ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Η/Υ

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ / **TECHNICAL UNIVERSITY OF CRETE**

ΠΛΗ 513 - Υπηρεσίες στο Υπολογιστικό Νέφος και την Ομίχλη
Cloud Project Part 2

Φοιτητής :
Σκλάβος Παναγιώτης 2018030170

Διδάσκων :
Ευριπίδης Πετράκης

1. Εισαγωγή

Η δεύτερη φάση της εργασίας πραγματεύεται ζητήματα εικονοποίησης της εφαρμογής που δημιουργήθηκε στο πρώτο μέρος, καθώς και την εξοικείωση με το περιβάλλον του GCP. Συγκεκριμένα η λογική του web app και server που υλοποιήσαμε στο προηγούμενο μέλος, εξελίχθηκε σε containerized application και με τον τρόπο αυτό, καθώς και με την αξιοποίηση των εργαλείων που παρέχει το docker, ενοποιήθηκε με άλλα built-in microservices που υπάρχουν στο dockerhub.io. Επιπλέον μετά την ολοκλήρωση της διασύνδεσης όλων των containers η εφαρμογή μεταφέρθηκε στο GCP, όπου πραγματοποιήθηκε εύκολη εγκατάσταση του συστήματος χρησιμοποιώντας το αρχείο docker-compose.yml που περιέχει όλη τη συγκεντρωτική πληροφορία για τα services που χρησιμοποιήθηκαν και τις συνδέσεις τους.

2. Microservices που χρησιμοποιήθηκαν:

- **KEYROCK**: Για Authentication και διαχείριση των χρηστών η πληροφορία των οποίον φυλάσσεται σε mysql βάση δεδομένων.
- **PEP PROXY-WILMA:** Για προστασία των backend εφαρμογών από μη εξουσιοδοτημένους-κακόβουλους χρήστες.
- **ORION Context Broker:** Για υλοποίηση Pub/Sub λειτουργίας κατά την οποίο οι χρήστες θα ενημερώνονται για την διαθεσιμότητα προϊόντων που τους ενδιαφέρουν.
- **3 Βάσεις Δεδομένων:** Μια MongoDb για την αποθήκευση των προϊόντων και των καλαθιών, Μια MongoDb για διατήρηση της πληροφορίας του Orion για τα subscriptions και μια MysqlDb για την διατήρηση των δεδομένων του Keyrock και των χρηστών του.
- **Data Storage Service:** το συγκεκριμένο αποτελεί ένα custom service με το οποίο τα Rest αιτήματα της εφαρμογής υλοποιούνται ως queries στην mongoDb. Η Αποστολή των Αιτημάτων πραγματοποιείται είτε με την χρήση του περιβάλλοντος του postman, είτε με Curl requests. Να σημειωθεί ότι τα requests αυτά αποστέλλονται περνώντας πρώτα από τον proxy ώστε να εγγυάται η ασφάλεια του data storage από μη εξουσιοδοτημένα requests (valid tokens).

3. Ανάλυση Υλοποίησης Ανά Τμήμα:

