

ΧΑΡΑ ΜΠΟΥΛΟΥΓΑΡΗ

Υπηρεσίες Υπολογιστικού Νέφους

Εργασία εαρινού εξαμήνου 2021

Τεχνολογίες: NodeRed Docker OpenWhisk MinIO NodeJS Nginx

25/6/2021

ΠΕΡΙΕΧΟΜΕΝΑ

1. Περιγραφή Σεναρίου	σελ. 3
2. Τεχνολογίες και Εργαλεία που χρησιμοποιήθηκαν	σελ. 4
3. Setting Up The Environment on Windows 10	σελ. 5
4. Αρχιτεκτονική	σελ. 7
5. Patterns που χρησιμοποιήθηκαν.....	σελ.8
6. Περιγραφή υλοποίησης.....	σελ. 10
7. Πλεονεκτήματα / Μειονεκτήματα	σελ. 21
8. Χρήσιμες docker commands που χρησιμοποιήθηκαν	σελ. 22
9. Credits	σελ. 23

1. Περιγραφή Σεναρίου

Την εποχή αυτή ολοένα και περισσότεροι άνθρωποι παγκοσμίως ασχολούμαστε με την ψηφιακή τεχνολογία! Αυτό έχει ως συνέπεια να θέλουμε να την χρησιμοποιήσουμε για την διευκόλυνσή μας σε σοβαρά προβλήματα που παρουσιάζονται στην καθημερινότητά μας ακόμα και σε ποιο μικρά και ασήμαντα απλά και μόνο για αισθητικούς λόγους. Πιο συγκεκριμένα χρησιμοποιούμε φίλτρα σχεδόν παντού πλέον σε όλα τα μέσα κοινωνικής δικτύωσης και σε όλα σχεδόν τα προφίλ μας ! Όπως και στις φωτογραφίες έτσι λοιπόν και στα κείμενα χρειαζόμαστε φίλτρα ακόμα και για τέτοιους απλούς λόγους αλλά και πιο σημαντικούς που θα ορίσουμε στην επόμενη παράγραφο !

Το σενάριο βασίζεται στην λογική της εφαρμογής φίλτρων σε αρχεία κειμένου. Πιο συγκεκριμένα ο χρήστης μπορεί να αποθηκεύει σε έναν χώρο στο cloud το αρχείο που επιθυμεί και έπειτα από εκεί να καλεί το φίλτρο μέσω μιας συνάρτησης ώστε να εφαρμόζεται στο αρχείο κειμένου του και να του το μετατρέπει σε μορφή ολίδια σαν να το είχε γράψει χειρόγραφα. Ο χρήστης με αυτόν τον τρόπο διευκολύνεται στο γεγονός ότι μπορεί να έχει οποιοδήποτε κείμενο αυτός επιθυμεί ανά πάσα ώρα και στιγμή με τέλεια ορθογραφία και σε χειρόγραφη μορφή ενώ το έχει γράψει από το πληκτρολόγιο του υπολογιστή και κατ' επέκταση έχει επωφεληθεί από όλα τα βοηθητικά ορθογραφικά φίλτρα που του παρέχει ο υπολογιστής. Στην συνέχεια μπορεί να καταθέσει το κείμενό του όπου αυτός επιθυμεί χωρίς να ανησυχεί για την ορθογραφία του.

Ο χρήστης δημιουργεί από μία διεπαφή τους απαραίτητους φακέλους που χρειάζεται στο cloud storage (minIO) ενεργοποιώντας ένα flow της διεπαφής και στην πορεία ανεβάζονται εκεί τα αρχεία κειμένου του μορφής .txt ... Στη συνέχεια ένα άλλο flow είναι υπεύθυνο για τη δημιουργία του φακέλου όπου θα αποθηκεύονται τα αρχεία κειμένου στα οποία έχει εφαρμοστεί το φίλτρο και τέλος ένα τελευταίο flow αναλαμβάνει να δημιουργήσει ένα presigned URL ώστε να μπορεί να το στείλει ως παράμετρο στο function το οποίο θα εφαρμόσει το φίλτρο., αφού κατεβάσει πρώτα το αρχείο τοπικά και στο τέλος θα το ανεβάσει στο cloud storage ενημερώνοντας τον χρήστη για την επιτυχή αποθήκευση του αρχείου ή όχι.

Το σενάριο υλοποιείται με τη λογική των serverless apps που υπάρχει και εφαρμόζεται στο cloud οπότε γλυτώνουν τον χρήστη από την εγκατάσταση σε κάποια του συσκευή το λογισμικό του app αυτού έτσι καθώς παρέχεται σαν service διευκολύνει τον χρήστη και στο ενδεχόμενο περιορισμένου αποθηκευτικού χώρου που ίσως διαθέτει ο ίδιος !

2. Τεχνολογίες και Εργαλεία που Χρησιμοποιήθηκαν



3. Setting Up The Environment on Windows 10 (it needs 64 bit system and a very nice RAM)

- Εγκατάσταση του docker toolbox
- Δημιουργία dockerhub account
Σε περίπτωση που κάτι δεν τρέχει ή πετάξει error η version που σίγουρα δουλεύει είναι: Docker version 18.03.0-ce, build 0520e24302 αν πάλι κάτι πάει στραβά τότε εγκαθιστούμε πιο παλιά version του virtual box !
- Εγκατάσταση Vs Code
- Εγκατάσταση του virtual box
Εδώ θα χρειαστεί να πάω στις ρυθμίσεις του vm που έχει φτιάξει το docker toolbox και να ανοίξω τα κατάλληλα ports ώστε να μπορώ να χτυπίσω από windows μέσω port mapping το openwhisk και τον nginx στην πορεία, οπότε πάω ρυθμίσεις -> δίκτυο -> προχωρημένο -> προώθηση θύρας και ορίζω τους κανόνες μου δίνω ένα όνομα στον κανόνα και βάζω για ip οικοδεσπότη το 127.0.0.1 και για θύρα οικοδεσπότη και επισκέπτη τον ίδιο αριθμό αναλόγως τη port ανοίγω για openwhisk το 9000 και για nginx το 80 !
- Εγκατάσταση NodeJS
- Εγκατάσταση του nodered ακούει στο localhost:1880
Μεταβαίνω στον φάκελο εγκατάστασής του και το τρέχω με την εντολή node-red
- Εγκατάσταση openwhisk ακούει στο localhost:3233
Κατεβάζουμε το wsk cli από εδώ
<https://github.com/apache/openwhisk-cli/releases/tag/1.0.0>
και το βάζουμε στον φάκελο που θα έχουμε την εγκατάσταση του όλου openwhisk, μετά μπορώ ανοίγοντας και από το vs code έναν terminal να γράψω την ακόλουθη εντολή

```
docker run --name openwhisk -v //var/run/docker.sock:/var/run/docker.sock  
-p 3233:3233 -p 3232:3232 openwhisk/standalone:66a9417
```

θα πρέπει να έχω εκκινήσει το docker toolbox πριν εκτελέσω την παραπάνω εντολή! Το docker toolbox δημιουργεί ένα vm εντός του virtual box! Η παραπάνω εντολή θα εμφανίσει μία εντολή εντός του terminal με μωβ χρώμα την οποία θα κάνω copy ώστε να εκτελέσω στην πορεία ώστε να πάρω εξουσιοδότηση και να γίνει η σύνδεση μου με το wsk cli που τρέχει στο vm ώστε να μπορώ να εκτελώ το cli με την εντολή wsk. Έτσι λοιπόν μόλις η στοίβα της εγκατάστασης του openwhisk ολοκληρωθεί εκτελώ την εντολή με το μωβ χρώμα.. κάνω cd στον φάκελο όπου έχω το openwhisk και ξεκινώ την εντολή με .\ αν την εκτελώ από τον terminal του vscode και δεν ξεχνώ να σβήσω τα " από το --apihost και το --auth πάνω στην εντολή πριν την

εκτελέσω! Αν τρέχω για 2^η φορά το openwhisk δεν εκτελώ την παραπάνω εντολή παρά εκτελώ την ακόλουθη αφού υπάρχει το image

```
docker run -v //var/run/docker.sock:/var/run/docker.sock -p 3233:3233 -p 3232:3232 f32250d71024 (όπου f32250d71024 είναι το id του image το οποίο μπορώ να βρω τρέχοντας την εντολή docker images)
```

με την εντολή `.\wsk namespace list` μπορώ να δω τα namespaces που υπάρχουν ή έχω ορίσει στο openwhisk

τέλος θα χρειαστούμε και το wskdeploy cli το οποίο βρίσκουμε εδώ:

<https://github.com/apache/openwhisk-wskdeploy/releases>

περισσότερες πληροφορίες εδώ:

