραφή του στο Cloud, να οργανώνουν τα απαραίτητα συστατικά και να τα αποστέλλουν στον πάροχο με σκοπό την τελική ανάπτυξη. Με αποτέλεσμα ο χρήστης να μην απασχολείται τόσο με αυτό το κομμάτι και τις αλλαγές που πρέπει να κάνει από πάροχο σε πάροχο, αλλά να επικεντρώνεται κυρίως στην εφαρμογή του και τις βελτιώσεις της. Παρόλα αυτά, δεν σημαίνει ότι οι υπάρχουσες εφαρμογές λύνουν όλα τα προβλήματα που αντιμετωπίζει ο κάθε χρήστης.

Συγκεκριμένα, τα υπάρχοντα AMFs υποστηρίζουν μόνο συγκεκριμένους παρόχους και έχουν έλλειψη σημαντικών χαρακτηριστικών όπως είναι η ελαστικότητα (Elasticity), η παρακολούθηση πόρων (monitoring) και άλλες σημαντικές λειτουργίες που θα παρουσιαστούν πιο κάτω.

Κάποια από τα βασικά χαρακτηριστικά που πρέπει να διέπουν αυτές τις εφαρμογές ως προς την λειτουργία τους, είναι τα ακόλουθα:

- Καθορισμό πολιτικών χρήσης πόρων
- Επεκτασιμότητα

- Ευκολία στη χρήση
- Παρακολούθηση πόρων
- Ανεξάρτητα Cloud παρόχου
- Λειτουργία ανεξάρτητα του Λ/Σ
- Αυτόματη ανάπτυξη στο σύννεφο

Οι εφαρμογές αυτές χωρίζονται σε δύο κατηγορίες για σκοπούς σύγκρισης των χαρακτηριστικών τους. Σε εμπορικές (Production Ready) και σε ακαδημαϊκές (Academic). Πιο κάτω παρουσιάζονται οι πιο γνωστές για κάθε κατηγορία με τα πλεονεκτήματα και τα μειονεκτήματά τους.

2.2.1 Production Ready Tools

Οι τρεις πιο διαδεδομένες εμπορικές εφαρμογές τύπου AMF είναι οι:

- Juju [10]
- Bluemix [11]
- ServiceMesh [12]

Όπου στον παρακάτω πίνακα παρουσιάζονται συνοπτικά κάποια από τα χαρακτηριστικά τους και οι βασικές τους διαφορές.

			
Easy of Use	✓	✓	✓
Elastic	✗	✓	✗
Monitoring	✓	✗	✓

[10] https://jujucharms.com/
[11] https://console.ng.bluemix.net/
[12] http://www.servicemesh.com/

Cloud Vendor Independent	✗	✗	✗
User OS Independent	✓	✓	✓
Auto Deployment	✓	✓	✓
Scalable	✓	✓	✓
Import-Export Option	✓	✓	✗
Frameworks	✓	✓	✓
Services	✓	✓	✓
GUI	✓	✓	✓
CMD	✓	✓	✓
Application Limitations • **RAM** • **CPU** • **Monitoring** • **Auto Caching**	✗	✓	✗
Git	✗	✓	✗

Πίνακας 2.1 Χαρακτηριστικά εμπορικών AMF

Όπως βλέπουμε και τα τρία εργαλεία υποστηρίζουν ένα από τα βασικά χαρακτηριστικά αυτό της ευκολίας στην χρήση. Δίνοντας στον χρήστη ένα απλό και φιλικό περιβάλλον που στις περισσότερες περιπτώσεις είναι εύκολα κατανοητό. Επιπλέον δίνουν στους πιο προχωρημένους χρήστες τη δυνατότητα πρόσβασης σε αυτά μέσω κάποιου Command Prompt για πιο προχωρημένες λειτουργίες. Από την άλλη, και τα τρία εργαλεία, **δεν υποστηρίζουν την μεταφορά μεταξύ διαφόρων παρόχων**. Δουλεύουν μόνο με συγκεκριμένους παρόχους και η μεταφορά μεταξύ αυτών απαιτεί από τον χρήστη αλλαγές

στην περιγραφή του. Το Juju και το ServiceMesh δίνουν την μεγαλύτερη ευκινησία με τους παρόχους που υποστηρίζουν να είναι οι *(VMware, Microsoft Azure, Amazon EC2, CSC, OpenStack, CloudStack, RackSpace and Eucalyptus.).*

Επίσης και τα τρία εργαλεία, έχουν τη δυνατότητα να τρέξουν σε οποιαδήποτε Λ/Σ που έχει ο χρήστης με την μόνη προϋπόθεση τη σύνδεση στο Διαδίκτυο. Είναι σε θέση μόλις τους δοθεί κάποια περιγραφή να την αναπτύξουν αυτόματα στον πάροχο -εφόσον τον υποστηρίζουν- χωρίς να απαιτούν από τον χρήστη περαιτέρω ενέργειες. Σημαντικό επίσης πλεονέκτημα είναι ότι προσφέρουν στο χρήστη κάποια έτοιμα (ready-to-use) plug-ins και υπηρεσίες που χρησιμοποιούνται συχνά όπως (Wordpress, MySQL Server) για να τα χρησιμοποιήσει με άμεσες κινήσεις με σκοπό την εξοικονόμηση χρόνου και την εξάλειψη της πολυπλοκότητας. Το Bluemix, είναι σε θέση να προσφέρει ελαστικές απαιτήσεις στις ανάγκες του χρήστη με σκοπό την ορθή κατανομή των πόρων, καθώς και πρόσβαση σε κάποιο Git Repository για άμεση πρόσβαση στον κώδικα του χρήστη και αυτόματη ενσωμάτωση των αλλαγών του με σκοπό να μην επιβαρύνεται ο χρήστης με το επιπλέον αυτό βήμα. Αντίθετα το Juju και το ServiceMesh δεν είναι σε θέση να προσφέρουν αυτές τις δύο δυνατότητες. Επιπλέον, τα δύο αυτά (Juju & ServiceMesh) δίνουν στον χρήστη τη δυνατότητα παρακολούθησης των πόρων που καταναλώνουν οι εφαρμογές του έτσι ώστε να προβεί σε ενέργειες βελτιστοποίησής τους, σε αντίθεση με το Bleumix που δεν προσφέρει τη δυνατότητα αυτή, αλλά συγχρόνως, περιορίζει τον χρήστη ως προς την χρήση των πόρων που μπορεί να χρησιμοποιήσει στις εφαρμογές του χωρίς επίσης να του παρέχει κάποιες θεμελιώδεις λειτουργίες όπως είναι το File Caching και το Local Write.

Επιπλέον, το Juju χρησιμοποιεί τα λεγόμενα *charms* που περιλαμβάνουν τις ρυθμίσεις και τη περιγραφή της εφαρμογής καθώς και επιπλέον πληροφορίες. Σημαντικό μειονέκτημά τους είναι ότι αναφέρονται συνήθως σε Linux εφαρμογές με αποτέλεσμα να περιορίζεται σημαντικά η φορητότητά τους σε διαφορετικά Λ/Σ, με αποτέλεσμα σε περίπτωση αλλαγής Λ/Σ της περιγραφής να χρειάζονται τυχόν επιπλέον αλλαγές στις εφαρμογές του χρήστη. Τέλος, το ServiceMesh δίνει τη δυνατότητα της δυναμικής διαχείρισης του κύκλου-ζωής της εφαρμογής επιτρέποντας τον ορισμό κανόνων σχετικά με την προσθήκη και την αφαίρεση VM αλλά συγχρόνως το κόστος χρήσης του, παραμένει αρκετά υψηλό σε σχέση με τις άλλες δύο επιλογές.

Αυτά είναι τα κύρια χαρακτηριστικά και οι διαφορές των πιο διαδεδομένων εμπορικών AMF. Είναι σε θέση να καλύψουν ένα μεγάλο εύρος από τις ανάγκες του χρήστη. Διατίθενται βάση κάποιου χρηματικού αντίτιμου και είναι άμεσα διαθέσιμα προς χρήση.

2.2.2 Academic Tools

Από ακαδημαϊκής πλευράς έχουν γίνει κάποια σημαντικά βήματα στον χώρο αυτό με σκοπό την επίλυση κάποιων αδυναμιών που διέπουν τις εμπορικές εκδόσεις και τη προσθήκη νέων δυνατοτήτων.

Πιο συγκεκριμένα τρία αρκετά διαδεδομένα ακαδημαϊκά AMF είναι τα:

- CAMF (c-Eclipse) [4]
- Wrangler [3]
- Winery [2]

Παρακάτω, παρουσιάζονται τα βασικά χαρακτηριστικά τους και οι διαφορές τους για σκοπούς σύγκρισης.

	CAMF	**Wrangler**	Winery
Easy of Use	✓	✗	✓
Elastic	✓	✓	✗
Monitoring	✓	✓	✗
Cloud Vendor Independent	✓	✓	✓
User OS Independent	✓	✓	✓
Auto Deployment	✓	✓	✗
Scalable	✓	✓	✗
Import-Export Option	✓	✗	✓
Application Management Modeling	✓	✗	✗

Πίνακας 2.2 Χαρακτηριστικά ακαδημαϊκών AMF

Τόσο το CAMF όσο και το Winery είναι εύκολα στη χρήση για τον χρήστη με ένα άκρως φιλικό περιβάλλον και αρκετά επεξηγηματικό χωρίς την απαίτηση ειδικών γνώσεων σε αντίθεση με το Wrangler που απαιτεί από τον χρήστη τη γνώση της δικής του XML γλώσσας που χρησιμοποιεί για να γίνει περιγραφή πολύπλοκων εξαρτήσεων μεταξύ των οντοτήτων. Το Winery με σκοπό να προωθήσει την φορητότητα μεταξύ των παρόχων, κάνει χρήση του μοντέλου TOSCA σε συνδυασμό με το περιβάλλον HTML5, επιτρέποντας στο χρήστη να δημιουργήσει στοιχεία TOSCA ή να επεξεργάζεται τα υπάρχοντα μέσω ενός περιβάλλοντος που μπορεί να τρέξει σε οποιονδήποτε Browser που υποστηρίζει HTML5. Επίσης, δεν επιτρέπει στον χρήστη τον προσδιορισμό ελαστικών απαιτήσεων με σκοπό την επεκτασιμότητα και τη διαχείριση των πόρων της εφαρμογής. Επιπροσθέτως, δεν παρέχει τη δυνατότητα παρακολούθησης των πόρων και δεν έχει τη δυνατότητα της αυτόματης ανάπτυξης της περιγραφής στον πάροχο. Τόσο το Winery όσο και το Wrangler δεν χρησιμοποιούν κάποιο μετά-μοντέλο του TOSCA για να μοντελοποιήσουν την περιγραφή που έχει δώσει ο χρήστης.

2.3 OASIS TOSCA

Το πρότυπο TOSCA [13], όπως λέει και το όνομα του, (*Topology and Orchestration Specification for Cloud Applications*) χρησιμοποιείται για την περιγραφή της τοπολογίας Cloud εφαρμογών (*service topology*) και της ενορχήστρωσης των παραγόμενων υπηρεσιών (*orchestration processes*). Σκοπός της είναι η βελτίωση της φορητότητας των περιγραφών του χρήστη μεταξύ των διαφορετικών παρόχων στο Cloud. Επιτρέπει την περιγραφή των εφαρμογών και των υποδομών τους, τις σχέσεις μεταξύ των οντοτήτων της περιγραφής καθώς και την συμπεριφορά τους κατά τη διαδικασία της ανάπτυξης. Επιπλέον, είναι σε θέση να προσφέρει υψηλού επιπέδου λειτουργίες που σχετίζονται με τη διαχείριση της υποδομής του Cloud. Η χρήση του TOSCA επιτρέπει τη:

- Φορητότητα της περιγραφής μεταξύ των υποστηριζόμενων παρόχων
- Διαχείριση του κύκλου ζωής της εφαρμογής (scaling, monitoring)

[13] https://www.oasis-open.org/committees/tosca

2.4 Cloud Application Management Framework (CAMF)

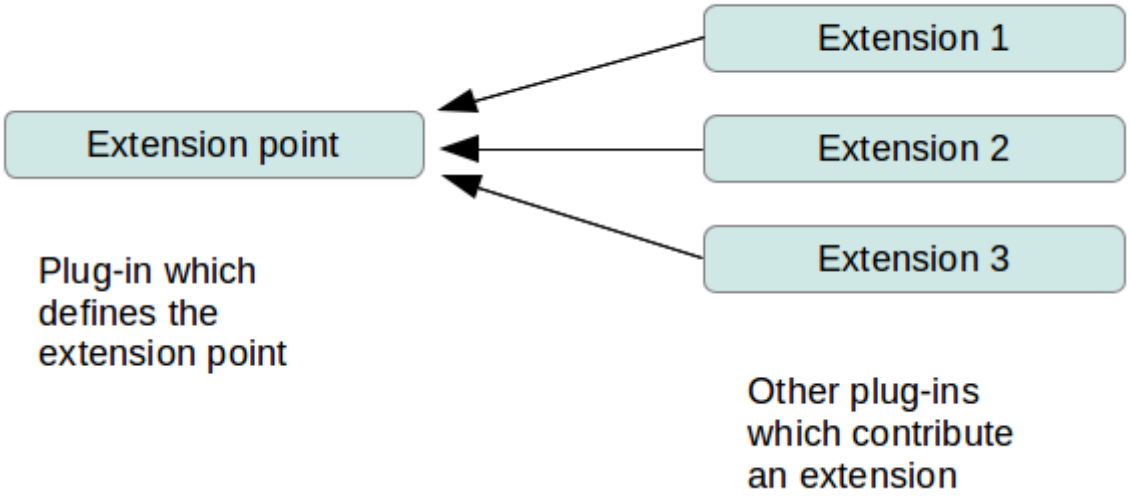
Το CAMF [4], ενσωματώνει και υλοποιεί όλα τα προηγούμενα χαρακτηριστικά που τα δύο παραπάνω εργαλεία δεν είναι σε θέση να υποστηρίξουν ή υποστηρίζονται μερικώς. Πιο συγκεκριμένα, χρησιμοποιεί ένα φιλικό περιβάλλον προς τον χρήστη που μέσω μίας κεντρικής παλέτας που περιέχει τα απαραίτητα συστατικά, όπως είναι οι εικόνες (virtual images), οι σχέσεις μεταξύ των Components, τα διαθέσιμα Key pairs κλπ., του επιτρέπει με εύκολες και άμεσες κινήσεις (drag'n'drop) να χρησιμοποιήσει τα συστατικά που χρειάζεται για την εφαρμογή του, κρύβοντας έτσι την πολυπλοκότητα που δημιουργείται στο επίπεδο του TOSCA. Επίσης είναι το μοναδικό εργαλείο από όλα που προσθέτει επιπλέον οντότητες στο TOSCA μοντέλο με σκοπό τον προσδιορισμό κανόνων για την ελαστικότητα και την επεκτασιμότητα της εφαρμογής. Σε αντίθεση με το Winery που χρησιμοποιεί το μοντέλο BPEL για να μοντελοποιήσει την εφαρμογή, το CAMF κάνει χρήση του TOSCA μετά-μοντέλου με σκοπό τη διαχείριση της περιγραφής και την αυτόματη ανάπτυξή της. Σημαντικό πλεονέκτημα, είναι ότι για όλο τον κύκλο ζωής του CAMF, το μετά-μοντέλο της περιγραφής που ορίστηκε παραμένει φορτωμένο στην μνήμη και γίνεται εύκολα αξιοποιήσιμο. Επιπλέον, παρέχει τη δυνατότητα Monitoring [5], επιτρέποντας στον χρήστη να παρακολουθεί τους πόρους που καταναλώνει η εφαρμογή του με σκοπό την επέκτασή της βάση των ελαστικών κανόνων που έχει ορίσει. Με τους κανόνες αυτούς, ο χρήστης ορίζει τι να γίνεται σε κάθε Virtual Instance όταν ικανοποιηθεί μια συνθήκη. Όπως για παράδειγμα αν η χρήση του CPU φτάσει στο 85% δημιούργησε ένα επιπλέον παρόμοιο Instance ή αν κάποιο Instance υπό-χρησιμοποιείται, διέγραψέ το. Επίσης, είναι από τα λίγα εργαλεία που είναι ανεξάρτητα παρόχου επειδή κάνει χρήση του προτύπου OpenTOSCA[14] που επιτρέπει στον χρήστη την περιγραφή της εφαρμογής του με έναν πολύ γενικό τρόπο και ο οποίος υποστηρίζεται από διάφορους παρόχους. Έτσι η φορητότητα μιας εφαρμογής γίνεται αρκετά εύκολη σε μικρό χρονικό διάστημα. Επιτρέπει επίσης την επεκτασιμότητα του ίδιου του εργαλείου μέσω διαφόρων Interface επιτρέποντας στον χρήστη να το προσαρμόσει στις ανάγκες του. Για παράδειγμα δίνει τη δυνατότητα να υλοποιηθούν συγκεκριμένες ενέργειες που ο χρήστης χρειάζεται καθώς και επιπλέον πιο εξειδικευμένοι μηχανισμοί Monitoring. Τέλος, είναι υπεύθυνο για τη συγκέντρωση όλων τον απαραίτητων συστατικών που χρειάζονται για την ανάπτυξη της εφαρμογής όπως το Image του Virtual Instance, τα τυχόν script εκτέλεσης που έχει ορίσει ο χρήστης, τα key pairs κλπ. και τα αποθηκεύει όλα σε ένα αρχείο τύπου *Cloud Service Archive* (CSAR) το οποίο στέλνεται στον πάροχο. Βασική προϋπόθεση είναι ο πάροχος να υποστηρίζει το πρότυπο OpenTOSCA και να προσφέρει

[14] http://www.iaas.uni-stuttgart.de/OpenTOSCA/

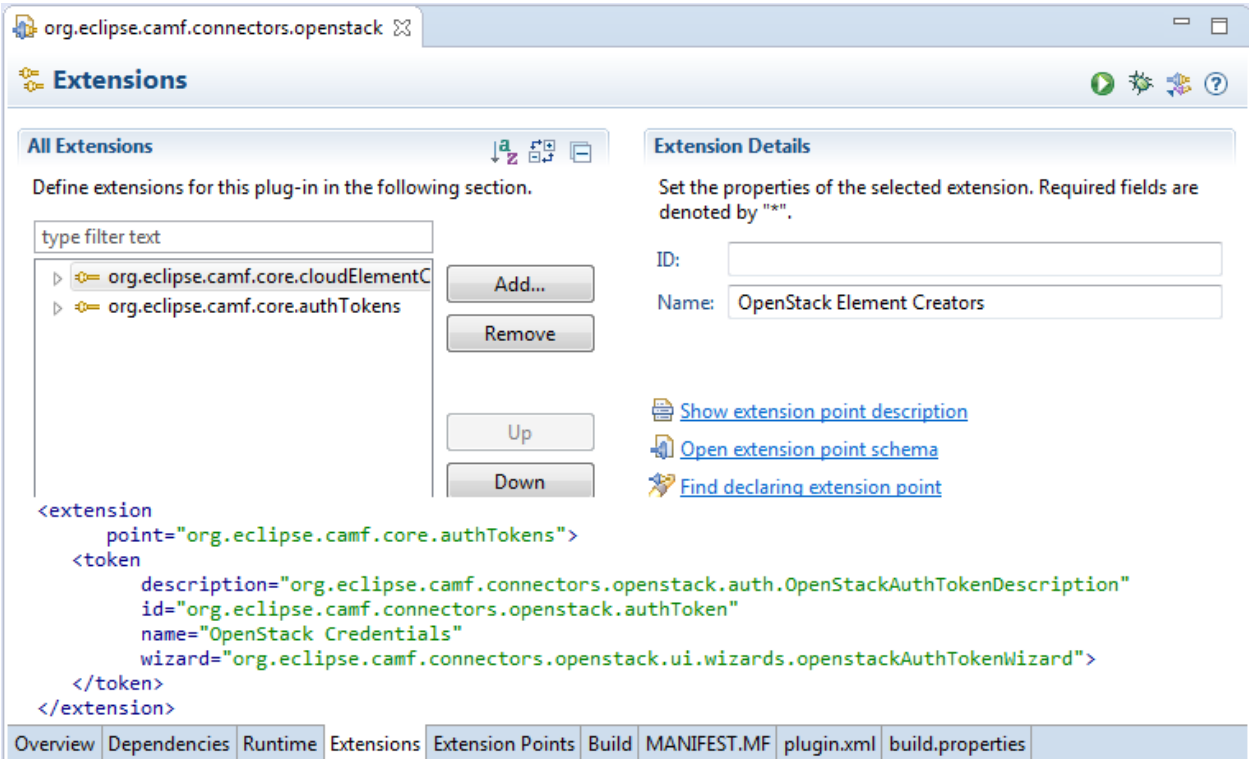
έναν Container με σκοπό την επεξεργασία του αρχείου αυτού και τη δημιουργία του μοντέλου της περιγραφής έτσι ώστε να ολοκληρωθεί η ανάπτυξή της.