3.1. ΑΝΑΠΤΥΞΗ (DEPLOYMENT) ΤΗΣ ΕΦΑΡΜΟΓΗΣ ΣΕ ΠΕΡΙΒΑΛΛΟΝ DOCKER

Σε πρώτη φάση έγινε εξοικείωση με τις δυνατότητες και τις λειτουργίες του docker προκειμένου να μπορέσει να υλοποιηθεί η εφαρμογή σε ένα containerized environment. Σε περισσότερη ανάλυση, δημιουργήθηκε αρχείο docker-compsoe.yml στο οποίο πραγματοποιούνται όλες απαραίτητες συνδέσεις. Στο αρχείο αυτό περιέχονται οι κατάλληλες εκδόσεις από τα images προκειμένου να εξασφαλισθεί η ορθή λειτουργία και συνεργασία όλων των υπηρεσιών, καθώς με τις latest εκδόσεις δεν θα μπορούσαμε να παρέχουμε εγγυήσεις υποστήριξης όλων των υπηρεσιών μετά από καιρό. Επιπλέον έχουν γίνει mapped τα ports τα οποίο επιθυμούμε να συνδέονται μεταξύ των containers και του localhost καθώς και exposed όσα ports θέλουμε να ακούν για traffic. Τέλος προκειμένου να διατηρούνται τα δεδομένα ακόμη και αν πέσει κάποιος container έχει γίνει volume mapping μεταξύ του docker host και των container. Συνεπώς ακόμη και αν ένας container σταματήσει την λειτουργία του η πληροφορία του παραμένει και κληροδοτείται από τον docker host. Αξίζει να σημειωθεί ότι το

mapping έχει πραγματοποιηθεί, όπου αυτό χρειάζεται, με volume bind δηλαδή σε ένα named volume εντός του docker host filesystem στο volumes section.

3.2 ΔΥΝΑΜΙΚΗ ΑΝΑΝΕΩΣΗ ΤΟΥ ΠΕΡΙΕΧΟΜΕΝΟΥ ΤΩΝ ΣΕΛΙΔΩΝ ΜΕΣΩ ΑΣΥΓΧΡΟΝΗΣ ΕΠΙΚΟΙΝΩΝΙΑΣ

Ουσιαστικά το τμήμα αυτό της εργασίας είχε υλοποιηθεί ήδη από το προηγούμενο μέλος της εργασίας καθώς το ίδιο αποτελούσε ζητούμενο κι εκεί. Ωστόσο αξίζει να αναφερθεί ο τρόπος οργάνωσης της υλοποίησης στο project προκειμένου να διευκολυνθεί η διορθωτική διαδικασία. Αναλυτικότερα, τα ajax requests αποστέλλονται από το javascript αρχείο της εκάστοτε σελίδας(πχ. confirm_user.php εντός του admin.js). Το php αρχείο που εκτελεί την ζητούμενη backend διεργασία βρίσκεται είτε στον φάκελο estore/assets/scripts/ajaxfiles/x.php, αν δεν χρειάζεται να εκτελεσθεί από την υπηρεσία data_storage με REST API, είτε στον φάκελο estore/assets/scripts/services/data_storage_proxy, αν αυτό χρειάζεται. Αντίστοιχα αν αναφερόμαστε στον orion τότε το αρχείο αυτό βρίσκεται στο ./services/orion_proxy. Στις αμφότερες 2 περιπτώσεις η backend λειτουργία πρακτικά εκτελείται μέσω του data_storage service, ενώ τα αρχεία που βρίσκονται στους προαναφερθέντες φακέλους απλώς παράγουν και αποστέλλουν με Curl το κατάλληλο αίτημα σε αυτό.

Πρακτικά, η πλειοψηφία των εργασιών ωστόσο που απαιτούν ajax στην εφαρμογή μας, απαιτούν πρόσβαση στις βάσεις δεδομένων, συνεπώς οι περισσότερες κλήσεις, αν όχι όλες βρίσκονται στον κατάλληλο xxx_proxy φάκελο. Ο φάκελος ajaxfiles κυρίως διατηρείται ως σημείο αναφοράς ως προς την προηγούμενη υλοποίηση που όλες οι διεργασίες πραγματοποιούνταν άμεσα από τον server της εφαρμογής.

3.3 ΑΠΟΘΗΚΕΥΣΗ ΤΩΝ ΔΕΔΟΜΕΝΩΝ ΤΗΣ ΕΦΑΡΜΟΓΗΣ ΜΕΣΩ ΤΗΣ ΥΠΗΡΕΣΙΑΣ DATA STORAGE

Για την αποθήκευση των δεδομένων της εφαρμογής έχει χρησιμοποιηθεί μια noSql MongoDb. Στην βάση δεδομένων αυτή διατηρείται η πληροφορία για τα products τα οποίο πωλούνται μέσω της εφαρμογής, καθώς και τα carts που δημιουργούν οι χρήστες. Η πληροφορία για τους χρήστες απομονώνεται στο Keyrock και την mysql βάση που διαχειρίζεται τα δεδομένα του.