<https://github.com/apache/openwhisk-wskdeploy#downloading-released-binaries>

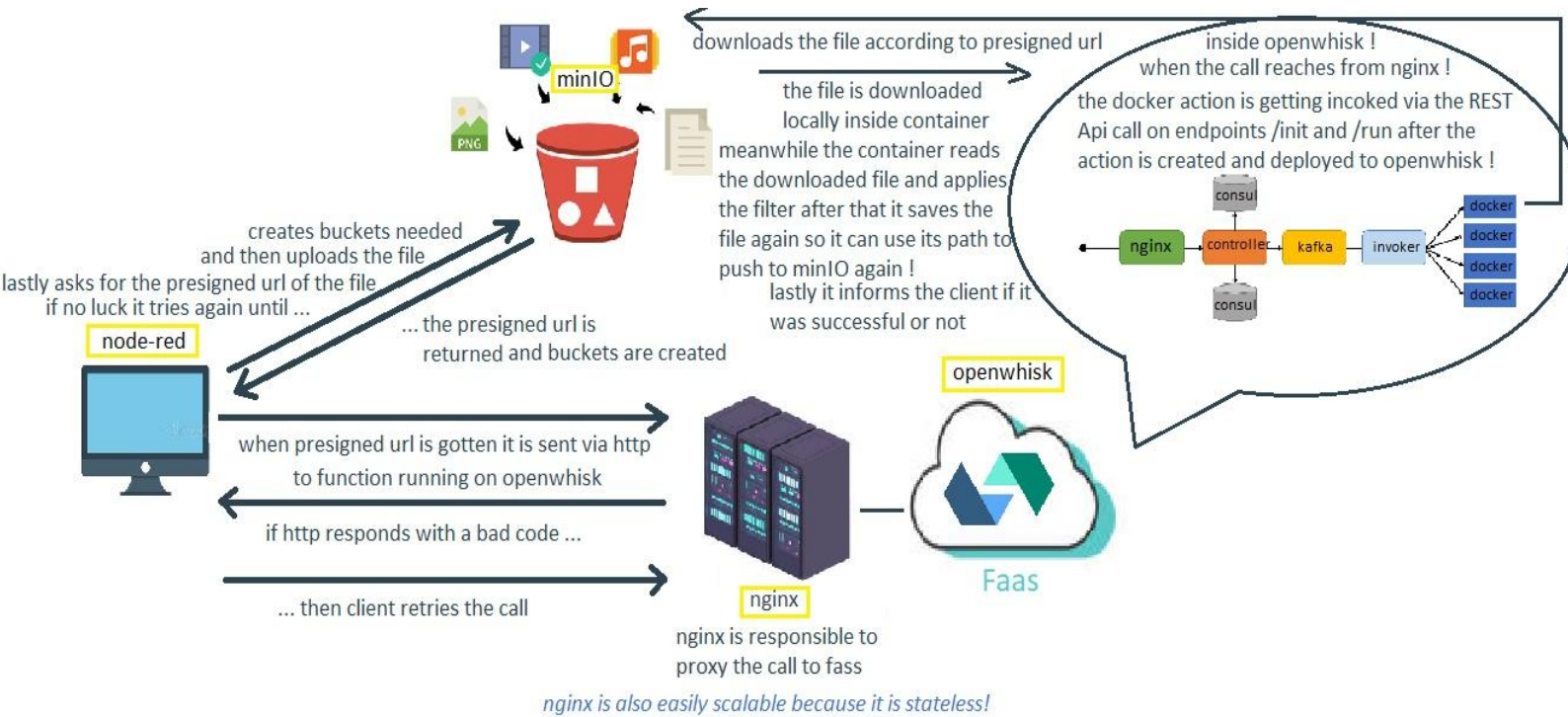
αφού το κατεβάσουμε το βάζουμε στον φάκελο εγκατάστασης του openwhisk μας !

- Εγκατάσταση minIO ακούει στο localhost:9000
Το κατεβάζω από τον σύνδεσμο:

<https://dl.min.io/server/minio/release/windows-amd64/minio.exe>

μετά σε ξεχωριστό terminal μεταβαίνω με `cd` στον φάκελο της εγκατάστασης του minIO όπου υπάρχει και το cli του ώστε να μπορέσω να τρέξω την εντολή `.\minio.exe server +the name of the folder where minIO saves objects` this name will create a folder locally on my machine

4. Αρχιτεκτονική



5. Patterns που χρησιμοποιήθηκαν

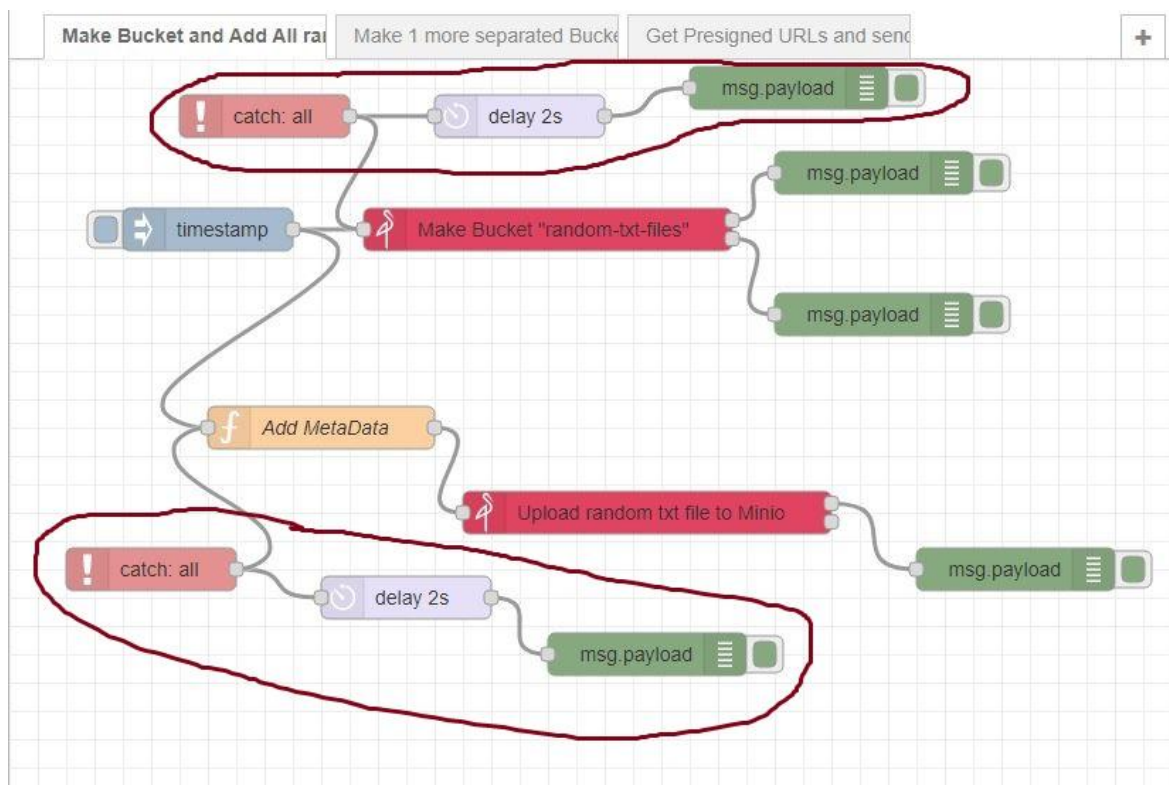
Retry Pattern

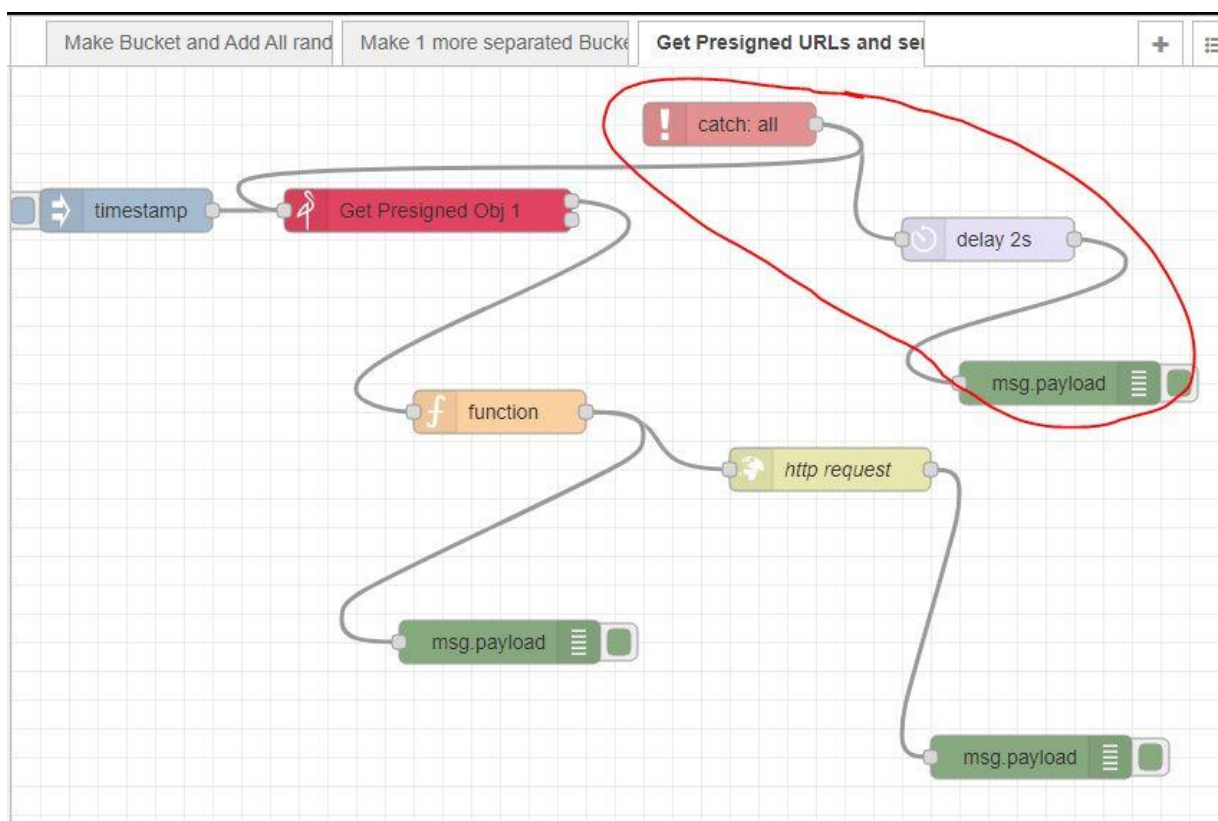
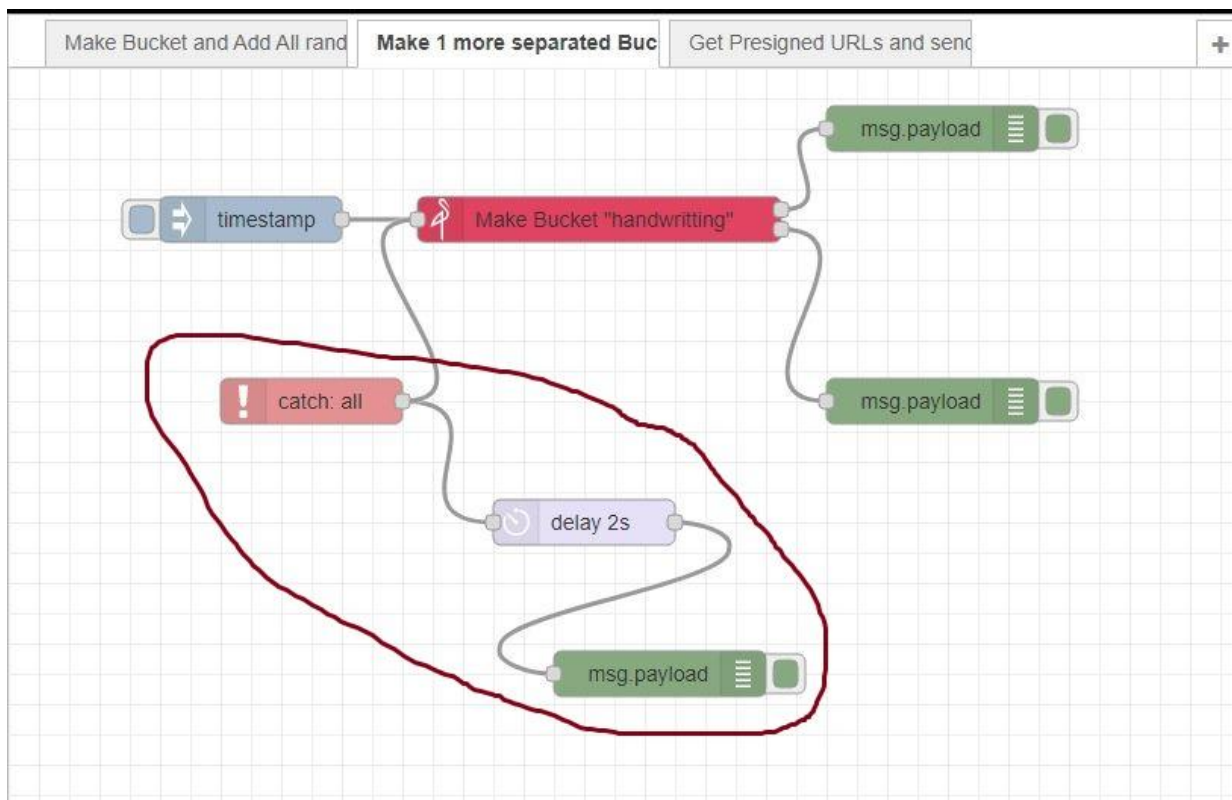
Ποιο πρόβλημα λύνει ?