Εδώ, είναι και το σημαντικό μειονέκτημά του. Σε περίπτωση που ο πάροχος δεν υποστηρίζει το πρότυπο OpenTOSCA και δεν παρέχει κάποιον Container με σκοπό την επεξεργασία του CSAR, το CAMF, από την στιγμή που θα στείλει την περιγραφή δεν θα γίνει κάποια περαιτέρω ενέργεια καθώς δεν θα αναγνωρίζει περί τίνος πρόκειται. Συγχρόνως, η ίδια εφαρμογή είναι σε θέση να επιλύσει το πρόβλημα αυτό, προσφέροντας κάποια Extension Point [15] τα οποία πρέπει να υλοποιηθούν μέσω Extension σε συνδυασμό με Abstract υλοποιήσεις με σκοπό ο χρήστης να προσφέρει αυτή την λειτουργία που λείπει και να διορθώσει το συγκριμένο μειονέκτημα.
Η καλύτερη μεταφορά για την κατανόηση του όρου Extension Point και Extension που αναφέρονται πιο πάνω και πρέπει να υλοποιηθούν είναι αυτή της ηλεκτρικής πρίζας. Η πρίζα - υποδοχή μπορεί να προσδιοριστεί ως το Extension Point και το καλώδιο που συνδέεται ως Extension. Όπως και στις πρίζες, μπορούν να συνδεθούν μόνο καλώδια που υποστηρίζονται. Όταν μία εφαρμογή θέλει να επεκταθεί, ορίζει ένα Extension Point που περιέχει κάποιες δηλώσεις - που συνήθως είναι ο συνδυασμός XML και JAVA Interfaces - ορίζοντας τι επιτρέπει να υλοποιηθεί από το Extension. Από τη πλευρά του το Extension, πρέπει να συμμορφώνεται με τους κανόνες που ορίζονται στο Extension Point και να υλοποιήσει τις λειτουργίες αυτές. Το κλειδί στην όλη διαδικασία είναι ότι το plug-in που θα δημιουργηθεί δεν γνωρίζει τίποτα για το plug-in που θα συνδεθεί παρά μόνο τους κανόνες που ορίζονται από το Extension Point. Αυτό επιτρέπει την ύπαρξη πολλαπλών plug-in τα οποία μπορούν να λειτουργούν παράλληλα χωρίς το ένα να γνωρίζει το για την ύπαρξη του άλλου. Υπάρχουν πολλά είδη Extension Point. Μερικά από αυτά είναι τα δηλωτικά που δεν χρειάζονται κάποιον επιπλέον κώδικα από τη πλευρά του Extension για να υλοποιηθεί η λειτουργία τους πχ. προσαρμοσμένες συντομεύσεις. Άλλο είδος είναι η παράκαμψη προεπιλεγμένων λειτουργιών όπως είναι η εισαγωγή ενός διαφορετικού code-formatter από αυτόν που χρησιμοποιεί το Eclipse. Τέλος, ακόμη μία κατηγορία είναι αυτή της ομαδοποίησης συσχετιζόμενων αντικειμένων στο γραφικό περιβάλλον. Όπως είναι η δημιουργία και η ομαδοποίηση των διαφόρων Wizard και ο ορισμός ενός συνεπούς τρόπου για την παρουσίαση των γραφικών στοιχείων.

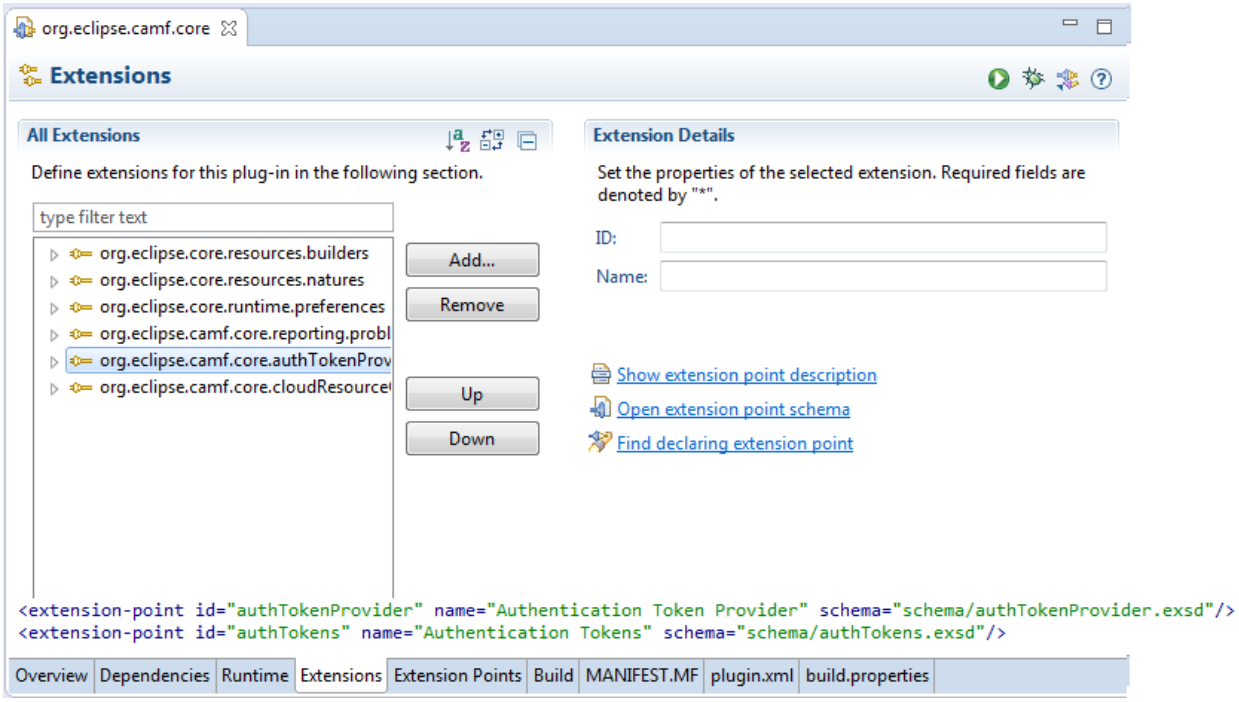
[15] https://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points%3F



(Σχήμα 2.3) Extension Point & Extensions



(Σχήμα 2.4) Ορισμός Extension για πιστοποίηση στο Openstack



(Σχήμα 2.5) Ορισμός Extension Point για πιστοποίηση στο OpenStack

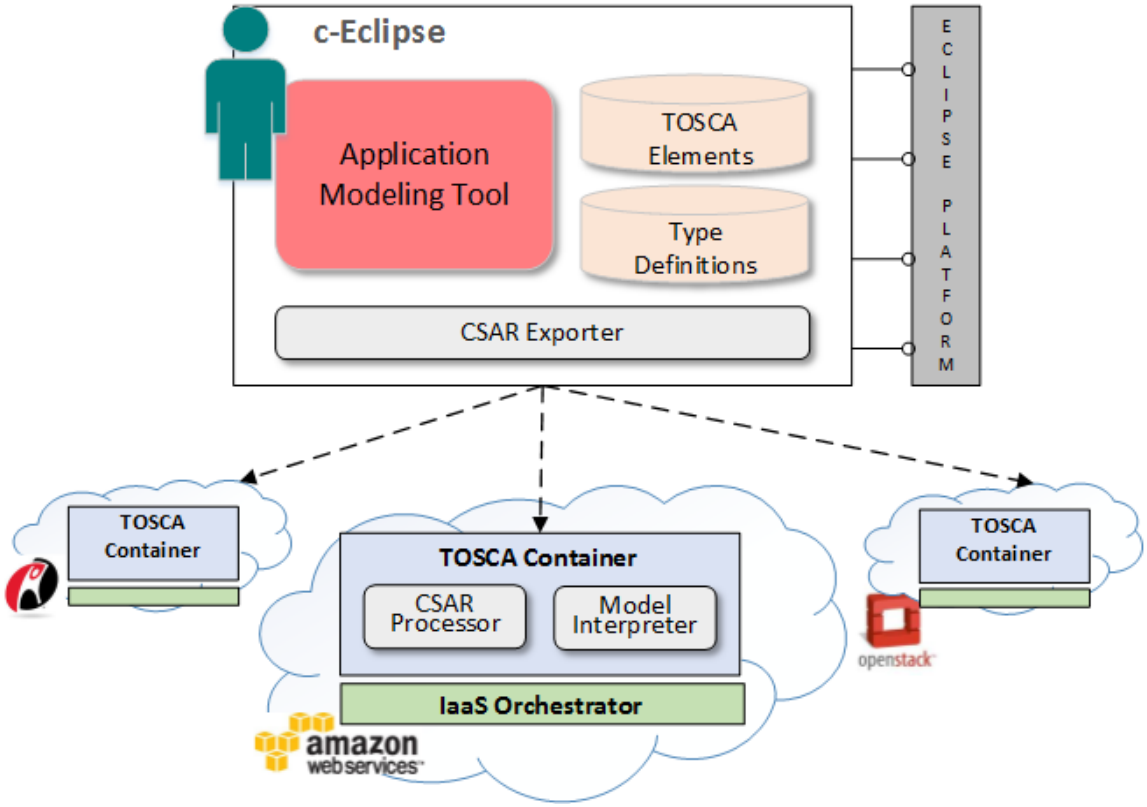
Κάνοντας χρήση των κατάλληλων Extension Point μπορεί να προστεθεί οποιαδήποτε λειτουργία επιθυμούμε όπως αναφέρθηκε και πιο πάνω. Μία από αυτές είναι ο καθορισμός πιο προχωρημένων καταστάσεων Monitoring. Χρησιμοποιώντας αυτά τα Extension Point και τις ανάλογες βιβλιοθήκες που υπάρχουν, είμαστε σε θέση να προσφέρουμε στην εφαρμογή τη λειτουργία σε οποιονδήποτε πάροχο ανεξάρτητα αν χρησιμοποιεί το πρότυπο OpenTOSCA. Με την χρήση της βιβλιοθήκης jClouds Connector είμαστε σε θέση να επικοινωνήσουμε με όλα τα IaaS στα οποία δίνει πρόσβαση μέσω συγκεκριμένων API κλήσεων με σκοπό την επίτευξη της ενέργειας που επιθυμούμε. Στο σενάριο αυτό, η περιγραφή που δημιουργήθηκε και μοντελοποιήθηκε από το TOSCA Meta-Model, αναλύεται τοπικά μέσω συγκεκριμένων λειτουργιών, ομαδοποιούνται τα στοιχεία που όρισε ο χρήστης και χρειάζονται για την ανάπτυξη, και κάνοντας χρήση συγκεκριμένων API κλήσεων που προσφέρει η βιβλιοθήκη jClouds, είμαστε σε θέση να επικοινωνήσουμε με οποιονδήποτε πάροχο άμεσα και να του στείλουμε τα απαραίτητα πράγματα που απαιτούνται για την ανάπτυξη. Με την αξιοποίηση του παραπάνω, είμαστε σε θέση να βγάλουμε από την μέση το κομμάτι της αποστολής του CSAR αρχείου και να αναμένουμε από τον πάροχο να το μεταφράσει και να το αξιοποιήσει με τη χρήση κάποιου OpenTOSCA Container καθώς, και οποιαδήποτε άλλη εξάρτηση προκύπτει.

Κεφάλαιο 3

Σχεδίαση και Υλοποίηση

3.1 Περιορισμοί του CAMF

Όπως αναφέρθηκε, το μεγαλύτερο μειονέκτημα στην τωρινή υλοποίηση του CAMF βρίσκεται στο γεγονός ότι είναι εξαρτημένο από το αν ο πάροχος υποστήριζει το πρότυπο TOSCA. Προϋποθέτει ο πάροχος να υποστηρίζει το πρότυπο OpenTosca και να παρέχει κάποιο Container με σκοπό να μεταφραστεί το CSAR που στέλνει το οποίο περιέχει τα απαραίτητα στοιχεία για την ανάπτυξη της περιγραφής στο σύννεφο όπως οι τεχνικές πληροφορίες για τα Virtual Machines, τυχόν επιπλέον script για εκτέλεση και πιθανά άλλα στοιχεία που έχει ορίσει ο χρήστης. Στην περίπτωση, που ο πάροχος δεν παρέχει αυτή τη δυνατότητα, τότε το CAMF δεν είσαι σε θέση να εξυπηρετήσει τον σκοπό του. Θα σταλθεί το CSAR αλλά δεν θα γνωρίζει ο πάροχος περί τίνος πρόκειται και τι να κάνει με αυτό το αρχείο. Η παρακάτω εικόνα, παρουσιάζει την όλη διαδικασία διαγραμματικά και δείχνει την εξάρτηση που υπάρχει στη πλευρά του παρόχου. Στην εικόνα υπάρχουν τρεις διαφορετικοί πάροχοι, Amazon, Openstack και RackSpace όπου και οι τρεις παρέχουν κάποιο Container αλλά διαφορετικό σε κάθε περίπτωση με σκοπό να μεταφραστεί το CSAR. Όπως παρουσιάζεται, δεν αρκεί ένα κοινό Container που θα μπορούσε να εξυπηρετήσει τον σκοπό αυτό, αλλά διαφορετικό από πάροχο σε πάροχο.



(Σχήμα 3.1) Τρέχουσα διαδικασία ανάπτυξης περιγραφής στον πάροχο[16]

3.2 Μπορεί να γίνει καλύτερο;

Το ερώτημα αυτό έχει μόνο μία απάντηση. Ναι! Αν παρακαμφθεί η διαδικασία που περιγράφεται στην πιο πάνω εικόνα. Δηλαδή, η αποστολή του TOSCA στον πάροχο με σκοπό να αναμένουμε να χρησιμοποιήσει κάποιον OpenTosca Container έτσι ώστε να τη μεταφράσει και να υλοποιήσει την ανάπτυξη που περιέγραψε ο χρήστης. Η λύση που προτείνουμε είναι η χρήση κάποιων επιπρόσθετων βιβλιοθηκών (jClouds[17]) με σκοπό την άμεση επικοινωνία μεταξύ του CAMF και του παρόχου παρακάμπτοντας τη διαδικασία μετάφρασης και αποστολής των απαραίτητων στοιχείων και στη συνέχεια την επεξεργασία τους από τον πάροχο. Το άμεσο πλεονέκτημα αυτού του εγχειρήματος είναι ότι επειδή βρισκόμαστε στη πλευρά του χρήστη με άμεση πρόσβαση στις λειτουργίες του CAMF, ό,τι χρειαζόμαστε είναι φορτωμένο στην μνήμη και άμεσα προσπελάσιμο.

[16] “c-Eclipse: An Open-Source Management Framework for Cloud Applications", C. Sofokleous, N. Loulloudes, D.Trihinas, G.Pallis and M. D. Dikaiakos, 20th International Conference on Parallel Processing (Euro-Par 2014), Porto, Portugal 2014
[17] https://jclouds.apache.org/

3.3 Απαιτήσεις

Οι απαιτήσεις για τη συγκεκριμένη υλοποίηση είναι η υποστήριξη του παρόχου από τη βιβλιοθήκη jClouds. Ευτυχώς, αυτό δεν παραμένει πρόβλημα καθώς παρέχει υποστήριξη για όλους τους διαδεδομένους παρόχους. Το jClouds πρόκειται για μία βιβλιοθήκη ανοιχτού κώδικα που παρέχει κάποια υλοποίηση από API που υποστηρίζει ο κάθε πάροχος με τις λειτουργίες του. Χωρίζει τις λειτουργίες του σε κάποια βασικά πακέτα όπως είναι το ComputeService όπου είναι υπεύθυνο για τη δημιουργία και τη διαχείριση των Instances. Το ComputeMetadata το οποίο παρέχει μετά-πληροφορίες για τα Instances και το NeutronAPI το οποίο παρέχει πληροφορίες για τη δομή του δικτύου. Παρέχει ένα απλό Interface με υψηλές τεχνικές προγραμματισμού, τρέχει σε όλες τις πλατφόρμες ανεξάρτητα του Λ/Σ του χρήστη, υποστηρίζει jUnit tests και παρέχει αρκετές δικλείδες απόδοσης όπως οι ασύγχρονες εντολές. Είναι σε θέση να υποστηρίζει 30 διαφορετικούς παρόχους[18]. Μερικοί είναι οι ακόλουθοι:

- Amazon
- Azure
- GoGrid
- OpenStack
- Rackspace
- Google.