Για την διαχείριση των δεδομένων της mongoDB βάσης από την εφαρμογή, αξιοποιείται η υπηρεσία data storage, όπως αναφέραμε σύντομα προηγουμένως. Στην φάση αυτή αξίζει να επεξηγηθεί περαιτέρω η λειτουργικότητα της υπηρεσίας ώστε να μη μην αποτελεί black box. Ουσιαστικά μια ολοκληρωμένη δρομολόγηση ενός Request είναι η ακόλουθη που παρουσιάζεται μέσω του παραδείγματος του add to cart. Αρχικά ο χρήστης πατά πάνω στο κουμπί για προσθήκη στο καλάθι για ένα προϊόν. Αυτό πυροδοτεί τον click event handler ώστε να ανανεωθούν τα δεδομένα στην βάση και να πραγματοποιηθεί η ζητούμενη λειτουργία. Ο handler αυτός λοιπόν, μέσω της jQuery(javascript) αποστέλλει ένα Post request το οποίο αναλαμβάνει το data_storage_proxy/add_to_cart.php αρχείο. Εκεί δομείται ένα REST(http) request το οποίο αποστέλλεται με curl στην υπηρεσία data_storage. Η υλοποίηση που ακολουθείται στο data_storage αποτελεί οντοκεντρική χρήση της php.

Συγκεκριμένα δημιουργούνται κλάση η κλάση config/Database.php για την MongoDb, η User για τους χρήστες, η Product για τα προιόντα και η Cart για τα καλάθια. Εντός των κλάσεων αυτών υπάρχουν υλοιποιήσεις ποικίλων συναρτήσεων που αφορούν τα αντίστοιχα αντικείμενα τους. Συνεπώς όταν λαμβάνεται το αίτημα που αποστείλλαμε με curl από το app λαμβάνεται από το data_storage/api/cart/add.php, αρχικοποιούνται τα κατάλληλα στιγμιότυπα των άνωθεν κλάσεων και μέσω εκείνων εκτελούνται οι συναρτήσεις για πραγματοποίηση της λειτουργικότητας στην βάση. Συγκεκριμένα για το παράδειγμά μας, για προσθήκη σε καλάθι καλείται η cart_obj->addCart() που πραγματοποιεί ένα insertOne() query στην mongodb.

Μια ιδιαιτερότητα της υλοποίησης που παραδίδεται, είναι το γεγονός ότι και οι λειτουργίες που αφορούν τους χρήστες (οι οποίοι διατηρούνται από το keyrock) έχουν ενταχθεί στο data_storage καθώς θεωρήθηκε καλή ιδέα να εφαρμόζουμε το ίδιο REST API για όλες τις λειτουργίες εντός της εφαρμογής. Συνεπώς όταν θέλουμε να κάνουμε access στο mysql db των χρηστών αυτό πραγματοποιείται αποστέλλοντας ένα RESTful Request στο data storage το οποίο με την σειρά του αξιοποιεί το REST API του Keyrock ώστε να λάβει τα ζητούμενα δεδομένα από την βάση.

3.4 ΕΠΕΚΤΑΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ ΜΕ ΧΡΗΣΗ PUB/SUB ΥΠΗΡΕΣΙΑΣ (ORION CB)

Η Υλοποίηση του Orion Context Broker αποτελεί το μόνο τμήμα της άσκησης που είναι ελλιπώς υλοποιημένο. Αναλυτικότερα, επιλέχθηκε να υλοποιηθεί το β) ζητούμενο που παρουσιάστηκε στην εκφώνηση, δηλαδή η ενημέρωση για τον χρήστη όταν ένα προϊόν που έχει κάνει sub παύει να είναι διαθέσιμο είτε γιατί γίνεται withdrawn είτε γιατί τελειώνει στο απόθεμα.