- Όταν έχουμε να κάνουμε με εξωτερικά microservices τα οποία καλούμε μέσω διαδικτύου και κατ'επέκταση αν έχουμε μια διακοπή διαδικτύου πολύ πιθανό να χάσουμε το αποτέλεσμα του call που κάναμε. Αν ξαναδοκιμάσουμε να πραγματοποιήσουμε το ίδιο call μπορεί να παρατηρήσουμε πως ίσως τη δεύτερη φορά όλα να πάνε καλά !

Πότε το χρησιμοποιούμε ?

- Όταν έχουμε αστοχίες που αργούν να διορθωθούν μόνες τους





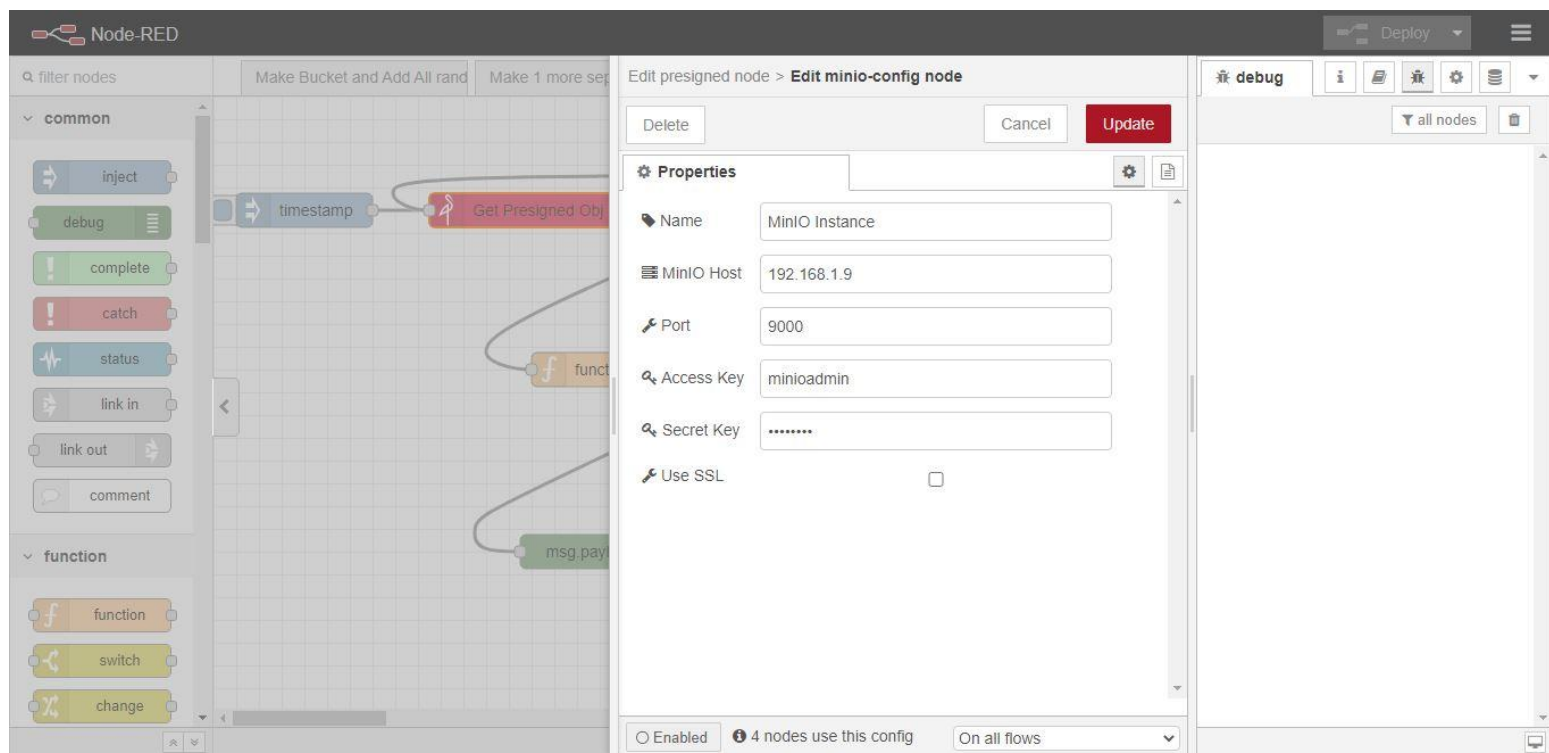
Οι τρεις παραπάνω εικόνες εξηγούν πού και πώς εφαρμόστηκε το retry pattern που σύμφωνα με το παρακάτω σύνδεσμο υλοποιείται με τον catch node ακολουθούμενο από ένα delay

<https://cookbook.nodered.org/basic/retry-on-error>

Οι κόμβοι κάνουν αυτό ακριβώς που αναφέρεται στο όνομά τους περισσότερες πληροφορίες για το περιεχόμενο του καθενός θα αναφερθούν σε επόμενη παράγραφο με συνοδεία εικόνων και επεξηγήσεις όπου είναι απαραίτητο !

6. Περιγραφή υλοποίησης

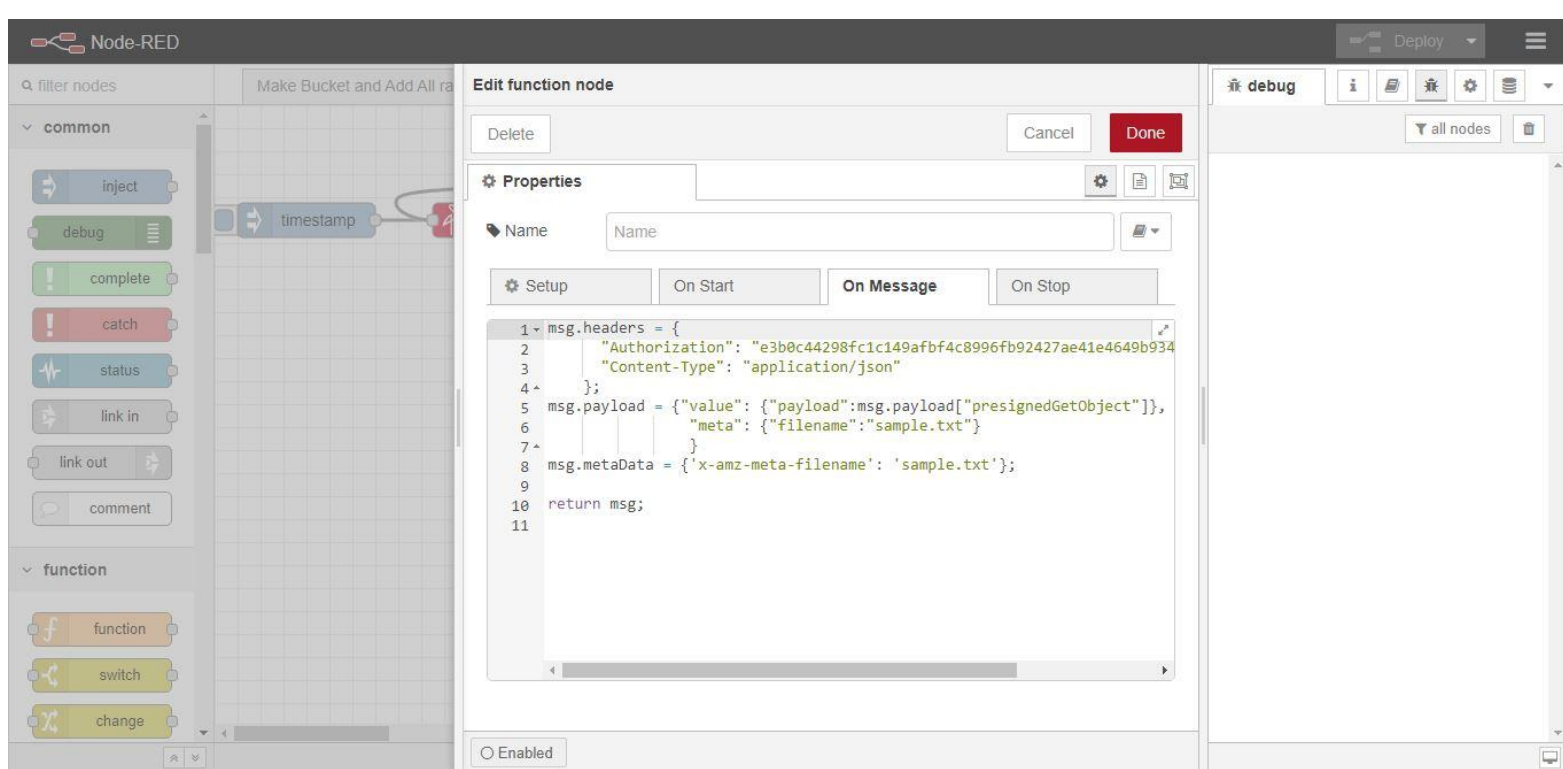
Αρχικά δημιουργήθηκαν τα node-red flows όπως φάνηκαν προηγουμένως και στην πορεία δημιουργήθηκε το function αρχείο . Παρακάτω στην εικόνα απεικονίζονται οι λεπτομέρειες των nodes που αξίζει να σημειθούν και που έχουν κάποια υλοποίηση μέσα τους από κώδικα καθώς οι κόμβοι του minIO είναι αυτοί που διατίθενται από το αντίστοιχο plugin και δέχονται στο σώμα τους απλώς ρυθμίσεις και όχι κάποιο συγκεκριμένο κομμάτι κώδικα ! Στο 3^ο flow (3^η εικόνα από τις προηγούμενες) ανοίγοντας κανείς τον κόμβο του minio παρατηρεί το παρακάτω