3.4 Σχεδιασμός

Ο σχεδιασμός έπαιξε πολύ βασικό ρόλο. Σκοπός δεν ήταν να χαθεί η υπάρχουσα λειτουργικότητα που παρέχει το CAMF και ο τρόπος με τον οποίο γίνεται μέχρι στιγμής η ανάπτυξη της περιγραφής. Επομένως, πρωταρχική σκέψη ήταν να είμαστε σε θέση να παρέχουμε στον χρήστη ό,τι δυνατότητες και επιλογές είχε και πριν χωρίς να χαθεί κάποια λειτουργικότητα. Επιπλέον, ο χρήστης, να έχει τη δυνατότητα να επιλέξει με ποιον τρόπο θέλει να ολοκληρωθεί η αποστολή της περιγραφής στον πάροχο και να ολοκληρωθεί η ανάπτυξη της εφαρμογής με αποτέλεσμα να παραμένει τόσο ο αρχικός τρόπος που γίνεται η ανάπτυξη (σχ. 3.1) αλλά και να έχει τη δυνατότητα να επιλέξει την νέα λειτουργία.

[18] https://jclouds.apache.org/reference/providers/

3.4.1 Σχεδιασμός και ανάπτυξη του Deployer

Λόγω των διαφορετικών παρόχων, θέλουμε ο Deployer να μπορεί να υποστηρίξει όσους πιο πολλούς γίνεται. Η δομή, η λειτουργία και η συνεκτικότητά του να παραμένει ίδια μεταξύ αυτών. Καθώς επίσης, να μην υπάρχει κάποιος περιορισμός από τη πλευρά της εφαρμογής και να είναι σε θέση να κάνει ότι επιθυμεί ο χρήστης μέχρι εκεί που του επιτρέπει ο κάθε πάροχος.

Για τον λόγο αυτό, χρησιμοποιήθηκε μία ιεραρχική δομή ως προς την ανάπτυξή του που μπορεί να περιγραφεί ως δενδρική δομή. Αρχικά έγινε ο καθορισμός ενός Interface όπου ορίζει τις υποχρεωτικές συναρτήσεις που πρέπει να υποστηρίζονται και να παρέχονται οι οποίες επιγραμματικά είναι οι ακόλουθες:

- Προσκόμιση ενεργών Virtual Instances
- Προσκόμιση διαθέσιμων Flavor
- Προσκόμιση διαθέσιμων Εικόνων
- Προσκόμιση περιορισμών λογαριασμού
- Προσκόμιση επιπλέον υπηρεσιών
- Προσκόμιση διαθέσιμων δικτύων
- Προσκόμιση διαθέσιμων Key pairs
- Προσκόμιση διαθέσιμων Security Groups
- Δημιουργία εικόνας από Virtual Instance
- Δημιουργία Virtual Instance
- Τερματισμός Virtual Instance

Ο κώδικας για τις συγκεκριμένες υλοποιήσεις δίνεται στο παράρτημα Α.

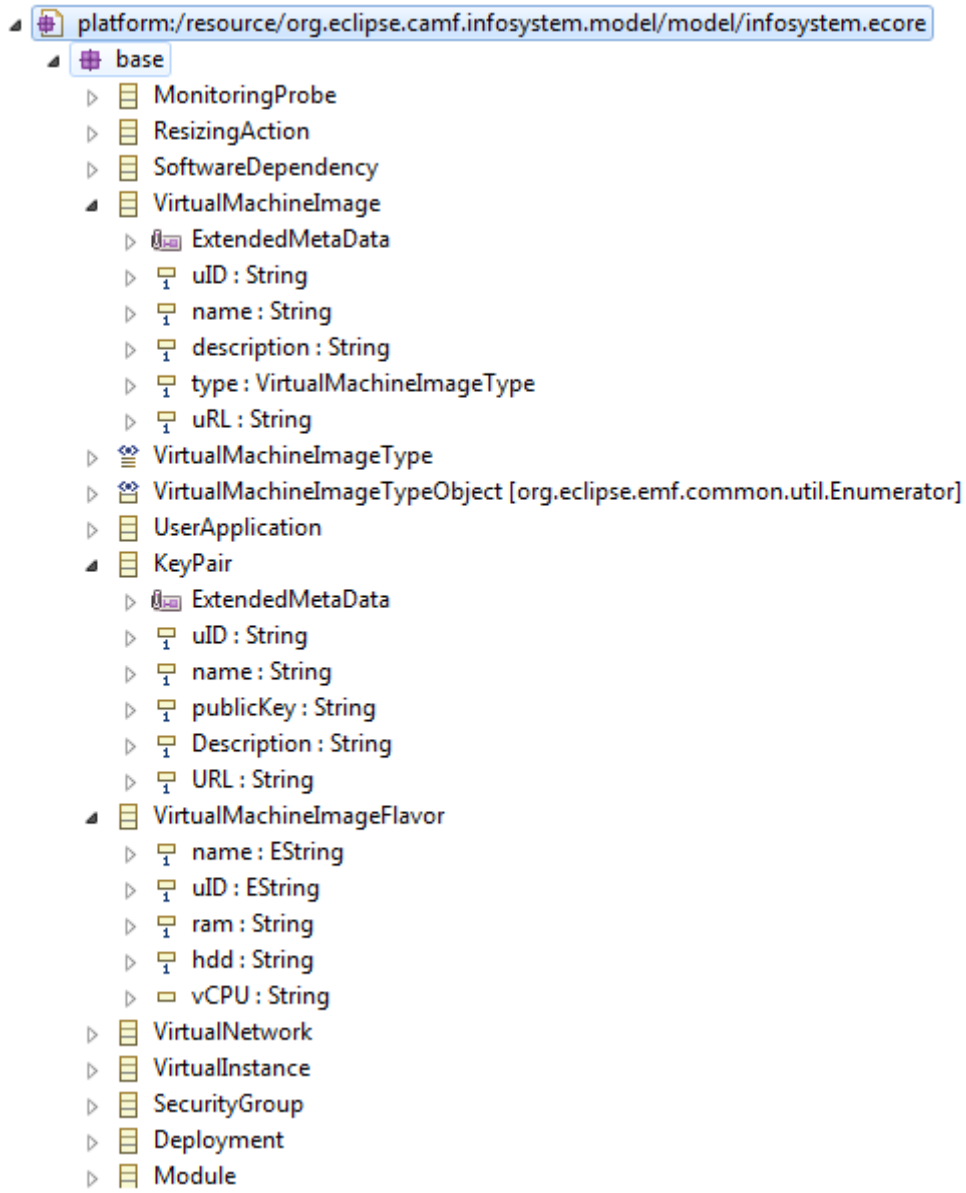
Ύστερα, υλοποιήθηκε μία κεντρική κλάση με σκοπό την επικοινωνία με το OpenStack όπου υλοποιούνται αυτές οι λειτουργίες που ορίστηκαν στο Interface μέσω των κλήσεων API που προσφέρει η βιβλιοθήκη jClouds. Για κάθε είδος αντικειμένου που επιστρέφεται, υλοποιήθηκε και η αντίστοιχη κλάση (Bean) για την συλλογή και αποθήκευση των αποτελεσμάτων. Τα διαθέσιμα Beans (επιγραμματικά) είναι:

- FlavorObj
- ImageObj
- Instance
- KeypairsObj
- NetworkObj
- SecurityGroupsObj

Για παράδειγμα, κατά την προσκόμιση των ενεργών Virtual Instances, χρησιμοποιείται η κλάση που είχε οριστεί όπου είχε τα κατάλληλα πεδία για το αποτέλεσμα που επιστρέφει ο πάροχος και μπορεί να δεχθεί μόνο τέτοιου είδους δεδομένα. Για την συγκεκριμένη οντότητα αποθηκεύουμε κάποια βασικά χαρακτηριστικά όπως είναι το όνομα του Virtual Instance και το μοναδικό ID του. Επιπλέον, αποθηκεύονται πιο προχωρημένες πληροφορίες όπως οι IP διευθύνσεις που έχουν αντιστοιχηθεί στο συγκεκριμένο Virtual Instance, η κατάσταση του και το URI του για απευθείας πρόσβαση σε αυτό.

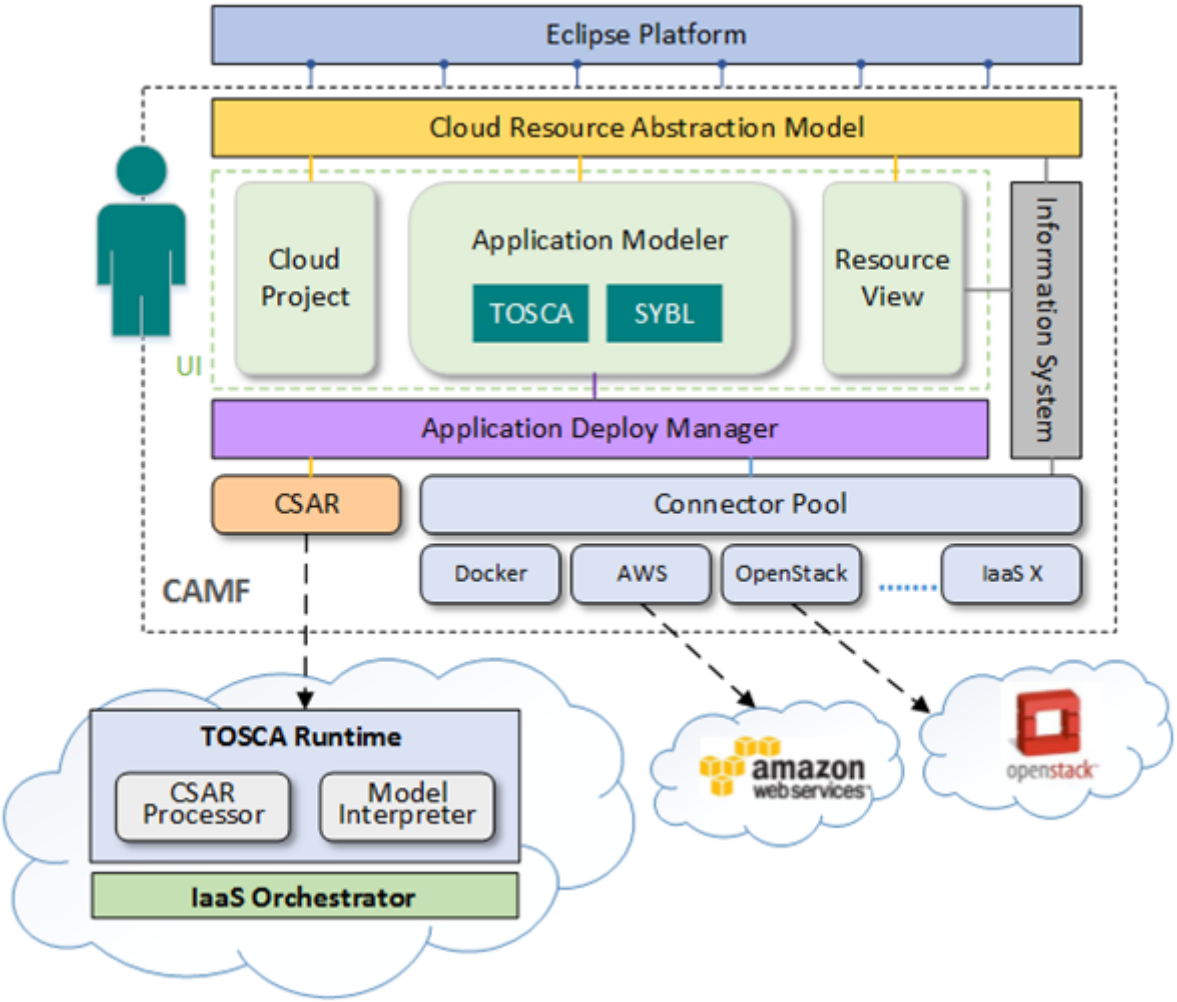
Σε όλες τα αντικείμενα, αποθηκεύονται κάποιες κοινές πληροφορίες όπως είναι τα μοναδικά ID τους, το όνομα του κάθε αντικειμένου και η περιγραφή του. Για παράδειγμα τόσο τα αντικείμενα τύπου Flavor έχουν κάποια μοναδικά ID's, ονόματα και περιγραφές όσο και τα αντικείμενα τύπου Key pair. Συγχρόνως όμως, στα αντικείμενα τύπου flavor αποθηκεύουμε επίσης και πιο προχωρημένες πληροφορίες όπως ο διαθέσιμος δίσκος που υποστηρίζει, η διαθέσιμη RAM και ο αριθμός των vCPU. Από την άλλη στα key pairs αποθηκεύονται επιπλέον το Public Key του κάθε key pair και το URI του.

Αυτό οδήγησε στον διαχωρισμό μεταξύ του όγκου των πληροφοριών που επεξεργαζόμαστε και στη καλύτερη ασφάλεια όσον αφορά την ορθότητα και την εγκυρότητα των δεδομένων. Αποκλείουμε έτσι τη πιθανότητα αντικείμενα διαφορετικού τύπου να αποθηκευτούν σε κλάση διαφορετική από αυτή που ανήκουν με λανθασμένες πληροφορίες. Τέλος, έπρεπε να προσφέρονται κάποιες μετά-πληροφορίες για περαιτέρω αξιοποίηση από τη πλευρά του χρήστη με το αποτέλεσμα της ανάπτυξης και τον κύκλο ζωής της.



(Σχήμα 3.9) Προβολή αντικειμένων και πεδίων των Flavor, Key Pair και Virtual Image στο EMF

Για σκοπούς φορητότητας μεταξύ των διαθέσιμων παρόχων, θα έπρεπε να υλοποιηθεί μόνο η αντίστοιχη κλάση που υλοποιεί τη σύνδεση και τη προσκόμιση των δεδομένων μέσω του API που προσφέρει το jClouds για τον εκάστοτε πάροχο, καθώς όλα τα άλλα παραμένουν ίδια ως προς τη λειτουργία τους κάνοντας την όλη υλοποίηση εύκολη στη φορητότητα μεταξύ διαφόρων παρόχων.



(Σχήμα 3.2) Αρχιτεκτονική του CAMF

3.4.2 Mapping TOSCA Περιγραφής

Επειδή, η υλοποίηση του συγκεκριμένου Connector, βρίσκεται από τη πλευρά του χρήστη, έχουμε το πλεονέκτημα ότι έχουμε απευθείας πρόσβαση στο μοντέλο της περιγραφής του χρήστη το οποίο βρίσκεται ήδη φορτωμένο στη μνήμη και μας το παρέχει ο πυρήνας του CAMF. Έτσι, γνωρίζουμε ανά πάσα στιγμή την περιγραφή του χρήστη και τις όποιες αλλαγές πραγματοποιεί και με τις κατάλληλες κλήσεις σε βασικές συναρτήσεις της εφαρμογής παίρνουμε αποκτάμε πρόσβαση στις πληροφορίες που χρειαζόμαστε για την ολοκλήρωση της ανάπτυξης.

Σε αντίθεση με την Standalone εφαρμογή, όπου έχει υλοποιηθεί ένας Parser που διαβάζει την XML TOSCA περιγραφή που δίνεται από τον χρήστη και παίρνει τα απαραίτητα στοιχεία τα οποία περνάει στον Deployer με σκοπό την περαιτέρω επεξεργασία τους και την τελική ανάπτυξη.

Σε κάθε περίπτωση, γίνεται διάβασμα της TOSCA περιγραφής με σκοπό να παρθούν οι βασικές οντότητες του. Όπως είναι τα Components-Virtual Instances που έχει περιγράψει ο χρήστης και από τι αποτελούνται, εικόνα, keypair, flavor κλπ., τυχόν επιπλέον script, τις ιδιότητες που έχει ορίσει, όπως πόσα Instances να εκκινούνται κατά την ανάπτυξη και ότι άλλο είναι απαραίτητο για την υλοποίησή της. Ύστερα, αποθηκεύονται σε αντικείμενα τύπου HashMap με σκοπό να χρησιμοποιηθούν περαιτέρω από τον Deployer όπου θα πάρει στην εκάστοτε περίπτωση ότι χρειάζεται και θα το αποστείλει στον πάροχο μέσω του API που προσφέρει το jClouds.

3.4.3 Υλοποιήσεις

Αρχικά έγιναν δύο διαφορετικές υλοποιήσεις. Μία Standalone εφαρμογή για σκοπούς λειτουργικότητας και να ελέγξουμε αν εξυπηρετεί τους σκοπούς του Connector όπως αυτοί είχαν οριστεί και τέλος, η ενσωμάτωση της υλοποίησης στο CAMF και η βελτίωσή της.

3.4.3.1 Υλοποίηση Standalone Εφαρμογής (Cloud Service Deployer[19])

Η συγκεκριμένη υλοποίηση παρέχει τη δυνατότητα σύνδεσης με το OpenStack. Έχουν οριστεί όλα τα απαραίτητα Beans που περιγράφηκαν πιο πάνω, για κάθε πιθανό αντικείμενο που παρέχεται από τον πάροχο και το καθένα από αυτά έχει συγκεκριμένα πεδία για τις πληροφορίες του. Για παράδειγμα το αντικείμενο για τα Virtual Instances έχει τα δύο βασικά πεδία που μας ενδιαφέρουν το όνομα και το μοναδικό ID του. Είναι σε θέση να προσκομίσει όλα τα απαραίτητα στοιχεία που έχουν οριστεί όπως Images, Flavors, Networks κλπ., χρησιμοποιώντας τα κατάλληλα Beans για την αποθήκευσή τους όπως παρουσιάστηκαν στην ενότητα 3.3.1 και να τα εμφανίσει στο χρήστη μέσω ενός πρόχειρου γραφικού περιβάλλοντος (σχ. 3.3) για σκοπούς πληροφόρησης. Επίσης είναι σε θέση, να αναπτύξει οποιαδήποτε περιγραφή του δώσει ο χρήστης σαν είσοδο ανεξάρτητα από τον αριθμό των Components που έχουν οριστεί και το βάθος τους καθώς επίσης και να τη τερματίσει. Επιπλέον, επιστρέφει στο χρήστη το αποτέλεσμα μέσω JSON (σχ. 3.4) με αναλυτικά στοιχεία για το αποτέλεσμα της ανάπτυξης. Τέλος, δίνει τη δυνατότητα στον χρήστη να χρησιμοποιήσει είτε το γραφικό περιβάλλον που παρέχει για σκοπούς επίδειξης είτε να το χρησιμοποιηθεί μέσω του Command Prompt.