Όσον αφορά το πρώτο σκέλος του ερωτήματος το date of withdrawal στο entity εισάγεται ως dateExpires συνεπώς δημιουργούμε transient Entities όπως περιγράφονται στο documentation. Η επιλογή αυτή έγινε καθώς θέλουμε μετά από τον χρόνο που δίνεται το προϊόν να αφαιρείται και να αξιοποιείται το event ώστε να ενημερώνονται οι χρήστες.

Έπειτα, για το δεύτερο σκέλος εισάχθηκε το πεδίο available στα προϊόντα το οποίο φαίνεται μόνο στο seller panel, το οποίο κρατάει ένα count των προϊόντων που απομένουν. Συνεπώς κάθε φορά που ένα προϊόν προστίθεται στο καλάθι το available μειώνεται και όταν αφαιρείται αυξάνεται. Συνεπώς για το subscription έχει δοθεί το expression: available ==0 ώστε να ενημερώνονται οι subscribers όταν δεν υπάρχουν άλλα διαθέσιμα.

Κατά τ’ άλλα έχει υλοποιηθεί το orion_cretate_entity, orion_update_entity, orion_add_subscription με επιτυχία, ωστόσο δεν πρόλαβε ν υλοποιηθεί η λογική στο data storage ώστε να δημιουργούνται τα alerts στουw χρήστες και να διαχειρίζονται τα subscriptions. Επιπλέον στα προϊόντα έχει προστεθεί κουμπί για εισαγωγή subscription αλλά δεν έχει υλοποιηθεί κάποια αλλαγή μέσω javascript ώστε να μπορεί ο χρήστης να βλέπει τα προϊόντα στα οποία είναι subscribed.

3.5 ΧΡΗΣΗ ΤΗΣ ΥΠΗΡΕΣΙΑΣ KEYROCK IDM (FIWARE) ΓΙΑ ΑΥΘΕΝΤΙΚΟΠΟΙΗΣΗ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗ ΤΩΝ ΧΡΗΣΤΩΝ

Όσον αφοράν την αυθεντικοποίηση των χρηστών, όλες οι διεργασίες εκτελούνται μέσω του idm του Keyrock αξιοποιώντας το πρωτόκολλο OAuth 2.0. Προκειμένου λοιπόν να γίνει η διαχείριση της εφαρμογής μας μέσω του Keyrock έχουν γίνει όλα τα αναγκαία βήματα που αναφέρονται στην εκφώνηση, συνεπώς δεν θα σχολιαστεί κάτι πάνω σε αυτά. Για την εφαρμογή μας επιπλέον, έχουν παραχθεί οι ρόλοι User, Admin, Seller στους οποίους έχουν αποδοθεί κάποια permissions που δημιουργήθηκαν στην γραφική διεπαφή του Keyrock. Επομένως, μετά την σύνδεση με το Keyrock όλες οι λειτουργίες σχετικά με authentication και διαχείριση των χρηστών πραγματοποιούνται μέσω REST API που παρέχει η εφαρμογή.