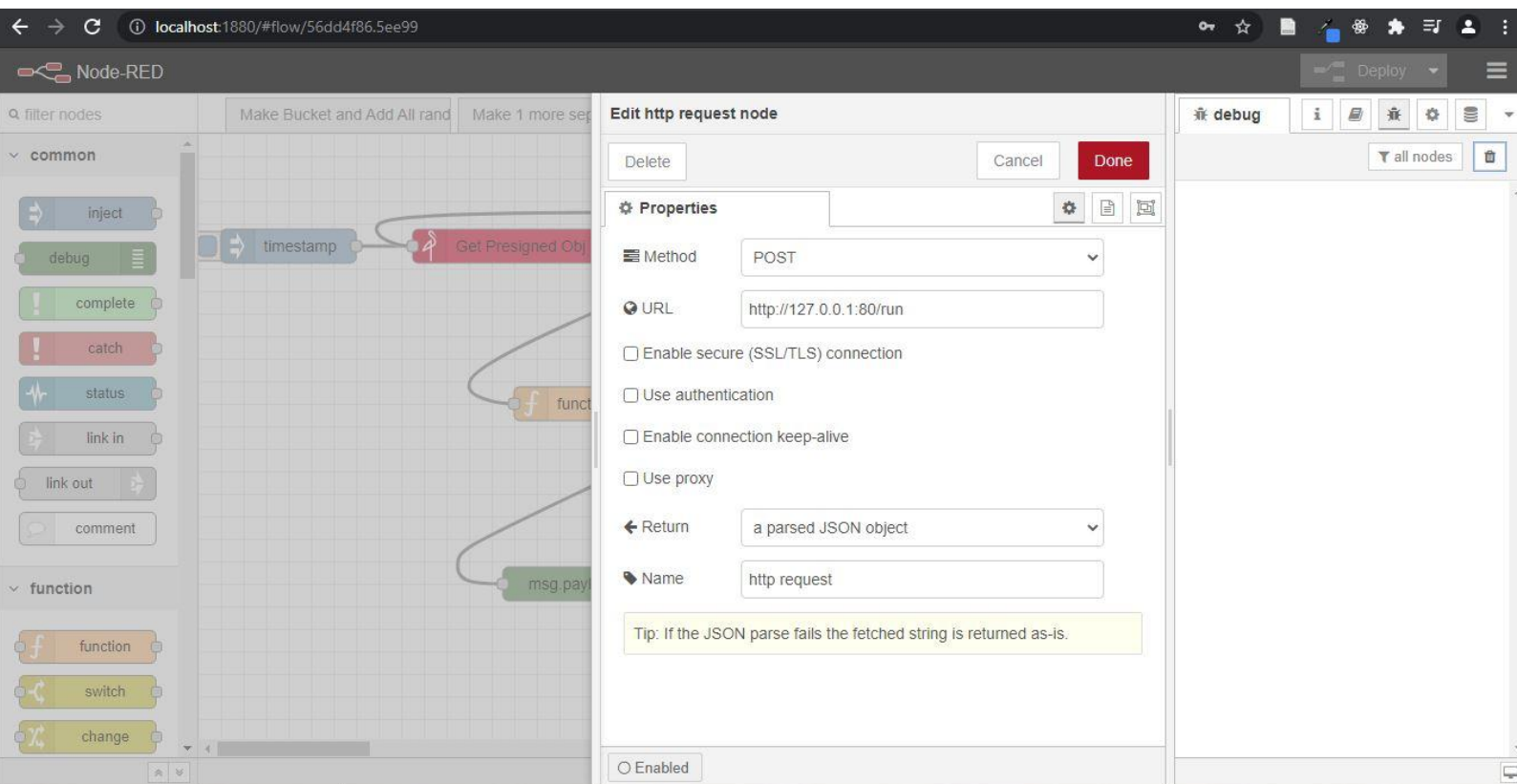
Το minio καθώς ξεκινά όταν με την κατάλληλη εντολή στον terminal το ξεκινάμε ακούει σε πολλές διευθύνσεις εκτός του localhost:9000 έτσι χρησιμοποίησα μία άλλη διαθέσιμη διεύθυνση διότι θα πρέπει να έχω πρόσβαση σε αυτό λόγω του ότι είμαι σε windows μέσα από container κάποια στιγμή ώστε να καταχωρισθεί το filtered αρχείο πάλι πίσω σε αυτό και σαν να παρατήρησα ότι βάζοντας 127.0.0.1 στη θέση του MinIO Host field δημιουργούσε πρόβλημα διότι και ο nginx και εσωτερικά το vm όταν από τον container χτυπούσα προς τα έξω τη διεύθυνση localhost μου εκλάμβανε υπέθεσα του vm το περιβάλλον ο container διότι μέσα σε

αυτό τρέχει κι έτσι δεν εκλάμβανε ως localhost:9000 το έξω από το vm περιβάλλον των windows.

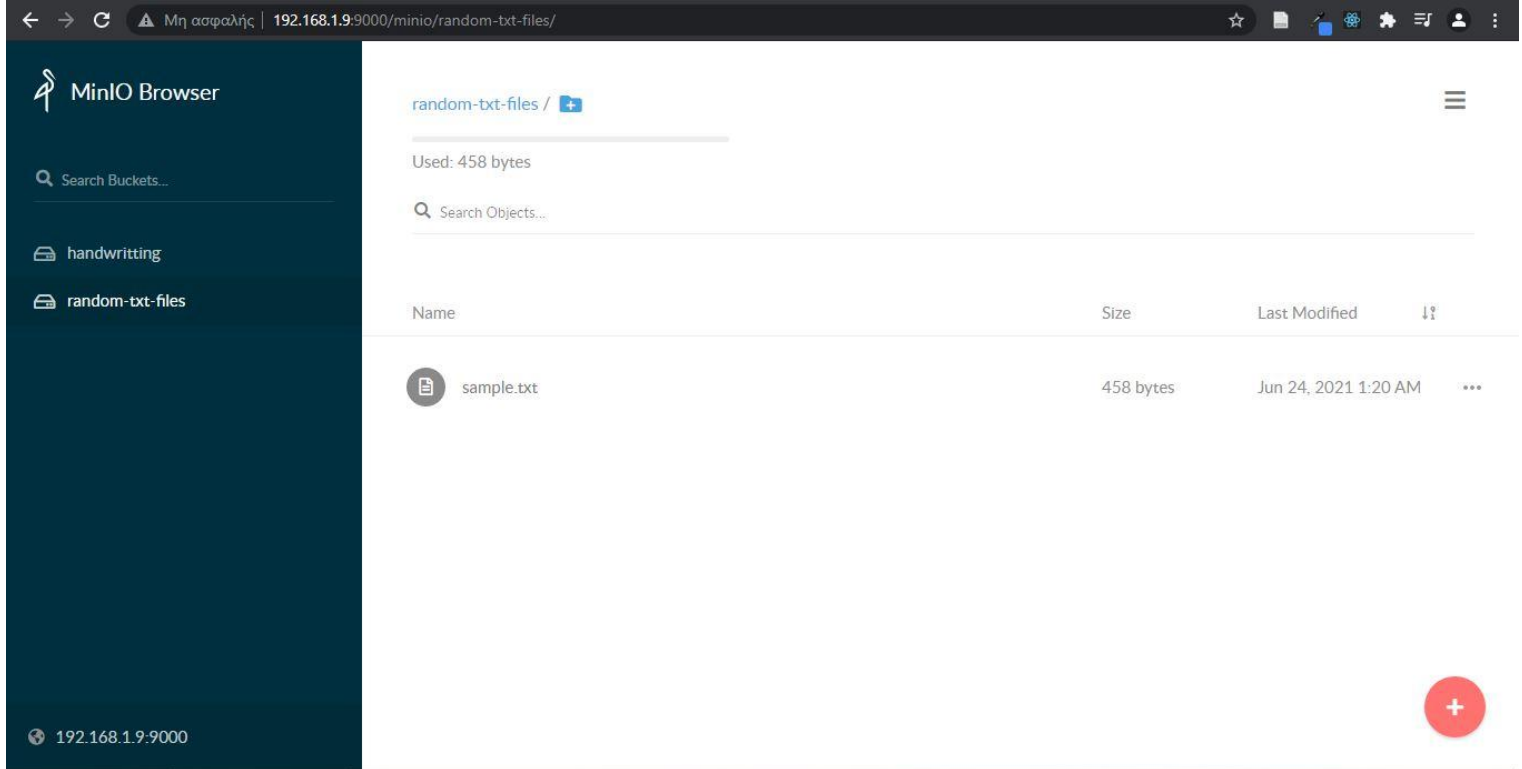
Επόμενος κόμβος που αξίζει να μελετηθεί είναι ο κόμβος της function που ακολουθεί στο flow