Πιο συγκεκριμένα, κατά την εκκίνηση της εφαρμογής μέσω του γραφικού περιβάλλοντος, προβάλλεται στον χρήστη το διαθέσιμο μενού (σχ. 3.3) και γίνεται η σύνδεση με τον πάροχο.

[19] https://github.com/UCY-LINC-LAB/CloudServiceDeployer

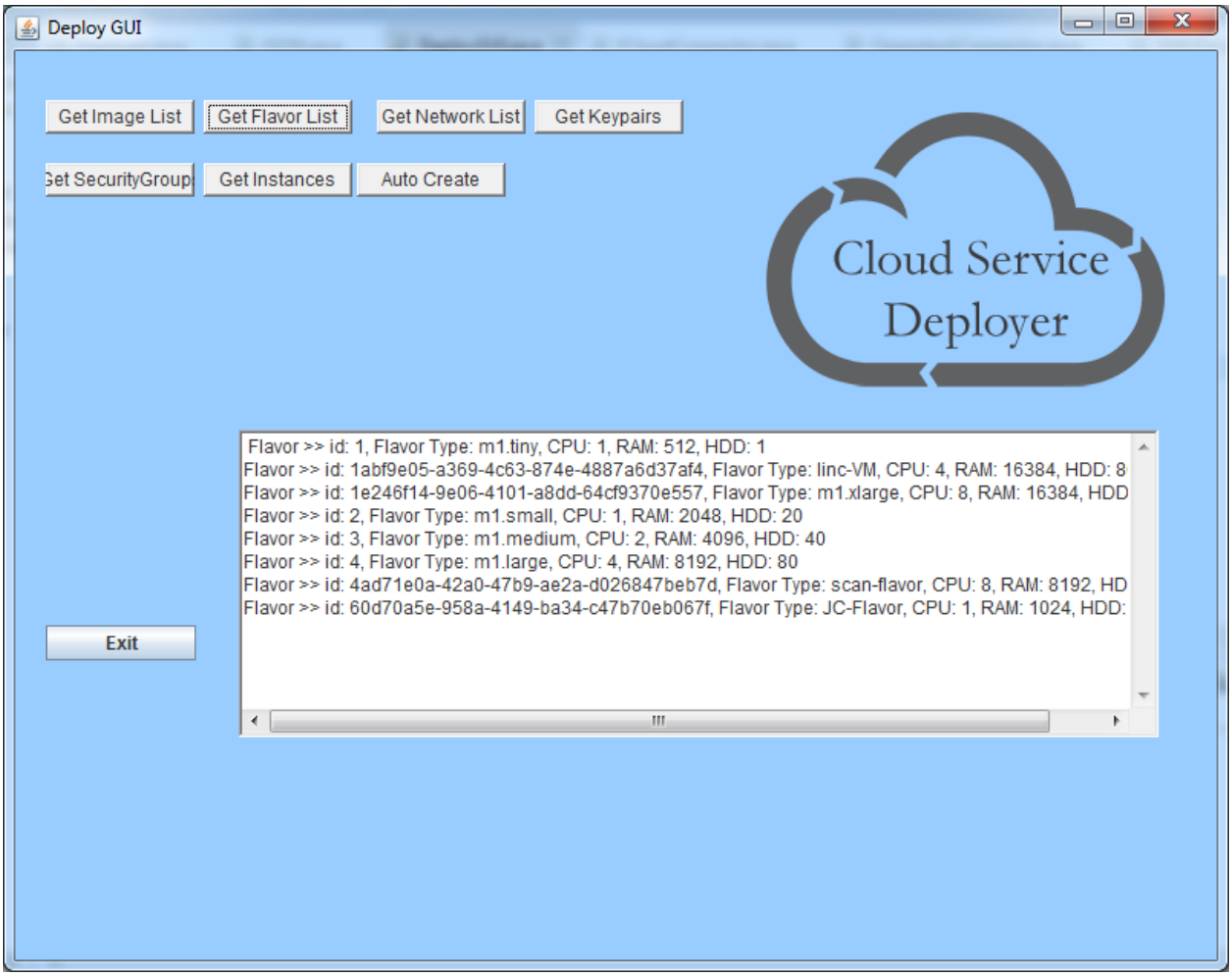
Αν είναι επιτυχής, επιτρέπει στον χρήστη να συνεχίσει με τις ενέργειες που επιθυμεί, διαφορετικά θα του εμφανίσει το ανάλογο σφάλμα.

Για κάθε λειτουργία προσκόμισης δεδομένων, χρησιμοποιούνται οι συναρτήσεις που έχουν οριστεί και υλοποιηθεί και εμφανίζονται τα αποτελέσματα στο διαθέσιμο Textarea που φαίνεται στο σχήμα 3.3.

Σε περίπτωση που ο χρήστης, επιθυμεί την ανάπτυξη μιας περιγραφής, η εφαρμογή του ζητά να δώσει το μονοπάτι από την TOSCA περιγραφή, την οποία διαβάζει με τον αντίστοιχο Parser που έχει υλοποιηθεί στην εφαρμογή, και συλλέγει τα απαραίτητα στοιχεία που χρειάζονται για να την αναπτύξει. Ορίζει για κάθε περιγραφή ένα μοναδικό Deployment ID, καθώς και ένα όνομα για την περιγραφή το οποίο παίρνει από το TOSCA και μία λίστα από Modules-Components. Για κάθε component, που διαβάζει από το TOSCA, δημιουργείται ένα μοναδικό Module ID όπου χαρακτηρίζει το κάθε Component μαζί με το όνομα του και δημιουργεί και μία λίστα για τα Instances που θα δημιουργηθούν η οποία αντιστοιχείται σε ένα μόνο Module. Στο στάδιο αυτό, διαβάζει από το TOSCA τα απαραίτητα στοιχεία που θα έχει το συγκεκριμένο Instance όπως είναι το Network, Image ID, key pair και τα υπόλοιπα στοιχεία τα οποία αποθηκεύει για επεξεργασία με τελικό σκοπό την χρήση τους κατά την ανάπτυξη. Αφού έχει συλλέξει όλα τα απαραίτητα στοιχεία για το Virtual Instance που πρέπει να δημιουργεί γίνεται χρήση των ανάλογων API call μέσω του jClouds με σκοπό την δημιουργία του. Εφόσον η δημιουργία του είναι επιτυχής, επιστρέφεται ένα μοναδικό Instance ID για το Virtual Instance που δημιουργήθηκε. Κάθε Virtual Instance, είναι αντιστοιχισμένο σε ένα Module και κάθε Module σε ένα Deployment. Κατά την ολοκλήρωση της ανάπτυξης, δημιουργείται μέσω της ανάλογης κλάσης ένα JSON αρχείο με τις λεπτομέρειες της περιγραφής όπως, τα Instances που αναπτύχθηκαν και οι πληροφορίες τους καθώς και την τελική κατάσταση της περιγραφής. Αν αναπτύχθηκε ή όχι και για ποιον λόγο. Επίσης είναι σε θέση να τερματίσει κάποιο Module ή instance μεμονωμένα βάση του Module ID ή του Instance ID που δημιουργούνται ή και ολόκληρης της περιγραφής βάση του Deployment ID.

Τόσο τα Deployment ID και Module ID δημιουργήθηκαν για σκοπούς λειτουργικότητας και πιο εύκολης χρήσης της εφαρμογής. Από την στιγμή που όλα τα ID είτε πρόκειται για Deployment είτε για Module είτε για Instance, είναι συσχετισμένα μεταξύ τους, μπορεί άμεσα να γίνει συσχέτιση και προσπέλαση των στοιχείων που περιέχονται χωρίς να χρειάζεται να γνωρίζουμε κάθε ID μεμονωμένα.

Επίσης, καθώς η έκδοση αυτή δημιουργήθηκε για σκοπούς επίδειξης και για τον ορισμό της λειτουργικότητας του Connector δεν συνεχίζεται περαιτέρω η ανάπτυξή της παρά μόνο η διόρθωση τυχόν σφαλμάτων που μπορεί να προκύψουν.



(Σχήμα 3.3) Γραφικό Περιβάλλον Cloud Service Deployer όπου παρουσιάζονται τα διαθέσιμα Flavor

3.4.3.2 Ενσωμάτωση στο CAMF

Ως πρώτη απόπειρα έγινε η ενσωμάτωση της πιο πάνω υλοποίησης με την ίδια ακριβώς λειτουργικότητα στο CAMF ακολουθώντας τα σχεδιαστικά πρότυπα τα οποία είχαν σχεδιαστεί. Για κάθε αντικείμενο (Bean) που υπήρχε στην Standalone εφαρμογή έγινε μεταφορά του στο *Eclipse Modeling Framework* (EMF) και οι κλάσεις ακολουθούν τη δομή που υπήρχε όσον αφορά τα αντικείμενά τους (σχ. 3.9).

Πιο συγκεκριμένα κάθε λειτουργία έχει τη δική της κλάση. Αρχικά, υλοποιήθηκε μία Abstract κλάση που όριζε τις συναρτήσεις που πρέπει να υλοποιηθούν και τι είδους αντικείμενα θα δέχεται και όριζε επιπλέον κάποιες βασικές συναρτήσεις από το βασικό Interface του Connector. Στη συνέχεια υλοποιούνταν αυτή η Abstract κλάση που στην ουσία είναι οι κλήσεις μέσω του API που προσφέρει το jClouds (σχ. 3.5 & 3.6).

```java
public abstract class AbstractOpenStackOpDescribeImages implements IOperation {

  /** The resulting list of AMIs */
  private List<Image> result;

  /** Any exception which came up during the inquiry. */
  private Exception exception;

  abstract public void run();

  public List<Image> getResult() {
    return this.result;
  }

  public Exception getException() {
    return this.exception;
  }

  /**
   * A setter for {@link #result}.
   *
   * @param describeImagesByOwner the param to set
   */
  protected void setResult( final List<Image> describeImages )
  {
    this.result = describeImages;
  }

  /**
   * A setter for {@link #exception}.
   *
   * @param ex the exception to set
   */
  protected void setException( final Exception ex ) {
    this.exception = ex;
  }
}
```

(Σχήμα 3.5) Abstract κλάση για την προσκόμιση των διαθέσιμων εικόνων

```
public class OpenStackOpDescribeImages extends
        AbstractOpenStackOpDescribeImages {

    private final ComputeService computeService;

    /**
     * Creates a new {@link OpenStackOpDescribeImages} with the given owners as
     * parameter.
     *
     * @param computeService
     *            the {@link ComputeService} to obtain data from
     */
    public OpenStackOpDescribeImages(final ComputeService computeService) {
        this.computeService = computeService;
    }

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {

            Set<Image> images = (Set<Image>) this.computeService.listImages();
            setResult(new ArrayList<Image>(images));
        } catch (Exception ex) {
            setException(ex);
        }

    }

}
```

(Σχήμα 3.6) Υλοποίηση Abstract κλάσης για την προσκόμιση των διαθέσιμων εικόνων

Το μεγάλο πλεονέκτημα εδώ όπως αναφέρθηκε είναι ότι η περιγραφή του χρήστη βρίσκεται ήδη στη μνήμη και δεν χρειάζεται να γίνεται Parsing κάποιου TOSCA αρχείου αλλά απλά κλήσεις σε βασικές συναρτήσεις του πυρήνα της εφαρμογής με σκοπό την ανάκτηση των τιμών αυτών. Ακολουθείται πάλι η ίδια διαδικασία, για κάθε περιγραφή δημιουργείται ένα μοναδικό Deployment ID που περιέχει μία λίστα από Modules. Για κάθε Component δημιουργείται ένα μοναδικό Module ID που περιέχει μία λίστα με τα Instances που αντιστοιχούν στο Module αυτό. Πάλι υπάρχει η δυνατότητα τερματισμού κάποιου αντικειμένου είτε μέσω του Instance ID είτε μέσω του Module ID αν πρόκειται για κάποιο Module είτε ολόκληρης της περιγραφής μέσω του Deployment ID. Τόσο και σε αυτή την περίπτωση, τα συγκεκριμένα ID's και αντικείμενα, βρίσκονται για σκοπούς διευκόλυνσης της επικοινωνίας και για σκοπούς συσχέτισης μεταξύ των οντοτήτων της περιγραφής. Τέλος, δημιουργούνταν και πάλι το αντίστοιχο JSON με το αποτέλεσμα της ανάπτυξης και τις επί μέρους οντότητές της.

Ύπάρχει και εδώ το πλεονέκτημα της φορητότητας μεταξύ τον παρόχων. Οι μόνες αλλαγές που πρέπει να γίνουν είναι αυτή της κλάσης που υλοποιεί τη σύνδεση με τον πάροχο και οι

υλοποιήσεις των Abstract κλάσεων που περιέχουν τις κλήσεις των API που προσφέρει το jClouds.

Κατά την συνέχεια της ανάπτυξης της εφαρμογής, προστέθηκε η δυνατότητα ο χρήστης να έχει τη δυνατότητα να βλέπει όλα τα Instances τα οποία εκτελούνται τη δεδομένη χρονική στιγμή μέσα από το γραφικό περιβάλλον του CAMF καθώς και η ενσωμάτωση των λειτουργιών προσκόμισης που έλειπαν στο γραφικό περιβάλλον. Προστέθηκαν νέα πεδία όπως το URI του Instance καθώς και οι IP's που του έχουν αντιστοιχηθεί όπως επίσης και η κατάστασή του, επιπλέον πληροφορίες για τα Key pairs όπως η περιγραφή, το public key και fingerprint, στα flavor οι πληροφορίες για κάθε flavor όπως η διαθέσιμη RAM, ο διαθέσιμος δίσκος και ο αριθμός των vCPU και τέλος, στα Security Group η περιγραφή τους.

(Σχήμα 3.4) JSON Μετά-πληροφορίες από επιτυχής ανάπτυξη

3.4.3.2.1 Διαχείριση Λαθών

Επίσης για σκοπούς κυρίως εμφάνισης προς τον χρήστη, όλα τα πιθανά σφάλματα που μπορεί να προκύψουν σε μία διαδικασία όπως είναι η προσκόμιση των διαθέσιμων Images, Flavors, η δημιουργία ή ο τερματισμός ενός Virtual Instance, χειρίζονται από την εφαρμογή με πιο φιλικά μηνύματα σε αντίθεση με αρκετές σειρές σφαλμάτων που πολλές φορές μπορεί να μην είναι και τόσο κατανοητές από τον μέσο χρήστη. Πιο συγκεκριμένα, για κάθε πιθανό σφάλμα έχει δημιουργηθεί και ένας νέος τύπος Exception όπου αναλαμβάνει να διαχειριστεί το αρχικό σφάλμα, να αποθηκεύσει το πρόβλημα για σκοπούς αποσφαλμάτωσης από τον προγραμματιστή και συγχρόνως να εμφανίσει μέσα σε μία γραμμή τι πήγε λάθος με ένα φιλικό μήνυμα προς τον χρήστη. Έτσι, για παράδειγμα όταν ο χρήστης θα προσπαθήσει να κάνει μία ανάπτυξη χρησιμοποιώντας ένα Flavor ID που δεν υπάρχει, θα του εμφανίσει το μήνυμα "*DEPLOY Exception: Failed to deploy. Please contact your system administrator.*" και στο παρασκήνιο του προγράμματος θα έχουν αποθηκευτεί αναλυτικά οι όλες λεπτομέρειες που θα πρέπει να λάβει υπόψη ο διαχειριστής του συστήματος.
Η σημαντικότερη αλλαγή, βρίσκεται στο πως διαχειριζόμαστε πλέον τυχόν λάθη που μπορεί να προκύψουν κατά την ανάπτυξη κάποιου μέρους των Components που έχει περιγράψει ο χρήστης. Όταν για παράδειγμα, ο χρήστης έχει μία αρκετά μεγάλη περιγραφή που αποτελείται από 100 Virtual Instances και για κάποιο λόγο η ανάπτυξη του 64^{ου} αποτύχει, πως θα ενημερωθεί ο χρήστης πέρα από το τυπικό μήνυμα σφάλματος που θα προκύψει; Τι ενέργειες μπορεί να κάνει;
Πλέον, ο Deployer έγινε πιο "έξυπνος" καθώς είναι σε θέση να υποστηρίξει τις τρεις ακόλουθες λειτουργίες κατά την ανάπτυξη μιας περιγραφής και σε περίπτωση που προκύψει κάποιο σφάλμα να το διαχειριστεί αναλόγως.

- CONTINUE_ON_ERROR
- STOP_ON_ERROR
- TERMINATE_AND_REVERT_ON_ERROR

Η πρώτη επιλογή και προεπιλεγμένη επιλογή, είναι η **CONTINUE_ON_ERROR.** Η οποία, όπως είναι και ο τίτλος της, απλά ενημερώνει τον χρήστη για το σφάλμα που προέκυψε αλλά δεν προβαίνει σε οποιαδήποτε άλλη ενέργεια. Προσπαθεί να συνεχίσει με την ανάπτυξη των υπόλοιπων Virtual Instance και αν προκύψει κάποιο επιπλέον σφάλμα, το αγνοεί. Παρά μόνο, στο τελικό JSON με τις μετά-πληροφορίες, η κατάσταση της περιγραφής είναι της μορφής "ολοκληρώθηκε αλλά με σφάλματα". (σχ. 3.7)

Η δεύτερη επιλογή, **STOP_ON_ERROR**, σε αντίθεση με την πρώτη, όταν προκύψει κάποιο σφάλμα, σταματάει η όλη διαδικασία της ανάπτυξης. Δεν γίνεται προσπάθεια να αναπτυχθούν τυχόν εναπομείναντα Virtual Instances. Ο χρήστης ενημερώνεται μέσω ενός μηνύματος σφάλματος στην οθόνη και δημιουργείται πάλι το αντίστοιχο JSON αρχείο με τις μετά-πληροφορίες το οποίο είναι πάλι της ίδιας μορφής. (σχ. 3.7)

Τέλος, η τρίτη επιλογή, **TERMINATE_AND_REVERT_ON_ERROR**, η οποία σε περίπτωση σφάλματος, τερματίζει τον ανάπτυξη της τρέχουσας περιγραφής όπως κάνει και η δεύτερη επιλογή, αλλά συγχρόνως, όσα Virtual Instances έχουν δημιουργηθεί πριν το σφάλμα τερματίζονται. Όπως και στις προηγούμενες περιπτώσεις, έτσι και σε αυτή, ο χρήστης ειδοποιείται με κάποιο μήνυμα σφάλματος. Εν αντιθέσει με τις άλλες δύο επιλογές, εδώ δεν δημιουργείται κάποιο JSON αρχείο με μετά-πληροφορίες για την ανάπτυξη, καθώς πλέον, δεν υπάρχει κάποιο στοιχείο ότι υπήρξε ποτέ.