Στο σημείο αυτό θα σχολιαστούν σε μεγαλύτερο βάθος ορισμένες implementation specific επιλογές που πραγματοποιήθηκαν όσον αφορά τις παραπάνω λειτουργικότητες. Αρχικά παρατηρήθηκε ότι οι χρήστες που εισάγονται εντός του Keyrock μετά το registration, δημιουργούνται ως enabled στο app. Η Default επιλογή αυτή δεν ήταν ιδιαίτερα βολική καθώς το επιθυμητό αποτέλεσμα θα ήταν κάποιος διαχειριστής να αναλαμβάνει από το admin panel του Keyrock την ενεργοποίηση του χρήστη. 3 πιθανά fixes δοκιμάστηκαν, και απέτυχαν όλα. Πρώτα επιχειρήθηκε κατά την δημιουργία του χρήστη να περνιέται στο json αρχείο με τα registration info (όπως το email,username και password), το πεδίο enabled: false. Αυτό έγινε καθώς παρατηρήθηκε ότι έτσι βρίσκεται αποθηκευμένο το πεδίο αυτό στην βάση idm. Έπειτα επιχειρήθηκε μετά την εισαγωγή του χρήστη στο Keyrock να γίνεται update με Patch του πεδίου αυτού σε false χωρίς κάποιο αποτέλεσμα ωστόσο. Τέλος με την εντολή grep -r enabled: true στα linux επιχειρήθηκε να εντοπιστεί το confiig file το οποίο διατηρεί την default αυτή τιμή, κα πάλι ανεπιτυχώς. Παρόλο που βρέθηκαν κάποια instances της παραπάνω αναζήτησης, οι αλλαγές των πεδίων σε false δεν αποφάνθηκαν αποτελεσματικές. Καταληκτικά, λοιπόν, επιλέχθηκε το πεδίο αυτό να είναι by default true, καθώς δεν επηρεάζει ιδιαίτερα την υλοποίηση, παρά μόνο την εκφώνηση.

Μια εφικτή λύση θα ήταν hardcoded να εκτελούμε ένα query μέσα στην βάση idm μέσω της Sql, ωστόσο η λύση αυτή δεν προτιμήθηκε καθώς θα έθετε σε κίνδυνο την βάση μας από κακόβουλες επιθέσεις, παρακάμπτοντας την ασφαλής υλοποίηση που είχαμε ως τώρα. Συνεπώς ο κίνδυνος θεωρήθηκε μεγαλύτερος από το κέρδος.

Επόμενη επιλογή που χρήζει ανάλυση είναι ο τρόπος με τον οποίο πραγματοποιείται το authentication του χρήστη εντός των πλαισίων της εφαρμογής. Ξεκινώντας, λοιπόν ο χρήστης, κάνοντας signup εισάγεται εντός των orgnanizations: estoreUsers ή/και estoreSellers. Τα organizations αυτά περιέχουν ως admins τους admins της εφαρμογής και ως members τους Users και τους Seller αντίστοιχα, και λειτουργούν ως queues αναμονής για τους χρήστες προς επιβεβαίωση. Όταν λοιπόν ένα χρήστης γίνεται reject, απλώς αφαιρείται από το organization στο οποίο βρίσκεται. Ενώ όταν γίνεται confirm, αφαιρείται από το κατάλληλο org(ή orgs αν είναι admin) και εισάγεται εντός των authorized users της εφαρμογής. Τα organizations αυτά είναι επίσης authorized στην εφαρμογή ωστόσο κατά το login δεν έχει υλοποιηθεί

λειτουργικότητα για εισαγωγή των χρηστών τους στην εφαρμογή συνεπώς κάτι τέτοιο δεν είναι εφικτό.

Όσον αφορά το administration Panel, διατηρείται το panel που είχαμε δημιουργήσει κατά το πρώτο μέλος της εργασίας καθώς θα ήταν άδικος κόπος εφόσον υλοποιήθηκε να χαθεί. Επιπρόσθετα, θεωρήθηκε καλύτερο η διαχείριση των χρηστών να μην απαιτεί την απομάκρυνση του χρήστη από την εφαρμογή μας. Ένα στοιχείο ωστόσο που αξίζει να σημειωθεί είναι το γεγονός ότι η βάση idm περιείχε δικά της στοιχεία για κάθε χρήστη και επιλέχθηκε να μην εισάγεται σε αυτήν επιπλέον πληροφορία όπως το όνομα το επίθετο κλπα, τα οποία διατηρούσαμε στην προηγούμενη υλοποίηση καθώς δεν επηρεάζεται με κάποιο τρόπο η λειτουργικότητα. Για τον λόγο αυτό δόθηκε στον χρήστη η επιλογή να μπορεί να επεξεργάζεται τα ήδη αποθηκευμένα cols του description και website αντ' αυτού.