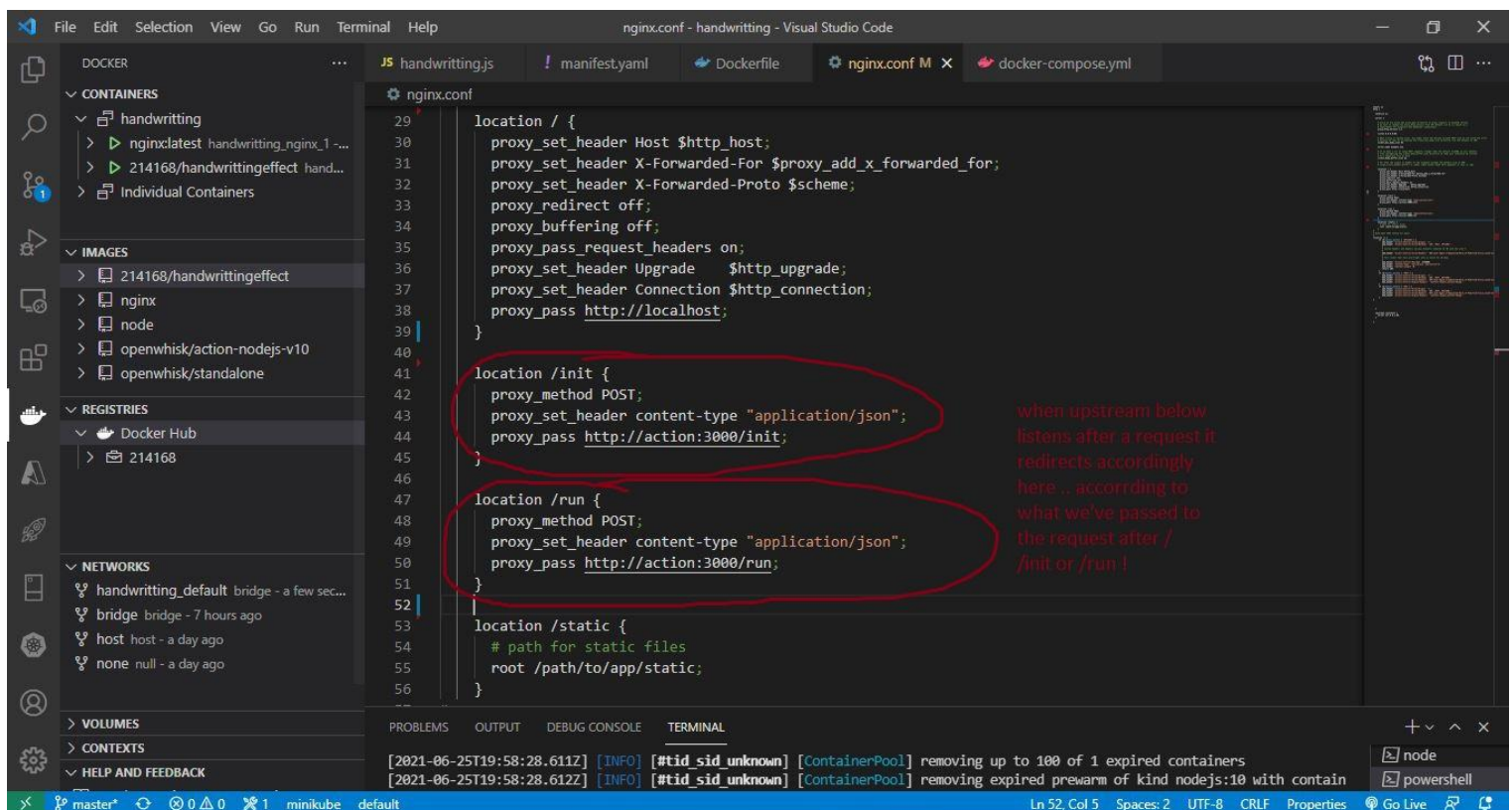
Εδώ απλώς πριν στείλουμε μέσω rest call το request το οποίο κουβαλάει πάνω του το presigned url τροποποιούμε λίγο το msg ώστε να προσθέσουμε πληροφορία . Την πληροφορία αυτήν στο header του msg την χρειαζόμαστε για να γίνει στη συνέχεια η σύνδεσή μας με το minio επιτυχώς ! στο payload έχω βάλει κλασικά το presigned url ώστε να γίνει proxied από τον nginx πίσω στον container στο /run endpoint όπου υλοποιεί. Η γραμμή 8 δεν είναι και πολύ αναγκαία νομίζω δεν την χρειάστηκα στον κώδικα πίσω τα έπαιρνα όλα τα data που ήθελα από το req.body parameter του endpoint που αντιστοιχεί στο μήνυμα που έρχεται από το msg.payload



Τελευταίος κόμβος που σημειώνεται είναι ο κόμβος του http request ώστε να γίνει το docker action invocation μέσω rest api call κι έτσι κλείνουμε την όλη υλοποίηση που έγινε στον client χρησιμοποιώντας τη διεπαφή του node-red. Αυτό που βλέπουμε να καλούμε εδώ είναι στην ουσία η διεύθυνση στην οποία ακούει ο nginx μόλις σηκώσουμε τον container τον οποίο βρίσκεται. Το /run είναι το τερματικό στο οποίο πέφτει το όλο ταξίδι του request μόλις φτάσει στον τελικό του προορισμό που είναι το docker action το οποίο docker action ακούει σε εσωτερική δική του διεύθυνση αλλά αυτό δεν μας νοιάζει διότι ο nginx τη βλέπει μέσω του docker-compose αρχείου που φτιάξαμε και στην ουσία είναι το όνομα που δώσαμε στο service αυτό, και χτυπώντας τον στο 127.0.0.1:80/run αυτός μας μεταφέρει στο docker_container_ip:3000/run κι έτσι στην ουσία γίνεται invoke και τρέχει το docker action ! εννοείται πως για να γίνει αυτό θα πρέπει πρώτα το docker action να έχει γίνει create .. περισσότερες λεπτομέρειες στην πορεία ! Μετά την εκκίνηση των πρώτων 2 flows στο minio έχουμε την εξής εικόνα



Ας παρουσιάσουμε τώρα λίγο γρήγορα τι γίνεται με τον nginx .. καταρχάς χρειαζόμαστε ένα αρχείο nginx.conf στο οποίο θα βάλουμε το configuration του στο οποίο οφείλεται το όλο proxied process ! τα κυκλωμένα είναι αυτά που κάνουν στην ουσία το redirection που προαναφέρθηκε ! Δεν θα απεικονιστεί όλο το configuration για λόγος διευκόλυνσης αλλά θα βρίσκεται στο repository που θα αναγράψω στο τέλος της σελίδας το οποίο θα κάνω public την ημέρα της παρουσίασης ..




```
nginx.conf
78
79 add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
80 add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control';
81 add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
82
83 if ($request_method = 'GET') {
84     add_header 'Access-Control-Allow-Origin' '*';
85     add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
86     add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control';
87     add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
88 }
89
90
91
92 upstream localhost {
93     server 127.0.0.1:80;
94 }
95
96
97
98
```

this upstream gets called on every request to this address and after that it maps with the previous locations the incoming address to the final one so it reach its destination