Κατά την διαδικασία της ανάπτυξης, ο χρήστης, θα έχει την δυνατότητα να επιλέξει μέσω του γραφικού περιβάλλοντος του CAMF μία από τις τρεις πολιτικές η οποία τον εξυπηρετεί.

Επιπλέον, έγιναν σημαντικές αλλαγές στον τρόπο με τον οποίο δημιουργούνται και οργανώνονται τα JSON αρχεία που είναι υπεύθυνα για την αποθήκευση των μετά-πληροφοριών της ανάπτυξης και για τυχόν σφάλματα που μπορεί να προκύψουν.

Αρχικά, δημιουργείται ένα εξωτερικό πεδίο το οποίο είναι ένας πίνακας που περιέχει τα ακόλουθα στοιχεία:

- Status
- Deployment ID
- Application Name
- Provider
- Start Time
- Finish Time

και ένα αντικείμενο που περιέχει πίνακες από τα αντίστοιχα Modules της περιγραφής με πεδία τις βασικές τους πληροφορίες όπως:

- Module ID
- Module Name
- InitInstances

όπου το τελευταίο πεδίο είναι ο αριθμός από τα αρχικά Instances που πρέπει να δημιουργηθούν. Κάθε Module περιέχει ένα αντικείμενο από Instances που περιέχει πίνακες οι οποίοι έχουν πληροφορίες για τα Virtual Instances που έχουν δημιουργηθεί. Για κάθε Instance, αποθηκεύονται τα:

- Instance ID

- Image ID
- Keypair
- Flavor ID

Τα παραπάνω, φαίνονται αναλυτικά στα σχήματα 3.7 και 3.8.

Τέλος, μία ακόμη σημαντική αλλαγή η οποία δεν είναι διαθέσιμη στην Standalone εφαρμογή, είναι η αξιοποίηση των όλων πιο πάνω μετά-πληροφοριών. Σε περίπτωση, που ο χρήστης τερματίσει την εφαρμογή του CAMF, οποιαδήποτε στοιχεία και πληροφορίες σχετικά με την περιγραφή του έχουν χαθεί. Με αποτέλεσμα να μην έχει τη δυνατότητα να προβεί σε μεμονωμένες αλλαγές όπως για παράδειγμα ο τερματισμός κάποιου συγκεκριμένου Virtual Instance, να ελέγξει την εγκυρότητα της περιγραφής ή και ακόμη να την τερματίσει όταν επανεκκινήσει την εφαρμογή. Έτσι, για κάθε περιγραφή που υπάρχει στο CAMF και έχει αναπτυχθεί στον πάροχο, πλέον είμαστε σε θέση να το αξιοποιήσουμε το JSON αρχείο που δημιουργείται με τις μετά-πληροφορίες που περιέχει για την τροποποίηση, τον τερματισμό ή τον έλεγχο εγκυρότητας της περιγραφής που βλέπει ο χρήστης μέσω του γραφικού περιβάλλοντος σε σχέση με το τι υπάρχει στον πάροχο. Για κάθε περιγραφή, διαβάζεται το αντίστοιχο JSON αρχείο και δημιουργείται άμεσα το μετά-μοντέλο της περιγραφής στη μνήμη προσφέροντας άμεση δυνατότητα επεξεργασίας. Συγχρόνως, γίνεται έλεγχος εγκυρότητας. Δηλαδή αν οι οντότητες που περιγράφονται μέσα στο JSON αρχείο είναι όντως ενεργές στον πάροχο και ενημερώνεται ο χρήστης σε περίπτωση που κάποια από αυτές δεν είναι έγκυρη. Με αυτή τη διαδικασία, μπορεί να ξέρει ανά πάσα στιγμή τη κατάσταση της περιγραφής του και να προβεί σε αλλαγές ή ακόμη και σε μερικό ή ολικό τερματισμό.

(Σχήμα 3.7) JSON Μετά-πληροφορίες από αποτυχημένη ανάπτυξη

(Σχήμα 3.8) JSON Μετά-πληροφορίες από επιτυχημένη ανάπτυξη με δύο Instances στο πρώτο Module και ένα στο δεύτερο

3.4.4 API Εφαρμογής

Κλήση API	**Παράμετροι**	**Τύπος επιστροφής**	**Περιγραφή**
getFlavorList()	Map<String, String>	List<FlavorObj>	Επιστρέφει όλα τα διαθέσιμα Flavor
getImageList()	String scope, Map<String, String>	List<ImageObj>	Επιστρέφει όλα τα διαθέσιμα Images
getAdditionalServices()	String	List<String>	Επιστρέφει επιπλέον πληροφορίες που μπορεί να επισυναφθούν σε ένα VM σε μεταγενέστερο στάδιο
getQuotas()	String	List<String>	Επιστρέφει τους περιορισμούς στο λογαριασμό του χρήστη
getNetworks()	-	List<Instance>	Επιστρέφει τα διαθέσιμα δίκτυα
getKeyPairs()	-	List<KeyPairsObj>	Επιστρέφει τα διαθέσιμα Key

			pairs
getSecurityGroups()	-	List<SecurityGroupsObj>	Επιστρέφει τα διαθέσιμα Security groups
getInstances()	-	List<Instance>	Επιστρέφει τα ενεργά Virtual Instances
createImageFromInstance()	String, String	String	Δημιουργεί εικόνα από ένα Virtual Instance
createInstance()	Map<String, String>	String	Δημιουργεί ένα Virtual Instance
terminateInstance()	String	Boolean	Τερματίζει ένα Virtual Instance
OpenstackConnector()	HashMap<String, String>	-	Δημιουργεί τη σύνδεση με τον πάροχο
Deployer()	File	-	Κάνει την ανάπτυξη
cloudConnect()	-	OpenstackConnector	Δημιουργεί τη σύνδεση
createDeployment()	HashMap<String, String>	String	Δημιουργεί ένα Deployment
terminateDeployment()	String	Boolean	Τερματίζει ένα

			Deployment
createModule()	String, HashMap<String, String>	String	Δημιουργεί ένα Module
terminateModule	String, String	Boolean	Τερματίζει ένα Module
addInstanceToModule()	String, String	String	Προσθετει ένα VM σε ένα Module
removeInstanceFromModule()	String, String	Boolean	Αφαιρεί ένα VM από ένα Module
parseConfig()	-	-	Διαβάζει το αρχείο με τις ρυθμίσεις
addModule()	Module	-	Προσθέτει ένα Module σε ένα Deployment
removeModule()	String	-	Αφαιρεί ένα Module σε ένα Deployment
JSON()	-	-	Δημιουργεί το JSON αρχείο
write_data()	HashMap[], String, String, List<String>,	-	Γράφει στο JSON αρχείο

	List<String>, String, String		
init()	-	-	Δημιουργεί το γραφικό περιβάλλον
parseXMLFile()	String	HashMap[]	Διαβάζει το TOSCA XML

Πίνακας 3.1 API εφαρμογής

Κεφάλαιο 4

Αξιολόγηση

4.1 Αξιολόγηση του Deployer

Στο κεφάλαιο αυτό, θα γίνει μία λεπτομερής αναφορά στην διαδικασία που χρειάζεται για να αναπτυχθεί μία περιγραφή στον πάροχο. Τα επιμέρους βήματα που πρέπει να γίνουν σε κάθε στάδιο από την πλευρά του χρήστη, το τελικό αποτέλεσμα σε μια επιτυχής ανάπτυξη και τι γίνεται σε περίπτωση σφάλματος.

Για σκοπούς παραδείγματος, θεωρούμε μία 3-tier εφαρμογή από κάποιο OnLine Shop η οποία αποτελείται από 3 Virtual Instances. Δύο Virtual Instances με την ονομασία Web Server όπου θα έχουν τον ρόλο του εξυπηρετητή της ιστοσελίδας τους καταστήματος καθώς και την ενσωματωμένη εφαρμογή για τις αγοραπωλησίες μέσω στις ιστοσελίδας. Επιπλέον, ένα ακόμη Virtual Instance με την ονομασία DB Server όπου θα έχει την βάση δεδομένων για τη λειτουργία των εξυπηρετητών αυτών και την επεξεργασία των δεδομένων που στέλνουν..

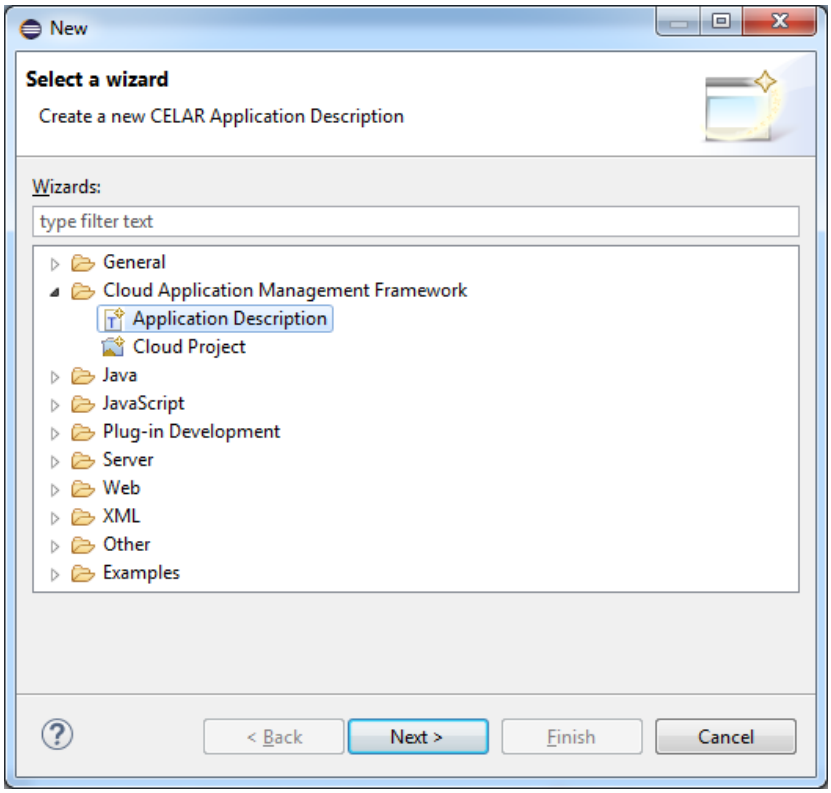
Οι δύο Web Server αποτελούνται από τα εξής χαρακτηριστικά:

- VMI: Ubuntu 14.04
- Flavor: m1.small
- Keypair: akasta
- Network: CAMF-NET

Ο DB Server αποτελούνται από τα εξής χαρακτηριστικά:

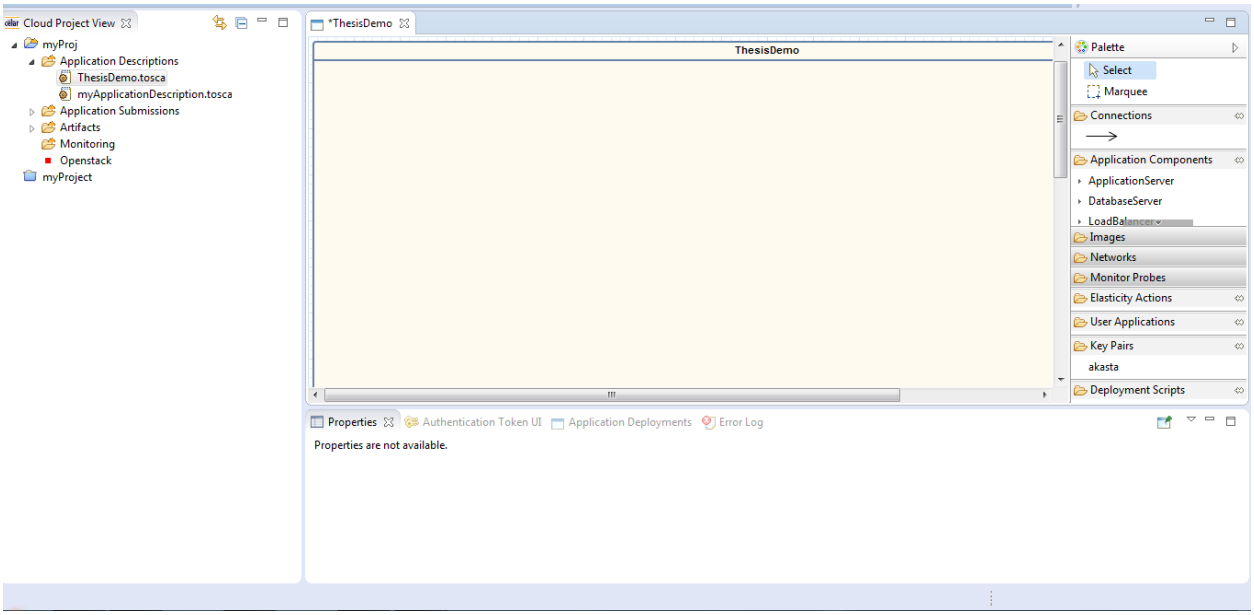
- VMI: CentOS 6.4
- Flavor: m1.medium
- Keypair: akasta
- Network: CAMF-NET

Αφού ο χρήστης έχει δημιουργήσει το σχετικό Project μέσω του γραφικού περιβάλλοντος που προσφέρει το CAMF, βλέπει την προεπιλεγμένη οθόνη και πρέπει να προβεί στη δημιουργία μίας νέας περιγραφής μέσω του οδηγού όπως φαίνεται στο σχήμα 4.1



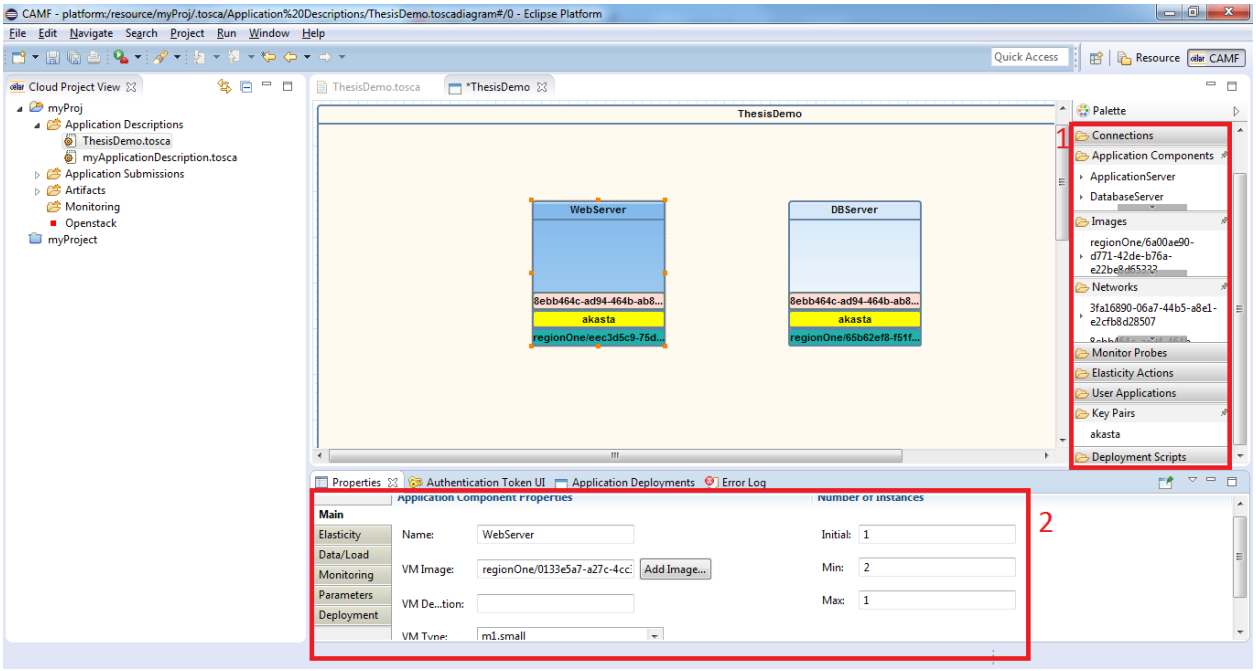
Σχήμα 4.1 Δημιουργία Project

Έπειτα, εμφανίζεται μία κενή περιγραφή όπως φαίνεται στο σχήμα 4.2 και ο χρήστης πρέπει να κάνει χρήση της παλέτας που προσφέρει το CAMF για τη δημιουργία της περιγραφής του. Στα αριστερά της εικόνας, φαίνεται και η δομή των φακέλων που υπάρχει. Οι περιγραφές βρίσκονται συγκεντρωμένες σε έναν φάκελο με την ονομασία Application Descriptions, η μετάφραση και το μοντέλο των περιγραφών στον φάκελο Application Submissions και οι επιπλέον οντότητες όπως εικόνες του χρήστη, deployment scripts, Json Files κλπ συγκεντρωμένες στους αντίστοιχους φακέλους τους στον φάκελο Artifacts.