Τέλος, αναφέρεται ως σημείωση ότι το auth2token που λαμβάνει ο χρήστης είναι valid για περιορισμένο χρονικό διάστημα. Συνεπώς μέσω της συνάρτησης check_login() ελέγχεται κάθε φορά που φορτώνεται μια σελίδα αν το token είναι valid ακόμη. Αν δεν είναι, τότε ο χρήστης γίνεται redirected σε κατάλληλη σελίδα σφάλματος.

3.6 ΧΡΗΣΗ ΤΗΣ ΥΠΗΡΕΣΙΑΣ PEP PROXY - WILMA (FIWARE) ΓΙΑ ΠΡΟΣΤΑΣΙΑ ΤΩΝ BACKEND ΕΦΑΡΜΟΓΩΝ ΑΠΟ ΜΗ ΕΞΟΥΣΙΟΔΟΤΗΜΕΝΟΥΣ ΧΡΗΣΤΕΣ

Προκειμένου να προστατεύεται το backend από κακόβουλους χρήστες έχει χρησιμοποιηθεί η υπηρεσία Wilma Pep Proxy σε κάθε προσπάθεια επικοινωνίας με την βάση. Είτε αυτή είναι για την mongodb με τα data της εφαρμογής, είτε για την mongo με τα δεδομένα για τον orion, είτε για αποστολή αιτημάτων ζητώντας πληροφορία από το Keyrock. Το γεγονός αυτό, πάνω στον κώδικα είναι μια μηδαμινή προσθήκη καθώς αντί να αποστέλλονται τα requests άμεσα προς το REST api του data storage, περνάνε πρώτα από το {data_storage_proxy}:{proxy_port}- το οποίο φαίνεται στο url target των curl requests.

3.7 MIGRATION ΤΗΣ ΕΦΑΡΜΟΓΗΣ ΣΕ INSTANCE ΤΟΥ GOOGLE CLOUD PLATFORM

Το Migration της εφαρμογής στο GCP δεν χρήζει κάποια εκτενή περιγραφή καθώς αποτελεί μια τυποποιημένη διαδικασία. Περιληπτικά, δημιουργήθηκε ένα VM μέσω του GCP λογαριασμού που μας δόθηκε στο μάθημα. Εντός του VM εγκαταστήσαμε docker και docker-compose και έγιναν upload με .zip αρχείο όλα τα αρχεία που αφορούν την υλοποίηση της άσκησης σε συνδυασμό με τα backups των βάσεων δεδομένων που χρησιμοποιούνταν τοπικά για να μην επαναλαμβάνεται η διαδικασία δημιουργίας τους. Το παραπάνω έγινε κάνοντας dump τα backups τοπικά μέσω του terminal και έπειτα εισαγωγή τους με αντίστοιχο τρόπο στους GCP containers από το SSH. Τέλος έγιναν expose, οι θύρες 80 kai 3005 για την εφαρμογή και τον keyrock manager αντίστοιχα, αφότου έγιναν setup τα κατάλληλα firewall rules.

Σημείωση για εκτέλεση 1: το index.php δεν είναι άμεσα accessible στην θύρα 80, αλλά θα πρέπει να τοποθετείται στο url το εξής: http://host/estore/assets. Εκεί γίνεται άμεσα ορατό το index αρχείο και η πρόσβαση παύει να είναι forbidden.

Σημείωση για εκτέλεση 2: Για να ελεγχθεί η εφαρμογή με όλα τα δεδομένα μπορείτε να εισάγεται τα backup files ως ακολούθως:

- **Mysql:** *docker exec -i mysql-db mysql -u root -psecret idm < backup.sql*
- **Mongodb(app-data):**
  1. *docker cp /dump mongo-db:/dump # to copy from outside docker inside the container*
  2. docker exec -i mongo-db /usr/bin/mongorestore --username root --password pass --authenticationDatabase admin --db estore /dump/estore

Όλοι οι χρήστες έχουν ως κωδικό «1234».