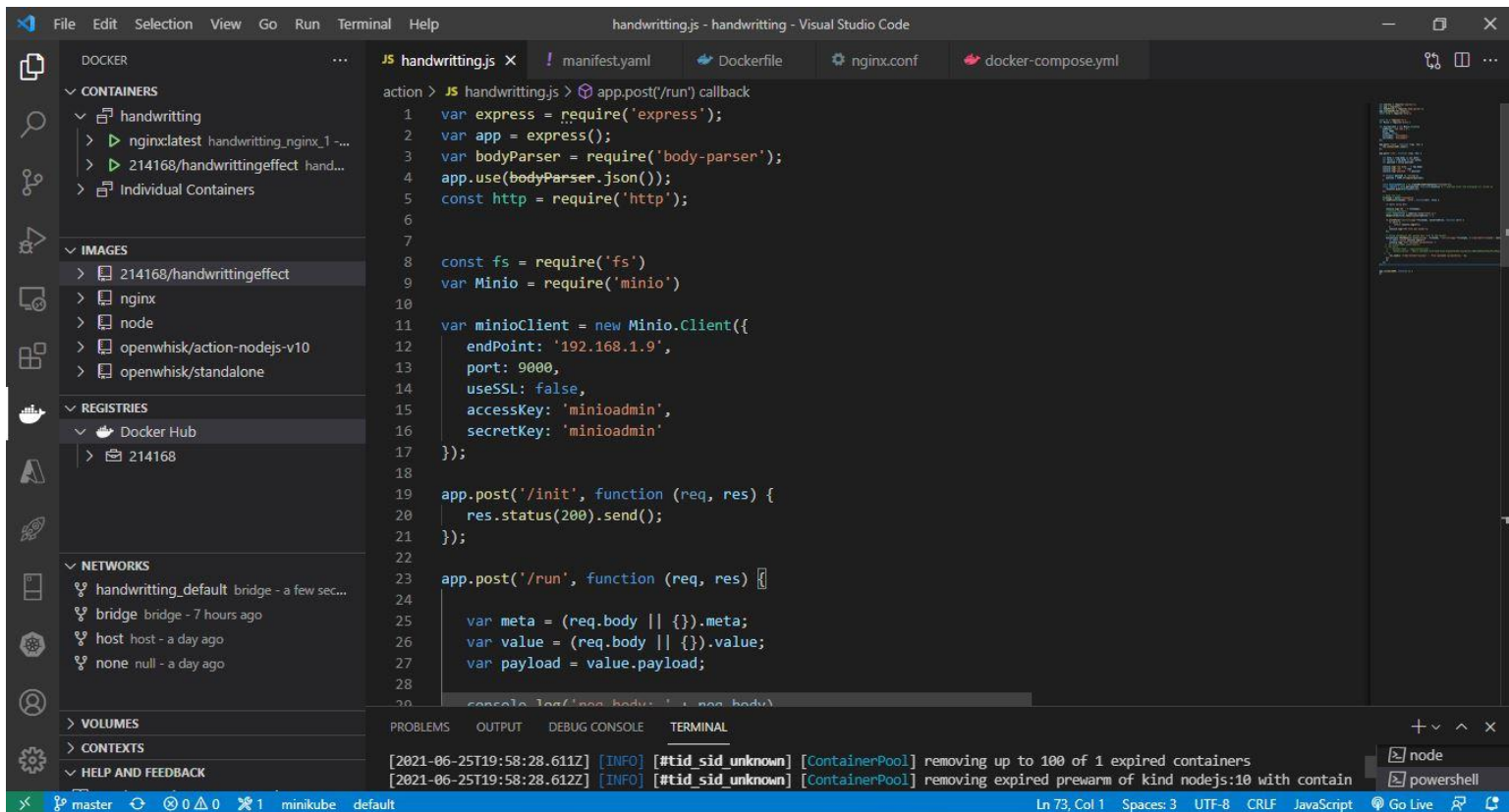
Όπως καταλαβαίνουμε ο nginx είναι ένα απαραίτητο component για τέτοιου είδους υλοποιήσεις ! αλλά δεν αρκεί μόνο αυτό το configuration file για να δουλέψει χρειάζεται και ο ίδιος ο server να έχει σηκωθεί σε container από πίσω ! Not to worry for the time being! Θα δούμε πως θα μπει και αυτό στην όλη υλοποίηση σε λίγο όταν θα δείξω στην πορεία το docker-compose file !

Πριν αφήσουμε τον nginx θα παραθέσω μια ωραία σύντομη πληροφορία που βρήκα από τον παρακάτω σύνδεσμο για αυτόν !

<https://thenewstack.io/behind-scenes-apache-openwhisk-serverless-platform/>

“This open source web server exposes the public-facing HTTP(S) endpoint to the clients. It is primarily used as a reverse proxy for API and also for terminating SSL. Every request hitting the OpenWhisk infrastructure, including those originating from the wsk CLI go through this layer. Since it is entirely stateless, the Nginx layer can be easily scaled out.”

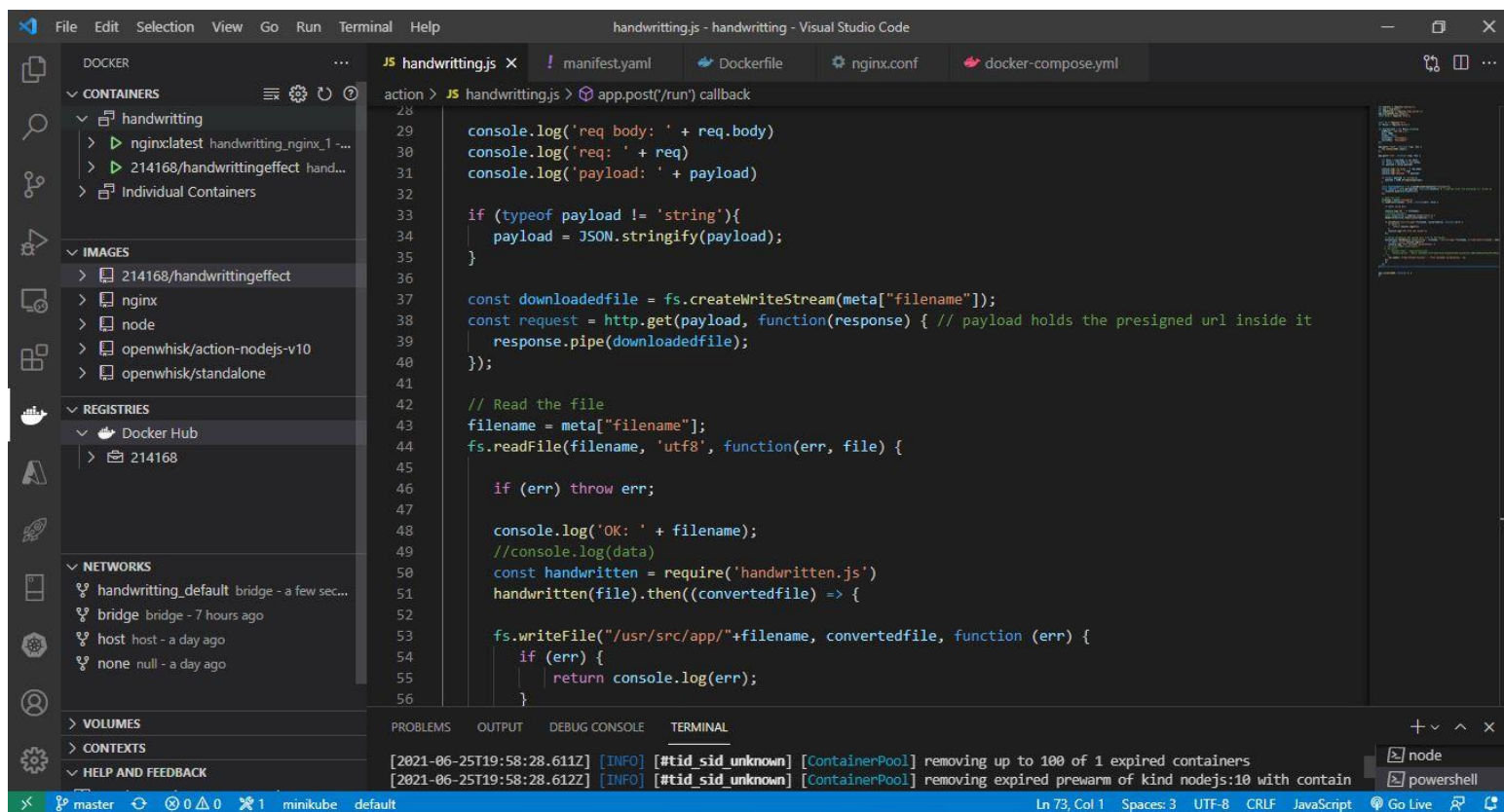
Σειρά έχει να δούμε λίγο τον nodejs express κώδικα που υλοποιεί την ουσία το φίλτρο μας !



Εδώ κάνουμε τα κατάλληλα imports και στη συνέχεια κάνουμε τη σύνδεση από τον minio client που έχουμε κανονίσει να εγκαταστήσουμε εντός του container από το dockerfile που θα δούμε στην πορεία και μετά ορίζουμε τα 2 endpoint μας το /init το οποίο στην προκειμένη περίπτωση ενημερώνει το χρήστη ότι το docker action δουλεύει και το /run που είναι το function του φίλτρου μας μέσα ! σε αυτό μας έρχεται το προαναφερθέν http request από το node-red και από αυτό παίρνουμε το payload το οποίο κουβαλά το presigned url ώστε να κατεβάσουμε το αρχείο στη πορεία και το όνομα του αρχείου που κατεβάζουμε από τα metadata ώστε να μπορούμε να το χρησιμοποιήσουμε ως παράμετρο όταν καλούμε την

```
minioClient.fPutObject()
```

ώστε να βάλει πίσω στο minio το φιλτραρισμένο αρχείο !



Τα console logs βοηθάνε στο debug όταν κάνουμε από το plugin αριστερά δεξί κλικ στον running container και μετά πατάμε view logs

Οι γραμμές 33-35 κάνουν το εισερχόμενο μήνυμα stringify αν δεν είναι !

Και στη συνέχεια γραμμές 37-40 αρχίζει να κατεβάζει το αρχείο στον τρέχοντα φάκελο εντός του container μετά γραμμές 43-το τέλος το διαβάζει από εκεί που το κατέβασε στη μνήμη και του περνά το handwritten φίλτρο στη γραμμή 51 επιστρέφει το converted file γιατί γίνεται ασύγχρονα και η γραμμή 50 ίσως έπρεπε να την έχω μαζί με τα άλλα imports, θα την μετακινήσω! Στην γραμμή 53 ξεκινά να γράφει το filtered file ξανά στο path /usr/src/app/ διότι η μέθοδος που θα το κάνει push στο minio θέλει να δει path αρχείου ως 3^η παράμετρο αναγκαστικά !


```
56 }
57 console.log("The file was saved!");
58 });
59
60 // Using fPutObject API upload your file to the bucket
61 minioClient.fPutObject('handwriting', filename, "/usr/src/app/"+filename, {'x-amz-meta-filename': meta
62 if (err) return console.log(err)
63 console.log('File uploaded successfully.')
64 // res.status(200).json(result);
65 // res.set({
66 //   'Content-Type': 'application/json',
67 //   'Authorization': 'Basic 23bc46b1-71f6-4ed5-8c54-816aa4f8c502:123z03xZCLrMN6v2BKk1dXVfPXL1Pkcc0Fqm1
68 // })
69 res.send({ 'X-Amz-Content-success' : 'File uploaded successfully.' });
70 });
71 });
72 });
73 });
74
75
76 app.listen(3000, function () {
77 })
```