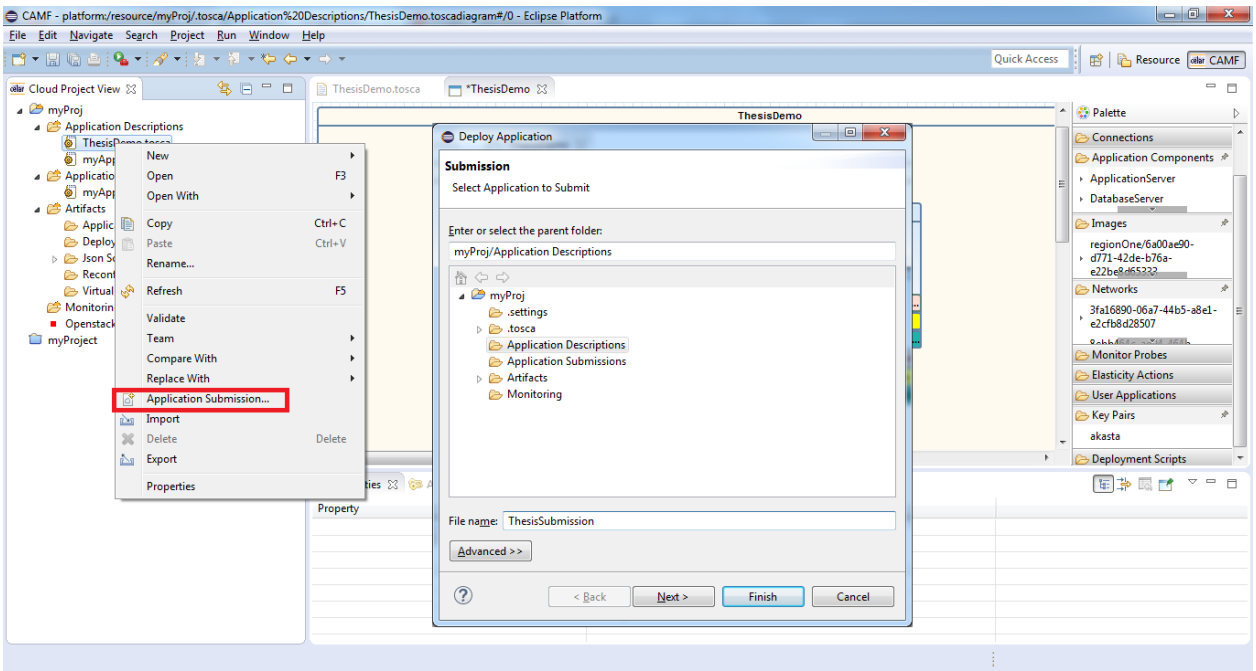
Σχήμα 4.2 Αρχική οθόνη Project

Η παλέτα που προσφέρει το CAMF, φορτώνει αμέσως τα διαθέσιμα χαρακτηριστικά - επιλογές που έχουν οριστεί στον πάροχο και με drag'n'drop από τα αντίστοιχα πεδία αξιοποιούμε την κάθε επιλογή. Χρησιμοποιώντας το πεδίο Application Components (σχ. 4.3.1) επιλέγουμε ένα Web Server και έναν Database Server τα οποία "ρίχνουμε" στην παλέτα. Για κάθε Component-κουτάκι που ορίσαμε, προσθέτουμε την εικόνα (Image) την οποία μπορούμε να επιλέξουμε μέσα από τις διαθέσιμες επιλογές του παρόχου είτε να ανεβάσουμε εμείς όποια επιθυμούμε όπως φαίνεται στο σχήμα 4.3.2. Επιλέγουμε επίσης το δίκτυο που θα ανήκει το Virtual Instance καθώς και το Keypair από τις διαθέσιμες επιλογές σύμφωνα με αυτά που προσφέρει ο πάροχος και τα δικαιώματα που έχουμε. Έπειτα, επιλέγοντας το κάθε Component μπορούμε να ορίσουμε επιπλέον λεπτομέρειες Όπως φαίνονται και στο σχήμα 4.3 έχουμε τις βασικές επιλογές όπως το όνομα του, την εικόνα του, το flavor, τα ελάχιστα και τα μέγιστα Virtual Instances που θα υπάρχουν, επιλογές ελαστικότητας (elasticity) όπου ορίζουμε κάποιες συνθήκες και τι πρέπει να γίνεται όταν επιτευχθούν για παράδειγμα, επιλογές Monitoring, επιπλέον παράμετροι για το VM και τέλος ορισμό επιπλέον Deployment Script που θα εκτελεστούν κατά την ανάπτυξη του Virtual Instance. Στο δικό μας παράδειγμα θα μας απασχολήσει μόνο η επιλογή Main και συγκεκριμένα ο αρχικός αριθμός των Virtual Instances που πρέπει να αναπτυχθούν για τον WebServer τον οποίο θέσαμε σε δύο όπως φαίνεται στο σχήμα 4.3.2. Επίσης, μέσω των συγκεκριμένων επιλογών, ορίζουμε και το Flavor του Virtual Instance που όπως φαίνεται έχει οριστεί σε m1.small.



Σχήμα 4.3 Περιγραφή εφαρμογής

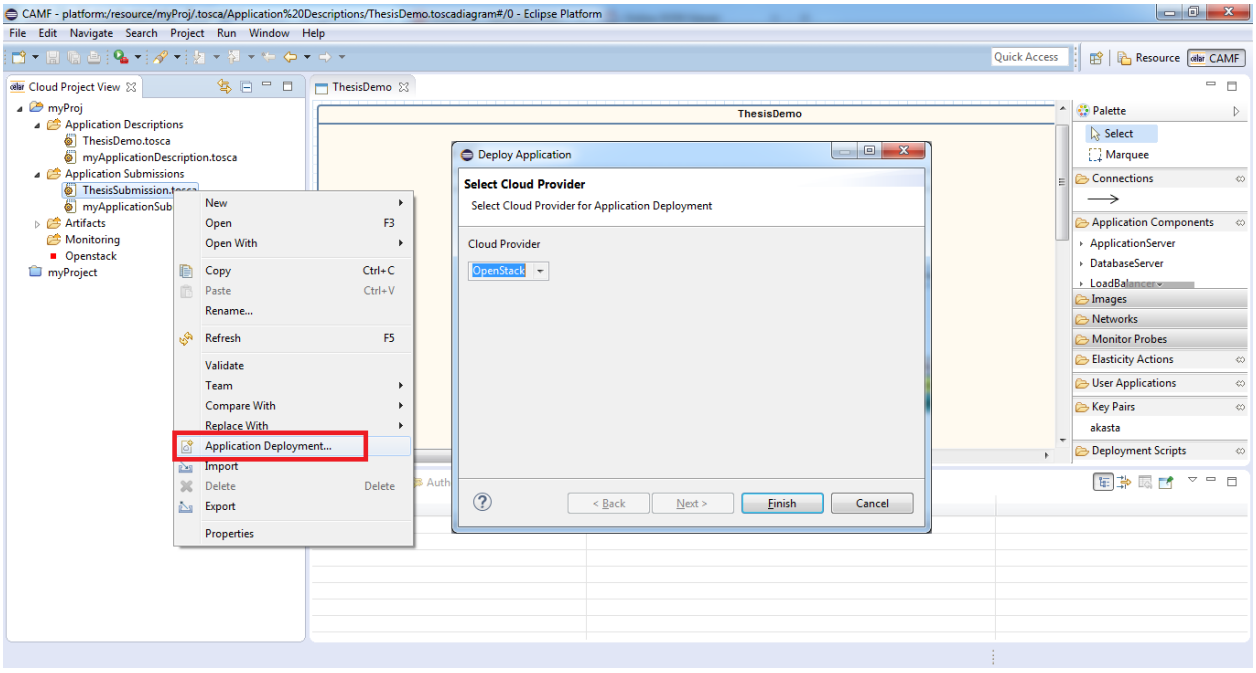
Αφού έχει ολοκληρωθεί η διαδικασία της περιγραφής της εφαρμογής, το επόμενο βήμα είναι η δημιουργία του TOSCA μοντέλου της περιγραφής και η περαιτέρω αξιοποίησή του όπως φαίνεται στο σχήμα 4.4 & 4.5.



Σχήμα 4.4 Δημιουργία TOSCA

```
<?xml version="1.0" encoding="UTF-8"?>
<tosca:Definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:elasticity="http://www.example.org/NewXMLSchema" xmlns:sybl="http://www.example.org/SYBL" xmlns:to
  <tosca:ServiceTemplate xsi:type="elasticity:TServiceTemplateExtension" id="hello" name="ThesisDemo">
    <tosca:BoundaryDefinitions xsi:type="elasticity:TBoundaryDefinitionsExtension">
      <tosca:Properties>
        <elasticity:ServiceProperties>
          <elasticity:Version>1.0</elasticity:Version>
        </elasticity:ServiceProperties>
      </tosca:Properties>
    </tosca:BoundaryDefinitions>
    <tosca:TopologyTemplate>
      <tosca:NodeTemplate xsi:type="elasticity:TNodeTemplateExtension" id="C997144673" maxInstances="1" minInstances="2" name="WebServer" initInstances="1" x="235" y="105">
        <tosca:Properties>
          <elasticity:NodeProperties>
            <elasticity:Flavor>m1.small</elasticity:Flavor>
          </elasticity:NodeProperties>
        </tosca:Properties>
        <tosca:DeploymentArtifacts>
          <tosca:DeploymentArtifact artifactRef="AppServerImage" artifactType="VMI" name="regionOne/0133e5a7-a27c-4cc3-9c78-6b6127b4a440"/>
          <tosca:DeploymentArtifact artifactType="Network" name="8ebb464c-ad94-464b-ab87-28cfc46d9ecb"/>
          <tosca:DeploymentArtifact artifactType="KeyPair" name="akasta"/>
          <tosca:DeploymentArtifact artifactType="Network" name="8ebb464c-ad94-464b-ab87-28cfc46d9ecb"/>
          <tosca:DeploymentArtifact artifactRef="WebServerImage" artifactType="VMI" name="regionOne/eec3d5c9-75d0-4a95-a66b-b08f9a963bc0"/>
        </tosca:DeploymentArtifacts>
      </tosca:NodeTemplate>
      <tosca:NodeTemplate xsi:type="elasticity:TNodeTemplateExtension" id="C722278586" maxInstances="1" minInstances="1" name="DBServer" initInstances="1" x="515" y="105">
        <tosca:Properties>
          <elasticity:NodeProperties>
            <elasticity:Flavor>m1.medium</elasticity:Flavor>
          </elasticity:NodeProperties>
        </tosca:Properties>
        <tosca:DeploymentArtifacts>
          <tosca:DeploymentArtifact artifactType="Network" name="8ebb464c-ad94-464b-ab87-28cfc46d9ecb"/>
          <tosca:DeploymentArtifact artifactType="KeyPair" name="akasta"/>
          <tosca:DeploymentArtifact artifactRef="DBServerImage" artifactType="VMI" name="regionOne/65b62ef8-f51f-4eac-95e5-fd3a473c7efd"/>
        </tosca:DeploymentArtifacts>
      </tosca:NodeTemplate>
    </tosca:TopologyTemplate>
```

Σχήμα 4.5 Περιγραφή TOSCA



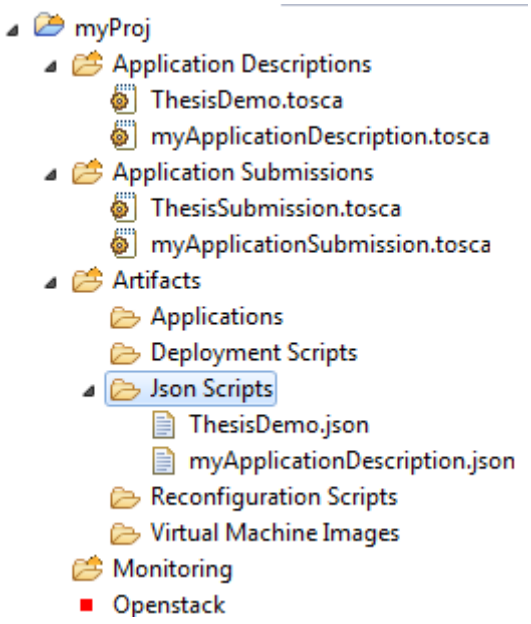
Σχήμα 4.6 Αποστολή ανάπτυξης στον πάροχο

Αφού ολοκληρώθηκε η ανάπτυξη της περιγραφής, όπως φαίνεται και στο σχήμα 4.7 οι οντότητες που περιγράφηκαν έχουν αναπτυχθεί στον πάροχο (OpenStack) και είναι έτοιμες προς χρήση. Συγχρόνως, όπως φαίνεται στο σχήμα 4.8 έχει δημιουργηθεί και το αντίστοιχο

JSON αρχείο στον αντίστοιχο φάκελο του Project με το αποτέλεσμα και τις λεπτομέρειες της περιγραφής.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Uptime	Actions
DBServer_1	centos6.4x86_64	192.168.0.100	m1.medium | 4GB RAM | 2 VCPU | 40.0GB Disk	akasta	Active	nova	None	Running	0 minutes	Create Snapshot | More
WebServer_2	ubuntu14.04	192.168.0.110	m1.medium | 4GB RAM | 2 VCPU | 40.0GB Disk	akasta	Active	nova	None	Running	2 minutes	Create Snapshot | More
WebServer_1	ubuntu14.04	192.168.0.109	m1.medium | 4GB RAM | 2 VCPU | 40.0GB Disk	akasta	Active	nova	None	Running	1 minute	Create Snapshot | More

Σχήμα 4.7 Προβολή ανάπτυξης στον πάροχο



Σχήμα 4.8 Δομή Project - Json Αρχείο

Σχήμα 4.9 Αποτέλεσμα περιγραφής στο Json αρχείο

Το παραπάνω σενάριο, ήταν μία επιτυχής ανάπτυξη καθώς δεν υπήρχε κάποιο σφάλμα στην όλη διαδικασία. Η πολιτική που επιλέχθηκε για την ανάπτυξη, ήταν η **CONTINUE_ON_ERROR** που σε περίπτωση σφάλματος, ο Deployer θα το αγνοούσε και θα προσπαθούσε να συνεχίσει την ανάπτυξη.

Ας δούμε τώρα, τι θα γινόταν στην περίπτωση που ο χρήστης είχε επιλέξει κάποιο λάθος Flavor για το Module DBServer και είχε ορίσει ως πολιτική ανάπτυξης την **TERMINATE_AND_REVERT_ON_ERROR.** Η όλη διαδικασία, παραμένει ακριβώς ίδια όπως πριν. Η μόνη διαφορά είναι στο τελικό JSON που θα δημιουργηθεί και ότι τα δύο Virtual Instances από το Component WebServer αφού δημιουργηθούν, θα τερματιστούν βάση της πολιτικής που όρισε.

Όπως φαίνεται και στο σχήμα 4.9 τα δύο πρώτα Virtual Instances από το Component WebServer, δημιουργήθηκαν επιτυχώς. Στην εξέλιξη της περιγραφής και καθώς προέκυψε το σφάλμα στο δεύτερο Component, έγινε τερματισμός της ανάπτυξης, εμφανίστηκε το ανάλογο μήνυμα στην οθόνη προς ενημέρωση του χρήστη (σχ.4.10) και τέλος, δημιουργήθηκε το JSON αρχείο ενημερώνοντάς τον για το πρόβλημα.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Uptime	Actions
WebServer_2	ubuntu14.04	192.168.0.103	m1.medium | 4GB RAM | 2 VCPU | 40.0GB Disk	akasta	Active	nova	None	Running	0 minutes	Create Snapshot | More
WebServer_1	ubuntu14.04	192.168.0.102	m1.medium | 4GB RAM | 2 VCPU | 40.0GB Disk	akasta	Active	nova	None	Running	0 minutes	Create Snapshot | More

Σχήμα 4.10 Επιτυχής ανάπτυξη WebServer

```
Starting of JSON creation...
Finished JSON creation...
DEPLOY Exception: Failed to deploy. Please contact your system administrator.
```

Σχήμα 4.11 Μήνυμα σφάλματος προς τον χρήστη

Instances

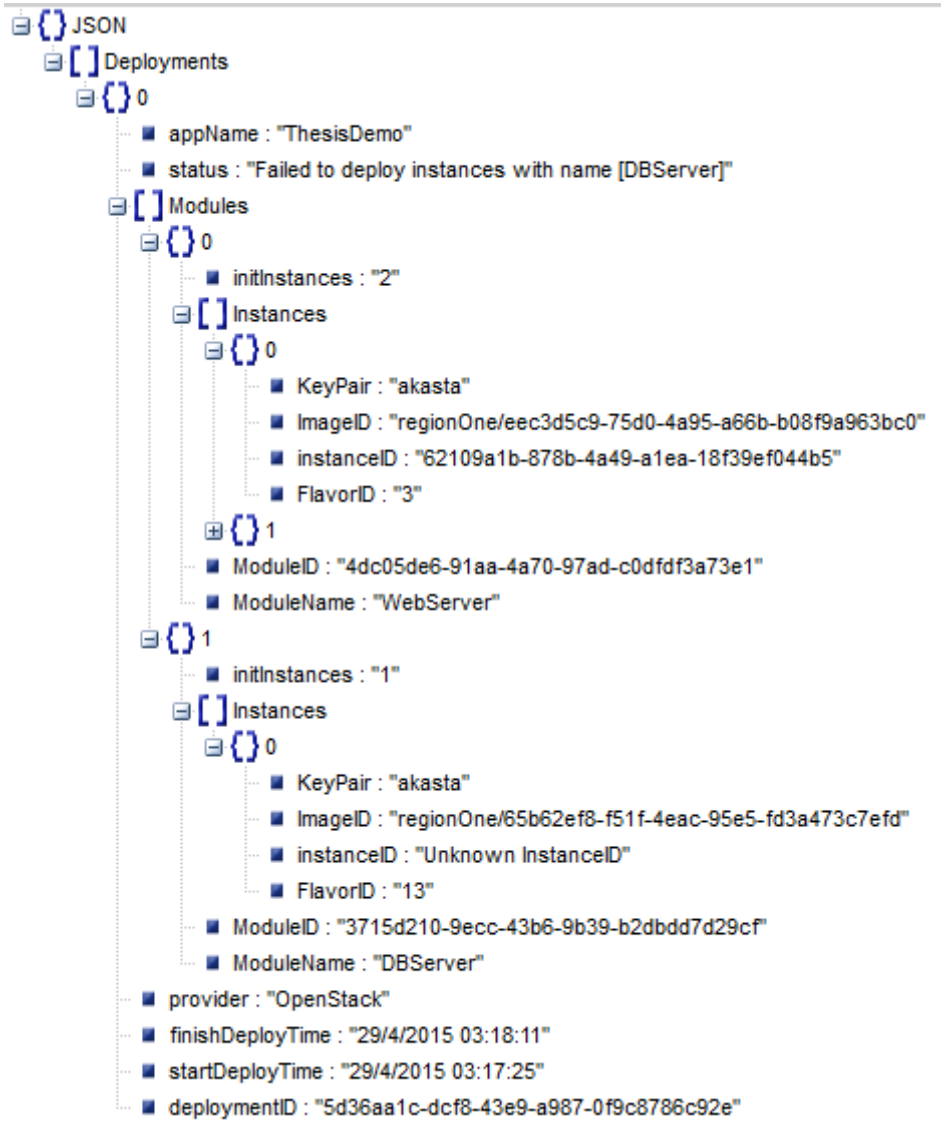
Instances

Filter | Filter | + Launch Instance | Soft Reboot Instances | Terminate Instances

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Uptime	Actions
Chef Server	ubuntu-12.4-x86_64	192.168.0.106 10.16.3.155	m1.medium | 4GB RAM | 2 VCPU | 40.0GB Disk	id_dsa	Active	nova	None	Running	5 months, 2 weeks	Create Snapshot | More

Displaying 1 item

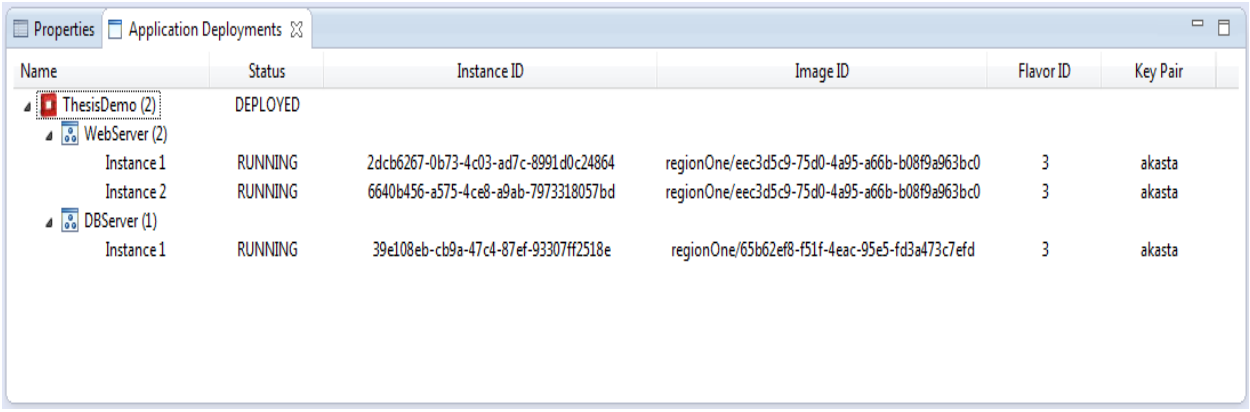
Σχήμα 4.12 Ενεργά Virtual Instances στον πάροχο μετά τον αυτόματο τερματισμό



Σχήμα 4.13 JSON αρχείο με το αποτέλεσμα της ανεπιτυχής ανάπτυξης

Και στα δύο σενάρια που περιγράφηκαν και εκτελέστηκαν πιο πάνω, τα αποτελέσματα ήταν τα αναμενόμενα. Στο πρώτο σενάριο όπως ειπώθηκε, το αποτέλεσμα ήταν επιτυχής. Τα Virtual Instances δημιουργήθηκαν άμεσα όπως επίσης και το αντίστοιχο JSON αρχείο για την περαιτέρω αξιοποίηση του όπως θα δούμε παρακάτω. Στο δεύτερο σενάριο της ανεπιτυχής ανάπτυξης λόγω του λανθασμένου Flavor ID και εδώ η συμπεριφορά του Deployer ήταν η αναμενόμενη. Δημιουργήθηκαν κανονικά τα δύο πρώτα Virtual Instances όπως φαίνεται στο σχήμα 4.10, στη πορεία και την στιγμή που προέκυψε το σφάλμα, ο χρήστης ενημερώθηκε για αυτό (σχ. 4.11), έγινε διακοπή της ανάπτυξης και τερματισμός των Virtual Instance που δημιουργήθηκαν επιτυχώς (σχ. 4.12) λόγω της πολιτικής ανάπτυξης που επιλέχθηκε και τέλος, δημιουργήθηκε το JSON αρχείο που περιγράφει όπως βλέπουμε στο σχήμα 4.13 στο πεδίο status το πρόβλημα που προέκυψε.

Όπως περιγράφηκε και στα προηγούμενα κεφάλαια, η δημιουργία του JSON αρχείου γίνεται με σκοπό την περαιτέρω αξιοποίησή του τόσο από τον χρήστη όσο και από την ίδια την εφαρμογή. Πιο συγκεκριμένα, στο σχήμα 4.14 προβάλλονται οι λεπτομέρειες της επιτυχής περιγραφής που προέκυψε πιο πάνω ενημερώνοντας τον χρήστη για την κατάσταση της αφού ο ίδιος έχει κλείσει και εκκινήσει πάλι την εφαρμογή του CAMF.



Σχήμα 4.14 Κατάσταση ανάπτυξης μετά από επανεκκίνηση της εφαρμογής

Κεφάλαιο 5

Συμπεράσματα και Μελλοντική Ανάπτυξη

5.1 Συμπεράσματα

Στην αρχική φάση της ΑΔΕ, είχε γίνει έρευνα για διάφορα εργαλεία τα οποία χρησιμοποιούνται για τον σχεδιασμό διαδραστικών γραφικών περιβάλλοντων στο διαδίκτυο. Έγινε έρευνα διαφόρων βιβλιοθηκών και των λειτουργιών τους όπως η jQuery και η jQueryUI που προσφέρουν αυτή τη δυνατότητα. Επίσης, ασχολήθηκα με κάποια έτοιμα Framework που προσφέρουν έτοιμες λύσεις για τέτοιου είδους περιβάλλοντα. Μερικά από αυτά ειναι το ExtJS (Sencha) και το SmartGWT(SmartClient). Τα οποία είναι σχεδιασμένα με τη χρήση Javascript και jQuery. Ήταν ένα πεδίο που είχα λίγες γνώσεις οι οποίες όμως βελτιώθηκαν μέσα από την έρευνα και τις δοκιμές ως προς την χρήση τους.

Ύστερα από την αλλαγή θέματος που προέκυψε, έγινε εκτενής έρευνα για το τι είναι το Cloud, ποια τα χαρακτηριστικά του και οι δυνατότητες του, πως χρησιμοποιείται και για ποιους λόγους καθώς και ποιες είναι οι λειτουργίες του όπως αυτές αναφέρθηκαν και στο δεύτερο κεφάλαιο. Ένα πεδίο που δεν δίνεται τόση έμφαση στα πλαίσια των μαθημάτων του πανεπιστημίου και λίγο πολύ άγνωστος τομέας. Στη πορεία, αναλύθηκαν οι διαθέσιμες επιλογές που υπάρχουν για τα AMF που είναι διαθέσιμα τόσο στο εμπορικό όσο και ακαδημαϊκό τομέα και ποια είναι τα χαρακτηριστικά τους και οι διαφορές τους. Όπως επίσης και όροι άγνωστοι πριν αυτή την ενασχόληση όπως η γλώσσα TOSCA, η ελαστικότητα στις εφαρμογές και η έννοια του Application Management Framework (AMF).

Σημαντικό κομμάτι της ΑΔΕ ήταν η έμφαση που δόθηκε στο Eclipse IDE και πιο συγκεκριμένα τη δημιουργία plug-in για αυτό και η επέκταση των ήδη υπαρχόντων. Δόθηκε έμφαση στο μοντέλο αντικειμένων που χρησιμοποιεί το Eclipse (EMF) καθώς και στην σωστή οργάνωση και δημιουργία του κώδικα. Πτυχές που εν μέρη είχαν καλυφθεί στα πλαίσια του μαθήματος ΕΠΛ463: Επαναχρησιμοποίηση Λογισμικού. Έγινε μελέτη της βιβλιοθήκης jClouds όπου μας έδωσε τη δυνατότητα για την επικοινωνία μεταξύ της

εφαρμογής μας και του παρόχου. Αν και δόθηκε έμφαση σε έναν μόνο πάροχο (Openstack), πλέον υπάρχει η τεχνογνωσία για τη χρήση της με σκοπό την επέκταση της εφαρμογής και στους υπόλοιπους υποστηριζόμενους παρόχους.
Τέλος, ήταν η δεύτερη φορά που ασχολήθηκα με μία τόσο μεγάλη εφαρμογή και μου έδωσε για άλλη μία φορά τη δυνατότητα χρήσης επιπλέον λειτουργιών όπως το Github για την καταγραφή του κώδικα αλλά και εμπειρία στον χρονοπρογραμματισμό του έργου όπως και επιπλέον εμπειρία στη δημιουργία μεγάλων εφαρμογών.

5.2 Μελλοντική Ανάπτυξη

Τόσο στην αυτόνομη εφαρμογή όσο και στα κομμάτια που ενσωματώθηκαν στο περιβάλλον του CAMF, υπάρχει η πρόνοια για περαιτέρω λειτουργίες χωρίς να υπάρχει η ανάγκη για αλλαγές στη δομή των υπαρχόντων υλοποιήσεων. Μερικές από αυτές τις λειτουργίες είναι η επεκτασιμότητα σε επιπλέον παρόχους όπως το Amazon, RackSpace, Docker κλπ. καθώς το μόνο επιπλέον που πρέπει να υλοποιηθεί είναι η κλάση που αξιοποιεί τις συναρτήσεις του jClouds μέσω του Interface που έχει οριστεί και η σύνδεση με τον πάροχο. Όπως επίσης και η υποστήριξη ελαστικών απαιτήσεων στον υπάρχων Deployer και σε όσους υλοποιηθούν στην πορεία.

Βιβλιογραφία

[1] [Binz, 2013] Binz, T., Breitenbucher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: Open- TOSCA - A Runtime for TOSCA-Based Cloud Applications. In: ICSOC. Volume 8274 of Lecture Notes in Computer Science., Springer (2013) 692–695

[2] [Kopp, 2013] Kopp, O., Binz, T., Breitenbucher, U., Leymann, F.: Winery: A Modeling Tool for TOSCA- Based Cloud Applications. In: Service-Oriented Computing. Volume 8274 of Lecture Notes in Computer Science. Springer (2013) 700–704

[3] [Juve, 2011] Juve, G., Deelman, E.: Automating Application Deployment in Infrastructure Clouds. In: Proceedings of the 2011 IEEE 3rd International Conference on Cloud Computing Technology and Science, IEEE Computer Society (2011) 658–665

[4] [Sofokleous, 2014] C. Sofokleous and N. Loulloudes and D. Trihinas and G. Pallis and M. Dikaiakos, c-Eclipse: An Open-Source Management Framework for Cloud Applications, (EuroPar 2014), Porto, Portugal 2014

[5] [Trihinas, 2014] D. Trihinas, G. Pallis, and M. D. Dikaiakos. JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud. In 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2014.

Παράρτημα Α

Στο παράρτημα αυτό θα παρατεθεί ο κώδικας που προσκομίζει τις διαθέσιμες πληροφορίες που υπάρχουν στον πάροχο, τον κώδικα για την δημιουργία και τον τερματισμό Virtual Instances και ο κώδικας διαχείρισης Exception.

Οι πηγαίοι κώδικες τόσο για την Standalone εφαρμογή όσο και για τον CAMF μπορούν να βρεθούν στα εκάστοτε Repository.

- https://git.eclipse.org/c/camf/
- https://github.com/UCY-LINC-LAB/CloudServiceDeployer

Προσκόμιση ενεργών Virtual Instances

```
public class OpenStackOpDescribeInstances extends AbstractOpenStackOpInstances {

    private final ComputeService computeAPI;

    /**
     * Creates a new {@link OpenStackOpDescribeInstances} to fetch available
     * data.
     *
     */

    public OpenStackOpDescribeInstances() {
        this.computeAPI = OpenStackClient.getInstance().getComputeService();
    }

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {

            Set<? extends ComputeMetadata> instances = this.computeAPI
                    .listNodes();

            setResult(new ArrayList<ComputeMetadata>(instances));
        } catch (Exception ex) {
            setException(ex);
        }
    }
}
```

Προσκόμιση διαθέσιμων Flavor

```
public class OpenStackOpDescribeFlavors extends AbstractOpenStackOpFlavors {

    private final FlavorApi flavorApi;

    /**
     * Creates a new {@link OpenStackOpDescribeFlavors} to fetch available data
     *
     *
     */

    public OpenStackOpDescribeFlavors() {
        this.flavorApi = OpenStackClient.getInstance().getFlavorApi();
    }

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {

            Set<Flavor> flavors = this.flavorApi.listInDetail().concat()
                    .toSet();

            setResult(new ArrayList<Flavor>(flavors));
        } catch (Exception ex) {
            setException(ex);
        }

    }

}
```

Προσκόμιση διαθέσιμων Εικόνων

```
public class OpenStackOpDescribeImages extends
        AbstractOpenStackOpDescribeImages {

    private final ComputeService computeService;

    /**
     * Creates a new {@link OpenStackOpDescribeImages} with the given owners as
     * parameter.
     *
     * @param computeService
     *            the {@link ComputeService} to obtain data from
     */
    public OpenStackOpDescribeImages(final ComputeService computeService) {
        this.computeService = computeService;
    }

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {

            Set<Image> images = (Set<Image>) this.computeService.listImages();
            setResult(new ArrayList<Image>(images));
        } catch (Exception ex) {
            setException(ex);
        }

    }

}
```

Προσκόμιση διαθέσιμων δικτύων

```
public class OpenStackOpDescribeNetworks extends
        AbstractOpenStackOpDescribeNetworks {

    private final NetworkApi networkApi;

    /**
     * Creates a new {@link OpenStackOpDescribeNetworks} to fetch available
     * networks.
     *
     */

    public OpenStackOpDescribeNetworks() {
        this.networkApi = OpenStackClient.getInstance().getNetworkApi();
    }

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {
            Set<Network> netSet = this.networkApi.list().concat().toSet();
            setResult(new ArrayList<Network>(netSet));
        } catch (Exception ex) {
            setException(ex);
        }
    }

}
```

Προσκόμιση διαθέσιμων Key pair

```
public class OpenStackOpDescribeKeyPairs extends AbstractOpenStackOpKeyPairs {

    private final KeyPairApi keyApi;

    /**
     * Creates a new {@link OpenStackOpDescribeKeyPairs} to fetch available data
     *
     */

    public OpenStackOpDescribeKeyPairs() {
        this.keyApi = OpenStackClient.getInstance().getKeyPairApi();
    }

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {

            Set<KeyPair> keypairs = this.keyApi.list().toSet();

            setResult(new ArrayList<KeyPair>(keypairs));
        } catch (Exception ex) {
            setException(ex);
        }

    }

}
```

Προσκόμιση διαθέσιμων Security Groups

```
public class OpenStackOpDescribeSecurityGroups extends
        AbstractOpenStackOpSecurityGroups {

    private final SecurityGroupApi SecurityGroupApi;

    /**
     * Creates a new {@link OpenStackOpDescribeSecurityGroups} to fetch
     * available data
     *
     */

    public OpenStackOpDescribeSecurityGroups() {
        this.SecurityGroupApi = OpenStackClient.getInstance()
                .getSecurityGroupApi();
    }

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {

            Set<SecurityGroup> SecurityGroups = this.SecurityGroupApi.list()
                    .toSet();

            setResult(new ArrayList<SecurityGroup>(SecurityGroups));
        } catch (Exception ex) {
            setException(ex);
        }
    }
}
```

Δημιουργία εικόνας από Virtual Instance

```
public String createImageFromInstance(String imageName, String instanceID) {
        ServerApi serverAPI = this.novaAPI
                        .getServerApiForZone(OpenstackConnector.DEFAULT_REGION);

        String imageID = serverAPI.createImageFromServer(imageName, instanceID);
        return imageID;
}
```