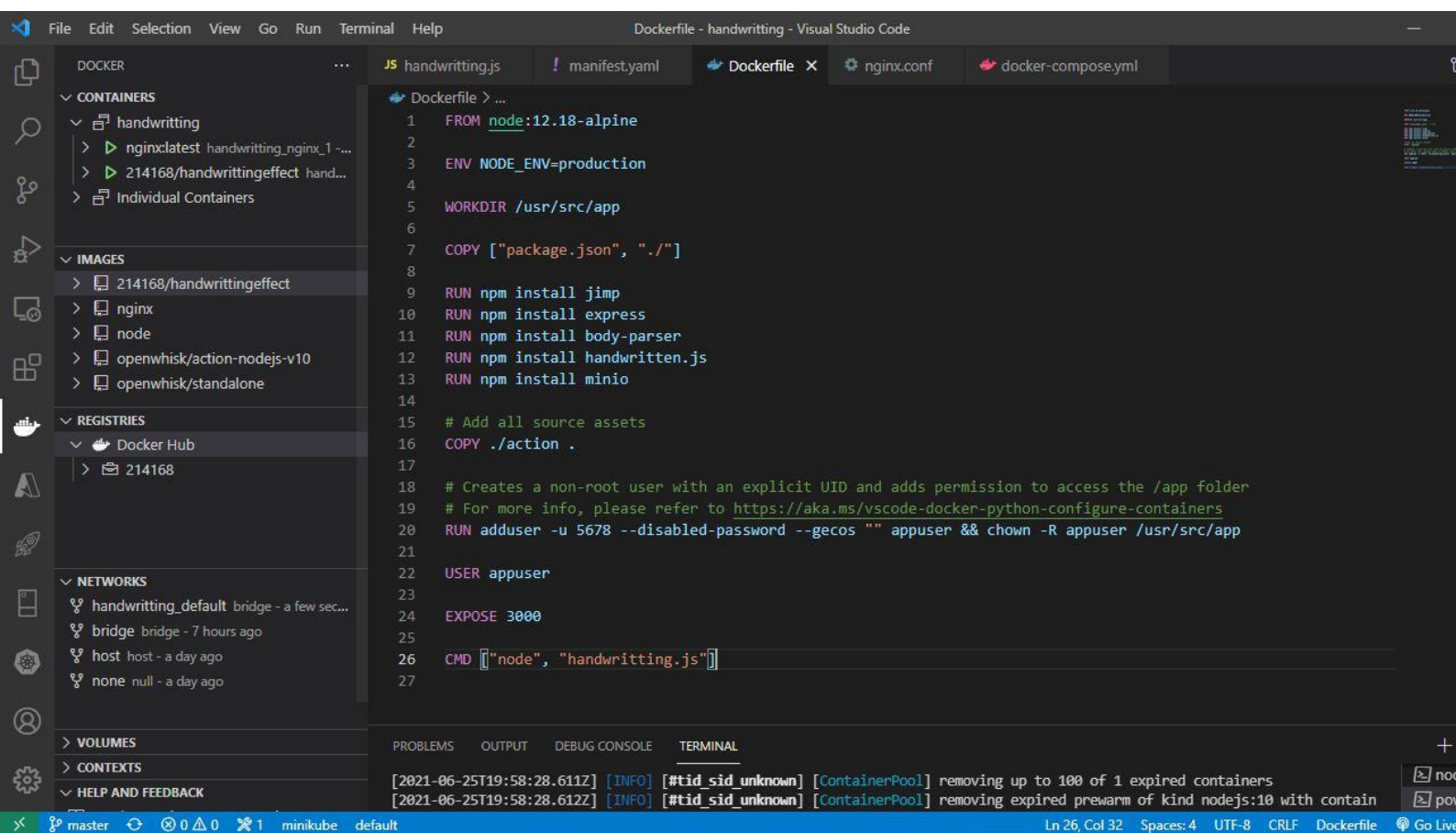
2021-06-25T19:58:28.611Z [INFO] [#tid_sid_unknown] [ContainerPool] removing up to 100 of 1 expired containers

2021-06-25T19:58:28.612Z [INFO] [#tid_sid_unknown] [ContainerPool] removing expired prewarm of kind nodejs:10 with contain

Γραμμές 61-73 βάζει στο minio το αρχείο όπως δηλώθηκε και στην 3^η ενότητα της αρχιτεκτονικής !

Στη γραμμή 76 ορίζουμε που θα ακούει όταν εκτελεστεί σε ποιο port ..

Τώρα ήρθε και η ώρα να δείξουμε που όλες αυτές οι βιβλιοθήκες αποθηκεύονται και μπορούμε και τις κάνουμε import σε αυτό το function αρχείο που μόλις περιγράψαμε ! Όλη αυτή η δουλειά γίνεται στο dockerfile με πραγματικά 2,3 βήματα ! ας το δούμε και αυτό λοιπόν !

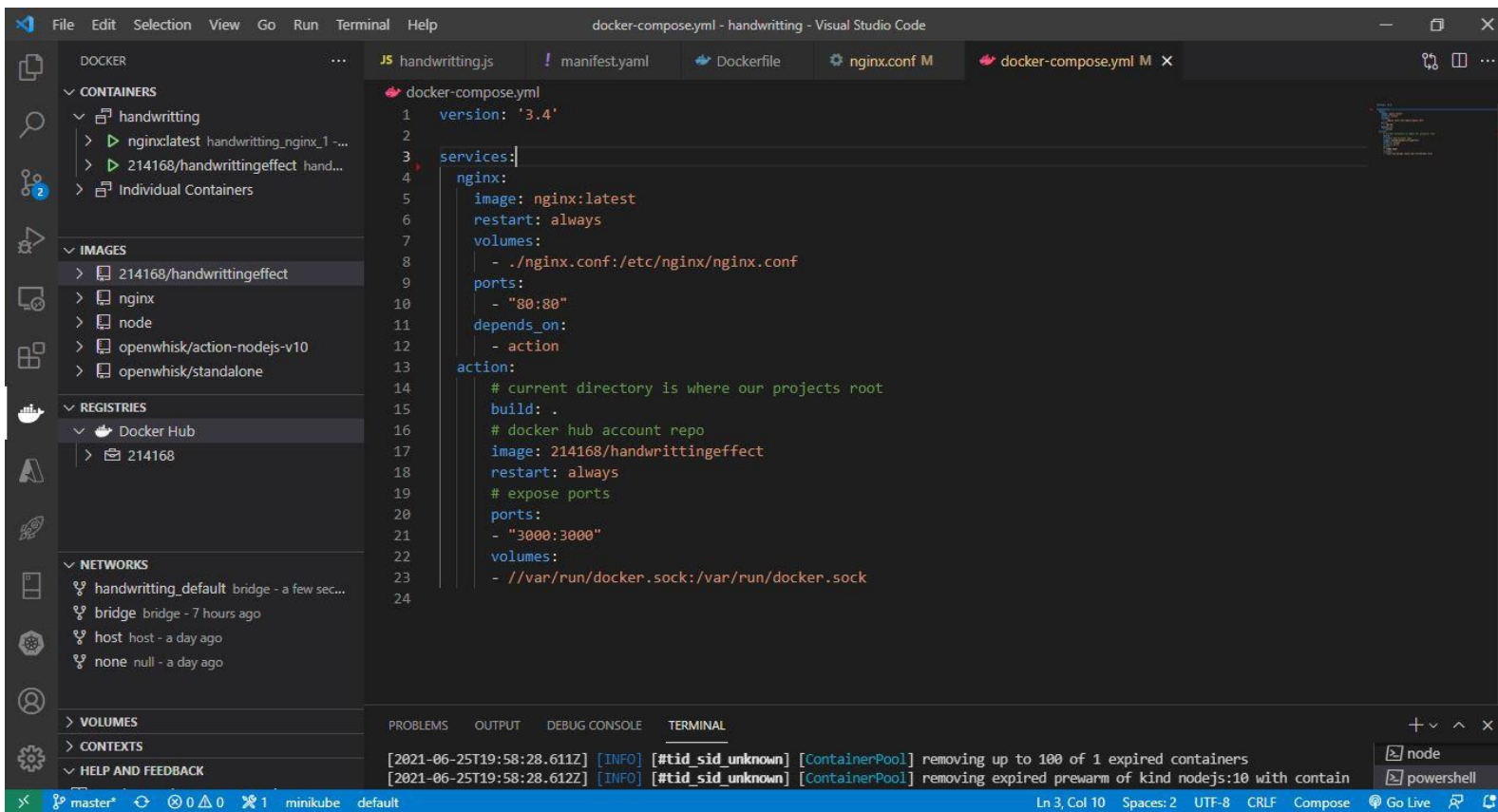


Στην 2^η γραμμή κατεβάζουμε το image του node τοπικά εντός του container ώστε να μπορούμε να έχουμε πρόσβαση και στο npm αργότερα που στις γραμμές 9-13 θα το χρειαστούμε ώστε να εγκαταστήσουμε όλες τις απαραίτητες βιβλιοθήκες κι έτσι να μπορούμε να τις κάνουμε import στο function μας όπως αναφέραμε προηγουμένως θα μπορούσαμε και μέσω του package.json να κάνουμε το installation αλλά λόγω του ότι ήταν λίγες οι βιβλιοθήκες το πήγα έτσι ..

Ορίζουμε και τον φάκελο που θα τα εγκαταστήσουμε όλα αυτά στη γραμμή 5 και κάνουμε αντιγραφή εκεί το package.json (αν και δεν το χρησιμοποίησα..) και τον φάκελο που βρίσκεται το function μας τοπικά στα windows τα κάνουμε αντιγραφή στον workdir φακελο μας που είναι ο /usr/src/app/

Ορίζουμε και νέο χρήστη για τα δικαιώματα για να μην είμαστε ο root χρήστης και έχουμε πρόσβαση σε ευαίσθητες δυνατότητες του container και κάνουμε ζημιά όπως περιγράφουν και τα σχόλια στις γραμμές 18, 19 .. κάνουμε expose το port και εκτελούμε την εντολή ώστε να εκκινήσει το function το node μας !

Προτελευταίο στη σειρά αλλά πολύ σημαντικό είναι το αγαπημένο σε όλους μας για την ευκολία που παρέχει με τα rules του στο να δημιουργηθούν όλοι οι container με τη μία (κάτι ιδιαίτερα επίπονο από το docker cli για έναν – έναν container ξεχωριστά μέσω εντολών) το docker-compose !



Απλό γρήγορο και εύκολο αρχικά ορίζει ένα service που λέγεται nginx κάτω από το όνομα που δίνεται στην γραμμή 4 κρύβεται το ip του container αυτού εσωτερικά του δικτύου που σηκώνει εντολή docker-compose up που εκτελείται μετά την docker-compose build προφανώς ! οπότε αν θέλουμε να καλέσουμε τη διεύθυνση του container αυτού σε κάποιον client (εντός του docker περιβάλλοντος να βρίσκεται ο client) αρκεί να βάλουμε στη θέση του host property το όνομα που δίνουμε στον container εδώ.. άρα τώρα μπορούμε να δούμε γιατί στο configuration του nginx στα redirected location είχαμε κάτι τέτοιο όπως στην 4^η γραμμή!

```
location /run {  
    proxy_method POST;  
    proxy_set_header content-type "application/json";  
    proxy_pass http://action:3000/run;  
}
```

Στην 5^η γραμμή κατεβάζουμε το image του nginx εντός του container που είχαμε πει ότι δεν φτάνει μόνο το configuration αρχείο προηγουμένως το επόμενο που χρειαζόμαστε λοιπόν είναι ακριβώς αυτό .. στη γραμμή 6 λέμε αν πέσει ο container να ξανασηκωθεί στη επόμενη γραμμή λέμε να κρατήσει εντός του container το configuration αρχείο που φτιάξαμε προηγουμένως το οποίο θα βρει στον φάκελο του project στα windows στο path ./nginx.conf και να το κάνει map αντιγράφοντας το αρχείο εκεί στο path /etc/nginx/nginx.conf στη γραμμή 11 του λέμε ότι πρώτα θα πρέπει να έχει σηκωθεί ο container του action πριν του nginx όπου μόλις σηκωθεί ο

nginx θα πρέπει να υπάρχει ήδη του action πάνω διότι κάποιος χρήστης μπορεί να στείλει κάποιο request οπότε ο nginx θα πρέπει να δει τον container του action τη στιγμή εκείνη έτοιμο ! τα ίδια βήματα ισχύουν και για τον άλλον container που δημιουργούμε παρακάτω στη γραμμή 13..

Το docker-compose βοηθάει να δούμε και αν η όλη συνδεσμολογία έγινε καλά πριν προχωρήσουμε το deploy σε άλλα συστήματα !

Τώρα κάνουμε docker-compose build

Και μετά docker login (αφού έχουμε φτιάξει το repo μας στο dockerhub)

Βάζουμε τα credential μας στον terminal στον οποίο και γράφουμε αυτές τις εντολές και έχουμε συνδεθεί στο repo μας μετά γράφουμε docker-compose push και θα ανεβάσει το image εκεί ..

Τώρα μπορούμε ως τελευταίο βήμα να βάλουμε και το openwhisk στη μέση

Έχοντας το wskdeploy λοιπόν μπορούμε να κάνουμε στο openwhisk deploy το image μας.. χρησιμοποιεί όμως ένα manifest.yaml αρχείο για να καταλάβει τι να κάνει deploy το οποίο και θα φτιάξουμε και έχει τη λογική του docker-compose οπότε δεν θα αναλύσω πολλά .. τα σχόλια στην εικόνα βοηθούν επίσης ως ανάλυση του τι γίνεται ! το hash μονο θα αναφέρω στη γραμμή 5 το οποίο παίρνουμε από τη μων εντολή που εμφανίζεται κατά την εκκίνηση του openwhisk μετά το flag --auth

Το namespace το βρήκα τρέχοντας την εντολή .\wsk namespace list στον terminal στο vs code αφού έκανα πρώτα cd στον φάκελο του openwhisk γιατί εκεί έχω το wsk cli

```
1 project:
2 # All of these params we will need to grab from your OpenWhisk credentials page
3 apiHost: http://localhost:3233
4 # example apiHost: openwhisk.ng.bluemix.net
5 credential: 23bc46b1-71f6-4ed5-8c54-816aa4f8c502:123z03xZCLrMN6v2BKK1dXYFpXlPkccOFqm12CdAsMgRU4VrNZ91yGVCGL
6 # credential is the same as our API Key
7 namespace: guest
8 # exmple namespace: cloud-foundry-org_cloud-foundry-space
9
10 packages:
11 # Packages group multiple actions into a package.
12 # We can call the packages whatever we'd like
13 MyOpenWhiskActionPackage:
14   version: 1.0
15   license: Apache-2.0
16   actions:
17     # We can list multiple actions here.
18     # We can call the actions whatever name we'd like ...
19     NodeAction:
20       docker: 214168/handwrittingeffect
21       # link to a dockerhub username/repo here
22       # We can add more actions below, we just need to uncomment the code and add our own...
23       # SecondAction:
24       # docker: my-username/docker-hub-repo
```

Τέλος τρέχω την εντολή `.\wskdeploy` στον φάκελο του `openwhisk` πάλι !

Έτσι έχω κάνει `create` το `action` μου και μπορώ να το κάνω τώρα `invoke` από το `nodered` εκκινώντας το `flow`. Θα παρατηρήσω ότι στον φάκελο `handwriting` υπάρχει το `filtered` αρχείο !

7. Πλεονεκτήματα / Μειονεκτήματα

Πλεονεκτήματα:

- Επειδή η όλη υλοποίηση βασίζεται στο `serverless` κομμάτι των `cloud` υπηρεσιών έχει το πλεονέκτημα ότι δεν απαιτείται η φροντίδα και το `maintainance` του `infrastructure` του `server` ούτε η τακτική του οργάνωση και ενασχόληση του προγραμματιστή με αυτό το κομμάτι ! οπότε η εφαρμογή είναι ευκολοδιαχειρίσιμη και επεκτάσιμη καθώς όλο το `maintainance` που προαναφέρθηκε ανήκει πλέον εξ' ολοκλήρου στον `service provider` ..
- Μπορεί να εξελιχθεί σε κατανεμημένο σύστημα και να δημιουργηθούν και άλλα φίλτρα τα οποία θα διευκόλυναν τους χρήστες διότι βασίζεται στη λογική των `containers` και έτσι διαμοιράζεται πιο εύκολα ανεξαρτήτου πλατφόρμας !
- Μπορεί να είναι προσβάσιμη μέσω `REST Api calls`

Μειονεκτήματα:

- Δεν υλοποιεί κάποιο `SLA (Service Level Agreement)` κι έτσι δεν μετράει με κάποια μετρική την απόδοση των `services` που παρέχονται ώστε να ενημερώσει τον χρήστη για την απόδοση της εφαρμογής που χρησιμοποιεί
- Δεν έχει υλοποιηθεί κάποιος τρόπος που να κάνει κάποιο από τα `components` `scalable`, οπότε σε μεγάλο όγκο δεδομένων ίσως δεν ανταποκρίνεται. Ο `Nginx` επειδή είναι `stateless` μπορεί πολύ εύκολα να γίνει `scalable` γι' αυτόν ακριβώς το λόγο! Επίσης και ο αριθμός των `containers`

8. Χρήσιμες docker commands που χρησιμοποιήθηκαν

`docker ps --format "{{.Names}}"`: all docker names of containers

`docker ps -a`: To see all the running containers in your machine.

`docker stop <container_id>`: To stop a running container.

`docker rm <container_id>`: To remove/delete a docker container(only if it stopped).

`docker image ls`: To see the list of all the available images with their tag, image id, creation time and size.

`docker rmi <image_id>`: To delete a specific image.

`delete rmi -f <image_id>`: To delete a docker image forcefully

`docker rm -f $(docker ps -a | awk '{print$1}')`: To delete all the docker container available in your machine

`docker image rm <image_name>`: To delete a specific image

To remove the image, you have to remove/stop all the containers which are using it.

`docker system prune -a`: To clean the docker environment, removing all the containers and images.

build docker image

`docker build -t image-name .`

run docker image

`docker run -p 80:80 -it image-name`

stop all docker containers

`docker stop $(docker ps -a -q)`

remove all docker containers

`docker rm $(docker ps -a -q)`

remove all docker images

`docker rmi $(docker images -q)`

Repository: <https://github.com/chara-b/cloudserviceslesson.git>

9. Credits

<https://medium.com/openwhisk/extending-openwhisk-to-the-iot-edge-with-node-red-docker-and-resin-io-bec7f30ea2de>

<https://openwhisk.apache.org/documentation.html#actions-nodejs>

<https://dzone.com/articles/running-openwhisk-actions-from-node-red>

<http://heidloff.net/article/how-to-create-docker-actions-openwhisk-bluemix>

<https://www.npmjs.com/package/handwritten.js>

<https://github.com/apache/openwhisk/blob/master/docs/actions-docker.md>

<https://www.npmjs.com/package/minio>

<https://github.com/apache/openwhisk/blob/master/docs/actions-docker.md>

<https://thenewstack.io/hands-guide-creating-first-serverless-application-apache-openwhisk/>

<https://nickjanetakis.com/blog/docker-tip-35-connect-to-a-database-running-on-your-docker-host>

<https://stackoverflow.com/questions/24319662/from-inside-of-a-docker-container-how-do-i-connect-to-the-localhost-of-the-mach>

<http://taswar.zeytinsoft.com/retry-pattern-in-node-js-using-promise/>

<https://medium.com/openwhisk/serverless-ruby-with-docker-and-apache-openwhisk-d64bc071e23a>

<https://thenewstack.io/behind-scenes-apache-openwhisk-serverless-platform/>

<https://stackoverflow.com/questions/11944932/how-to-download-a-file-with-node-js-without-using-third-party-libraries>