Δημιουργία Virtual Instance

```
public class OpenStackOpDeployApplication extends
        AbstractOpenStackOpDeployApplication {

    private final TOSCAResource toscaResource;
    private final ICloudProject project;

    public enum ActionToDo {
        CONTINUE_ON_ERROR, STOP_ON_ERROR, TERMINATE_AND_REVERT_ON_ERROR
    };

    public String depID;
    public HashMap<String, Deployment> deployments = new HashMap<String, Deployment>();
    public List<String> modules = new ArrayList<String>();
    public List<String> instances = new ArrayList<String>();
    private ActionToDo ActiontoDo;
    private HashMap<String, String> params = new HashMap<String, String>();
    private JSON json = new JSON();;
    // TODO - Hack for Nephelae. Only one region there.
    // Fix it for other providers with more zones.
    private String zone = "regionOne";
    public List<String> failed_to_deploy = new ArrayList<String>();

    private NovaApi nova = OpenStackClient.getInstance().getNova();
    private KeyPairApi keyPairApi = nova.getKeyPairExtensionForZone(this.zone)
            .get();
    private ServerApi serverApi = nova.getServerApiForZone(this.zone);

    public OpenStackOpDeployApplication(final OpenStackClient client,
            final TOSCAResource tosca) {
        this.toscaResource = tosca;
        this.project = this.toscaResource.getProject();
    }

    public OpenStackOpDeployApplication(final OpenStackClient client,
            final TOSCAResource tosca, ActionToDo action) {
        this.toscaResource = tosca;
        this.project = this.toscaResource.getProject();
```

```
        this.ActiontoDo = action;
    }

    /*
     * (non-Javadoc)
     *
     * @see
     * org.eclipse.camf.connectors.aws.operation.AbstractEC2OpDeployApplication
     * #run()
     */

    @Override
    public void run() {
        setResult(null);
        setException(null);
        try {
            // set Start time of the deployment
            String start_time = new SimpleDateFormat("dd/M/yyyy hh:mm:ss")
                    .format(Calendar.getInstance().getTime());

            TOSCAModel toscaModel = this.toscaResource.getTOSCAModel();
            TServiceTemplate serviceTemplate = toscaModel.getServiceTemplate();
            List<HashMap<String, String>> xmlinfo = new ArrayList<HashMap<String, String>>();
            HashMap<String, String> array = null;

            if (serviceTemplate != null) {
                TTopologyTemplate topologyTemplate = serviceTemplate
                        .getTopologyTemplate();

                // get AppName and create the Deployment object
                this.depID = this.createDeployment(serviceTemplate.getName());
                System.out.println("Created new Deployment with depID: "
                        + this.depID);

                if (topologyTemplate != null) {
                    EList<TNodeTemplate> nodeTemplateList = topologyTemplate
                            .getNodeTemplate();
                    if (nodeTemplateList != null) {
```

```
if (nodeTemplateList != null) {
    // Instantiate each node in the topology
    for (TNodeTemplate node : nodeTemplateList) {

        String modID = this.createModule(this.depID,
                node.getName());
        this.modules.add(modID);
        System.out
                .println("Created new module with modID: "
                        + modID + ", name: "
                        + node.getName());

        int minCount = node.getMinInstances();
        int maxCount = ((BigInteger) node.getMaxInstances())
                .intValue();

        PropertiesType properties = node.getProperties();
        String nodeFlavor = null;

        FeatureMap propFeatMap = properties.getAny();
        for (Entry entry : propFeatMap) {
            if (entry instanceof NodePropertiesType)
                nodeFlavor = ((NodePropertiesType) entry)
                        .getFlavor();
        }

        // Get Deployment Artifacts
        TDeploymentArtifacts deploymentArtifacts = node
                .getDeploymentArtifacts();

        String amiID = null;
        String flavorID = null;
        String keyPairArtifact = null;
        String keypairName = null;
        KeyPair keyPair = null;
        boolean keyPairExists = false;
        String networks = null;
        String script = null;
        String vID = null;
```

```
for (TDeploymentArtifact artifact : deploymentArtifacts
        .getDeploymentArtifact()) {

    array = new HashMap<String, String>();

    // Find the VMI Keypair, Network
    // and Script artifacts
    String artifactType = artifact
            .getArtifactType().toString();
    if (artifactType.equals("VMI")) { //$NON-NLS-1$
        amiID = artifact.getName();
    } else if (artifactType.equals("KeyPair")) { //$NON-NLS-1$
        keyPairArtifact = artifact.getName();
    } else if (artifactType.equals("Network")) {
        networks = artifact.getName();
    } else if (artifactType.equals("Script")) {
        script = artifact.getName();
    }

    if (keyPairArtifact != null) {
        String encodedPublicKey = importKeyPair(
                keyPairArtifact, this.project);

        // Strip extension. - Get file name only
        if (encodedPublicKey != null) {
            keypairName = keyPairArtifact
                    .substring(0, keyPairArtifact
                            .indexOf(".")); //$NON-NLS-1$
        } else {
            keypairName = keyPairArtifact;
        }
        keyPairExists = keyPairExists(
                this.keyPairApi, keypairName);
        if (!keyPairExists) {
            keyPair = this.keyPairApi
                    .createWithPublicKey(
                            keypairName,
```

```
                                encodedPublicKey);
            }
        }

        if (nodeFlavor != null) {
            flavorID = flavorRefForRegion(this.nova,
                    this.zone);
        } else {
            flavorID = "3";
        }

        String statements = null;
        if (script != null) {
            statements = importScript(script,
                    this.project);
        }
    }

    CreateServerOptions sv;
    if (networks != null)
        sv = CreateServerOptions.Builder.adminPass(
                "test").networks(networks); //$NON-NLS-1$
    else
        sv = CreateServerOptions.Builder
                .adminPass("test"); //$NON-NLS-1$

    if (!keyPairExists) {
        if (keyPair != null) {
            sv.keyPairName(keypairName);
        }
    } else {
        sv.keyPairName(keypairName);
    }

    for (int i = 1; i <= minCount; i++) {
        this.params.put("name", node.getName() + "_"
                + i);
        this.params.put("amiID", amiID);
```

```
    this.params.put("flavor", flavorID);

    try {
        vID = this.addInstanceToModule(this.depID,
                modID, this.params, sv);
    } catch (Exception ex) {
        try {
            throw new OpenStackOpExceptions(
                    "Failed to deploy. Please contact your system administrator.",
                    OpenStackOpExceptions.ExceptionType.DEPLOY);
        } catch (OpenStackOpExceptions CustomExcpetion) {
            this.setException(ex);
            System.err.println(CustomExcpetion
                    .getLocalizedMessage());
        }
    }
    if (vID == null) {
        this.failed_to_deploy.add(node.getName());
        this.instances.add("Unknown InstanceID");
        if (!ActiontoDo
                .equals(ActionToDo.CONTINUE_ON_ERROR))
            break;
    } else {
        this.instances.add(vID);
    }
    System.out
            .println("Added new Instance to module with id: "
                    + vID);
}
// save info for later use on JSON creation
array.put("name", node.getName());
array.put("VMI", amiID);
array.put("flavor", flavorID);
array.put("KeyPair", keypairName);
array.put("initInstances", String.valueOf(minCount));
xmlinfo.add(array);
if (!this.failed_to_deploy.isEmpty()
        && !ActiontoDo
```

```
                                .equals(ActionToDo.CONTINUE_ON_ERROR))
                            break;
                        }
                    }
                }

                // set the time that the deployment finished
                String finish_time = new SimpleDateFormat("dd/M/yyyy hh:mm:ss")
                        .format(Calendar.getInstance().getTime());

                // create JSON file
                // if there is no error create the normal JSON
                // else create the JSON containing the error
                if (this.failed_to_deploy.isEmpty())
                    this.json.write_data(xmlinfo, serviceTemplate.getName(),
                            this.depID, this.modules, this.instances,
                            start_time, finish_time);
                // create JSON with information about the error
                else
                    this.json.write_data(xmlinfo, serviceTemplate.getName(),
                            this.failed_to_deploy, this.depID, this.modules,
                            this.instances, start_time, finish_time);
            }
            getDeploymentStatus();
        } catch (Exception ex) {
            this.setException(ex);
            ex.printStackTrace();
        }
    }

    protected String flavorRefForRegion(NovaApi api, String regionId) {
        FlavorApi flavorApi = api.getFlavorApiForZone(regionId);
        return DEFAULT_FLAVOR_ORDERING.min(flavorApi.listInDetail().concat())
                .getId();
    }
```

```
    static final Ordering<Flavor> DEFAULT_FLAVOR_ORDERING = new Ordering<Flavor>() {
        public int compare(Flavor left, Flavor right) {
            return ComparisonChain.start()
                    .compare(left.getVcpus(), right.getVcpus())
                    .compare(left.getRam(), right.getRam())
                    .compare(left.getDisk(), right.getDisk()).result();
        }
    };

    public Object getResult() {
        return null;
    }

    private static boolean keyPairExists(final KeyPairApi keyPairApi,
            final String keyPairName) {
        if (keyPairApi != null && keyPairName != null) {
            FluentIterable<? extends KeyPair> list = keyPairApi.list();
            Iterator<? extends KeyPair> iterator = list.iterator();

            while (iterator.hasNext()) {
                KeyPair keypair = iterator.next();
                if (keypair.getName().equals(keyPairName)) {
                    return true;
                }
            }
        }
        return false;
    }

    /**
     * @param project
     * @param ec2
     * @param keyp
     * @throws IOException
     */
    private static String importKeyPair(final String publicKeyFile,
            final ICloudProject project) throws IOException {
        /* Read Public Key */
        String encodedPublicKey = null;
        BufferedReader br = null;
        try {
            // ICloudElement element =
            // CloudModel.getRoot().findChildWithResource(
            // publicKeyFile );
            // For now get the File
            // TODO - Incorporate Keypairs in CloudModel
            File file = new File(
                    Platform.getLocation()
                            + "/" + project.getName() + "/Artifacts/Deployment Scripts/" + publicKeyFile); //$NON-NLS-1$ //$NON-NLS-2$
            if (file.exists()) {
                br = new BufferedReader(new FileReader(file));
                encodedPublicKey = br.readLine();
                encodedPublicKey.trim();
            }
        } catch (IOException ioe) {
            throw ioe;
        } finally {
            if (br != null) {
                try {
                    br.close();
                    br = null;
                } catch (IOException ioe) {
                    throw ioe;
                }
            }
        }
        return encodedPublicKey;
    }

    private static String importScript(final String scriptFile,
            final ICloudProject project) throws IOException {

        /* Read Script */
```

```
    /* Read Script */
    String scriptContent = null;
    BufferedReader br = null;
    try {
        // ICloudElement element =
        // CloudModel.getRoot().findChildWithResource(
        // publicKeyFile );
        // For now get the File
        // TODO - Incorporate Keypairs in CloudModel
        File file = new File(
                Platform.getLocation()
                        + "/" + project.getName() + "/Artifacts/Deployment Scripts/" + scriptFile); //$NON-NLS-1$ //$NON-NLS-2$
        if (file.exists()) {
            br = new BufferedReader(new FileReader(file));
            scriptContent = br.readLine();
            scriptContent.trim();
        }
    } catch (IOException ioe) {
        throw ioe;
    } finally {
        if (br != null) {
            try {
                br.close();
                br = null;
            } catch (IOException ioe) {
                throw ioe;
            }
        }
    }
    return scriptContent;
}

/**
 * Creates a deployment object
 *
 * @param appName
 *          Application Name
 * @return deploymentID
 */
private String createDeployment(String appName) {
    Deployment dep = InfoSystemFactory.eINSTANCE.createDeployment();
    dep.setDepID(UUID.randomUUID().toString());
    dep.setDeploymentName(appName);
    this.deployments.put(dep.getDepID(), dep);

    return dep.getDepID();
}

/**
 * Creates a module for a specific deployment
 *
 * @param depID
 *            deploymentID we want to associate with the module
 * @param modName
 *            module name
 * @return moduleID
 */
private String createModule(String depID, String modName) {
    Deployment d = this.deployments.get(depID);
    Module module = null;
    String modID = null;

    if (d != null) {
        module = InfoSystemFactory.eINSTANCE.createModule();
        module.setModID(UUID.randomUUID().toString());
        module.setModuleName(modName);
        modID = module.getModID();
        addModule(d, module);
    }

    return modID;
}
```

```java
    /**
     * Adds an Instance on a specific module and creates it on the OpenStack
     * Provider
     *
     * @param depID
     *            deploymentID
     * @param modID
     *            ModuleID
     * @param params
     *            instance details
     * @param sv
     *            Server options
     * @return instanceID
     * @throws Exception
     *
     */
    private String addInstanceToModule(String depID, String modID,
            HashMap<String, String> params, CreateServerOptions sv)
            throws Exception {
        String instID = null;

        Deployment d = this.deployments.get(depID);
        if (d != null) {
            Module module = null;
            for (Module m : d.getModules()) {
                if (m.getModID().equals(modID))
                    module = m;
            }

            if (module != null) {
                ServerCreated newServer = this.serverApi.create(
                        params.get("name"), params.get("amiID"),
                        params.get("flavor"), sv);
                instID = newServer.getId();
                addInstance(module, instID);
            } else
                System.out
                        .println("module ID does not exist, please create a module in a valid deployment first");
```

```
        } else
            System.out
                    .println("deployment ID does not exist, please create a deployment first");

        return instID;
    }

    /**
     * Adds a specific Module Object to a Deployment Object
     *
     * @param dep
     * @param module
     */
    private void addModule(Deployment dep, Module module) {
        dep.getModules().add(module);
    }

    /**
     * Adds a Virtual Instance Object to a Module Object
     *
     * @param mod
     * @param instID
     */
    private void addInstance(Module mod, String instID) {
        VirtualInstance vi = InfoSystemFactory.eINSTANCE
                .createVirtualInstance();
        vi.setUID(instID);
        mod.getInstances().add(vi);
    }

    /**
     * Gets from getInputData the deployment model and check the instances for
     * their status
     */
    private void getDeploymentStatus() {
        List<Deployment[]> deploymentData = this.json.getInputData();
        ComputeService computeService = OpenStackClient.getInstance()
                .getComputeService();
```

```
        for (int i = 0; i < deploymentData.size(); i++) {
            Deployment[] dep = deploymentData.get(i);
            for (int j = 0; j < dep[i].getModules().size(); j++) {
                Module mods = dep[i].getModules().get(j);
                for (int k = 0; k < mods.getInstances().size(); k++) {
                    VirtualInstance vi = mods.getInstances().get(k);
                    try {
                        if (!computeService
                                .getNodeMetadata(this.zone + "/" + vi.getUID())
                                .getStatus().equals("RUNNING")) {
                            // do nothing instance exists
                        }
                    } catch (Exception ex) {
                        System.err.println("Instance with id:" + vi.getUID()
                                + " NOT found!");
                    }
                }
            }
        }
    }

    /**
     * Returns the ActionToDo
     *
     * @return ActiontoDo
     */
    public ActionToDo getActiontoDo() {
        return this.ActiontoDo;
    }
}
```

Τερματισμός Virtual Instance

```
public class OpenStackOpTerminateApplication extends
        AbstractOpenStackOpTerminateApplication {

    private HashMap<String, Deployment> deployments = new HashMap<String, Deployment>();
    private String depID = new String();
    private List<String> modules = new ArrayList<String>();
    private List<String> instances = new ArrayList<String>();

    private NovaApi nova = OpenStackClient.getInstance().getNova();
    // TODO - Hack for Nephelae. Only one region there. Fix it for other
    // providers with more zones.
    private String zone = "regionOne";

    public OpenStackOpTerminateApplication(HashMap<String, Deployment> dep,
            String depid, List<String> mods, List<String> inst) {
        this.deployments = dep;
        this.depID = depid;
        this.modules = mods;
        this.instances = inst;
    }

    /*
     * (non-Javadoc)
     *
     * @see
     * org.eclipse.camf.connectors.aws.operation.AbstractEC2OpDeployApplication
     * #run()
     */
    @Override
    public void run() {

        setResult(null);
        setException(null);
        try {

            for (int i = 0; i < this.modules.size(); i++) {
                if (this.terminateModule(this.depID, this.modules.get(i)))
```

```
                    System.out
                            .println("Successfully terminate module with modID: "
                                    + modules.get(i)
                                    + ", for deployment with depID: "
                                    + this.depID);
                else
                    System.out
                            .println("Failed to terminate module with modID: "
                                    + this.modules.get(i)
                                    + ", for deployment with depID: "
                                    + this.depID);
            }

            if (this.terminateDeployment(this.depID))
                System.out
                        .println("Successfully terminated deployment with depID: "
                                + this.depID);
            else
                System.out
                        .println("Failed to terminated deployment with depID: "
                                + this.depID);

        } catch (Exception ex) {
            this.setException(ex);
            ex.printStackTrace();
        }
    }

    public Object getResult() {
        return null;
    }

    /**
     * Terminates all instances inside a module given a deployment and module ID
     *
     * @param depID
     *            a DeploymentID
     * @param modID
```

```
     *            a ModuleID
     * @return true if termination was successful else false
     */
    private boolean terminateModule(String depID, String modID) {
        Deployment d = this.deployments.get(depID);
        if (d != null) {
            Module module = null;
            for (Module m : d.getModules()) {
                if (m.getModID().equals(modID)) {
                    module = m;
                    EList<VirtualInstance> ilist = module.getInstances();
                    for (int i = ilist.size(); i > 0; i--)
                        this.removeInstanceFromModule(depID, modID,
                                ilist.get(i - 1).getUID());

                    removeModule(d, modID);

                    return true;
                }
            }
        }
        return false;
    }

    /**
     * Removes an instance from a module given a deployment id, module id and
     * the instance id we want to remove
     *
     * @param depID
     *            a DeploymentID
     * @param modID
     *            a ModuleID
     * @param vID
     *            an InstanceID
     * @return true if instance removed successfully
     */
    private boolean removeInstanceFromModule(String depID, String modID,
            String vID) {
        Deployment d = this.deployments.get(depID);
        if (d != null) {
            Module module = null;
            for (Module m : d.getModules()) {
                if (m.getModID().equals(modID))
                    module = m;
            }

            if (module != null) {
                if (terminateInstance(vID)) {
                    removeInstance(module, vID);
                    System.out.println("Successfully removed instance: " + vID);
                    return true;
                } else
                    System.out.println("Instance does not exist: " + vID);
            } else
                System.out
                        .println("module ID does not exist, please create a module in a valid deployment first");
        }
        System.out
                .println("deployment ID does not exist, please create a deployment first");

        return false;
    }

    /**
     * Terminates an instance from the provider based on the parameter that was
     * given
     *
     * @param vID
     *            an InstanceID
     * @return true if the termination from the provides was successful
     */
```

```
    private boolean terminateInstance(String vID) {
        ServerApi serverAPI = this.nova.getServerApiForZone(this.zone);
        try {
            if (serverAPI.delete(vID)) {
                return true;
            }
        } catch (Exception ex) {
            try {
                throw new OpenStackOpExceptions(
                        "Failed to terminate. Please contact your system administrator.",
                        OpenStackOpExceptions.ExceptionType.TERMINATE);
            } catch (OpenStackOpExceptions CustomExcpetion) {
                this.setException(ex);
                System.err.println(CustomExcpetion.getLocalizedMessage());
                // CustomExcpetion.printStackTrace();
            }
        }
        return false;
    }

    /**
     * Terminates a deployment based on the deployment ID
     *
     * @param depID
     *            a DeploymentID
     * @return true if deployment terminated successfully
     */
    private boolean terminateDeployment(String depID) {
        boolean response = (this.deployments.remove(depID) != null) ? true
                : false;

        return response;
    }
    /**
     * Removes a module from a deployment
     *
     * @param dep
     *            a Deployment Object
     * @param modID
     *            a ModuleID
     */
    private void removeModule(Deployment dep, String modID) {
        for (int i = 0; i < dep.getModules().size(); i++) {
            Module m = dep.getModules().get(i);
            if (m.getModID().equals(modID))
                dep.getModules().remove(i);
        }
    }

    /**
     * Removes an instance from the module that was given
     *
     * @param m
     * @param instID
     */
    private void removeInstance(Module mod, String instID) {
        for (int i = 0; i < mod.getInstances().size(); i++) {
            VirtualInstance obj = mod.getInstances().get(i);
            if (obj.getUID().equals(instID))
                mod.getInstances().remove(i);
        }
    }
}
```

Διαχείριση Exception

```java
public class OpenStackOpExceptions extends Exception {
    private static final long serialVersionUID = 1L;

    public enum ExceptionType {
        FLAVORS, IMAGES, INSTANCE, KEYPAIR, NETWORK, SECURITY_GROUP, DEPLOY, TERMINATE
    };

    private String message = null;
    private ExceptionType extype;

    public OpenStackOpExceptions() {
        super();
    }

    /**
     * Creates custom exception based on the parameters given
     *
     * @param message
     *            Custom Exception message
     * @param type
     *            Custom Exception type
     */
    public OpenStackOpExceptions(String message, ExceptionType type) {
        super(message);
        this.message = type + " Exception: " + message;
        this.extype = type;
    }

    public OpenStackOpExceptions(Throwable cause) {
        super(cause);
    }

    public ExceptionType getExceptionType() {
        return this.extype;
    }

    @Override
    public String toString() {
        return message;
    }

    @Override
    public String getMessage() {
        return message;
    }
}
```