Distributed back-up system

Περίληψη

Ο σχεδιασμός και η αρχιτεκτονική ενός συστήματος, στις μέρες μας, επικεντρώνεται στο να είναι το σύστημα ικανό ανά πάσα χρονική στιγμή, να ανταπεξέλθει τυχόν αποτυχίας είτε σε επίπεδο κλήσης ενός feature είτε σε επίπεδο αποτυχίας ενός service είτε σε περίπτωση ολικής καταστροφής ενός server. Στις περιπτώσεις κατανεμημένων συστημάτων, έννοιες όπως high availability, single point of failure, consistency είναι μερικές οι οποίες λαμβάνονται σοβαρά υπόψη κατά την αρχιτεκτονική σχεδίαση του συστήματος. Στην παρούσα εργασία παρουσιάζεται η αρχιτεκτονική και η υλοποίηση ενός κατανεμημένου back-up συστήματος. Θα αναπτυχθεί πλήρως η επιλογή του εργαλείου που χρησιμοποιήθηκε για τους λόγους του demo, καθώς και 3 σενάρια χρήσης, κλιμακώνοντας την ανάγκη για back-up πληροφοριών, σημαντικών για την σωστή διαχείριση ενός συστήματος. Στην παρούσα εργασία θα επικεντρωθούμε μόνο σε περιπτώσεις αποτυχίας ενός service καθώς και σε αποτυχία ενός ολόκληρου κόμβου.

Ιδέα

Έστω ότι έχουμε ένα κατανεμημένο σύστημα αποτελούμενο απο N κόμβους, όπου 1 είναι ο Master κόμβος και οι υπόλοιποι N-1 είναι οι Workers κόμβοι. Το κατανεμημένο αυτό σύστημα έχει την ικανότητα να τρέχει κατανεμημένους αλγορίθμους σε N κόμβους. Για να είναι το σύστημά μας δυναμικό ο master κόμβος δεν είναι αναγκαίο να περιμένει τους υπόλοιπους N-1 κόμβους για να ξεκινήσει, αλλά ξεκινάει είτε workers είναι έτοιμοι να συνδεθούν με αυτόν, είτε όχι. Με αυτό τον τρόπο οι workers ανά πάσα χρονική στιγμή μπορούν να συνδεθούν με τον master κόμβο. Για να γίνει όμως αυτό, πρέπει να υπάρχουν κάπου πληροφορίες για το ήδη υπάρχον σύστημα όπως η IP του master κόμβου, στην οποία οι workers θα πρέπει να συνδεθούν. Για τον σκοπό αυτό έστω ότι χρησιμοποιείται ένα Key-Value Store το οποίο είναι υπεύθυνο να κρατά πληροφορίες σημαντικές κατά το start-up των κόμβων (και όχι μόνο). Όπως καταλαβαίνουμε, το ένα Key-Value Store αποτελεί και ένα single point of failure, πράγμα που σημαίνει ότι σε πιθανότητα failure, δεν μπορεί να υπάρξει καμιά δυνατότητα ανάκαμψης αφού το ίδιο δεν μπορεί να διαβάσει πληροφορίες παρά μόνο να κρατήσει πληροφορίες που γράφονται σε αυτό. Οι πληροφορίες αυτές, λοιπόν, είναι σημαντικό να παραμένουν σε όλη την διάρκεια ζωής του συστήματος και να ανανεώνονται όταν οι συνθήκες το επιβάλλουν (οι ίδιοι οι κόμβοι μπορούν να τροποποιήσουν τις πληροφορίες αυτές σε περίπτωση failure και ανάκαμψης του ίδιοι του συστήματος). Στη παρούσα εργασία, παρουσιάζονται 3 cases. Το

πρώτο case, περιγράφει ακριβώς αυτή την περίπτωση αποτυχίας και μηδενικής ανάκαμψης. Το δεύτερο case προσφέρει μια εν μέρει ανάκαμψη σε τυχόν αποτυχία.Τέλος, το τρίτο case παρουσιάζει ανάκαμψη σε οποιαδήποτε αποτυχία παρουσιάσει το σύστημα. Περισσότερα για τα cases θα βρείτε σε παρακάτω section.

Αρχιτεκτονική

Στην εργασία αυτή έχει χρησιμοποιηθεί σε ένα πολύ μεγάλο βαθμό η οικογένεια του Docker. Το docker αποτελεί ένα εργαλείο με το οποίο μπορεί να δημιουργηθεί ένα isolated περιβάλλον με το δικό του λογισμικό, συγκεκριμένα libraries, dependencies αλλά και applications που μπορούν να τρέξουν σε ενα container. Μέσω του docker, και συγκεκριμένα των Dockerfiles “πακετάρονται” όλες οι πληροφορίες σε ένα image για ένα απομονωμένο από το υπόλοιπο εξωτερικό περιβάλλον, υπο-περιβάλλον. ‘Αλλη μια ευκολία που χαρίζει το docker είναι ο robust τρόπος να γίνει deploy ένα τέτοιο υπό-περιβάλλον ανεξάρτητα από το περιβάλλον πάνω στο οποίο τρέχει. Τα container που δημιουργούνται τρέχουν ναι μεν πάνω στο λογισμικό που έχει επιλεχθεί, χρησιμοποιώντας τον kernel του host μηχανήματος. Τα Dockerfiles, είναι text documents που περιέχουν τις εντολές για τη δημιουργία ενός τέτοιου image. Πέρα από τα Dockerfiles, έχουν χρησιμοποιηθεί στην παρούσα εργασία και docker-compose αρχεία που είναι yaml αρχεία για τη δημιουργία services (με κάποιο Image που έχει ήδη δημιουργηθεί απο το Dockerfile), network, volumes και άλλα. Τέλος, έχει χρησιμοποιηθεί το docker swarm το οποίο δίνει την δυνατότητα δημιουργίας ενός ολόκληρου cluster με φυσικά ή virtual μηχανήματα πάνω στα οποία θα τρέξουν τα services τα οποία έχουμε δημιουργήσει μέσω των docker-compose αρχείων και των Images που έχουμε δημιουργήσει μεσω των Dockerfiles. Τα μηχανήματα που παίρνουν μέρος στο cluster ονομάζονται nodes, και η κατηγορία που ανήκουν ανάλογα με τα configuration που δίνονται κατά τη δημιουργία είναι managers ή workers.

Οσων αφορά το Raft Consensus όλοι οι managers που παίρνουν μέρος στο swarm, παίρνουν αυτόματα μέρος και στο Raft πρωτόκολλο. Αυτό σημαίνει ότι ένας από αυτούς μετά απο Election, γίνεται ο Leader και είναι υπεύθυνος για ανάκαμψη του swarm σε περίπτωση failure ενός ή και περισσότερων services αλλά και ενός ολόκληρου κόμβου. Ένα swarm με 3 managers οι οποίοι είναι και μέρος του Raft πρωτοκόλλου μπορούν να ανακάμψουν μέχρι και 1 manager που κάνει fail. Για 5 managers του swarm μπορεί να γίνει ανάκαμψη μέχρι και σε 2 managers που κάνουν fail κ.ο.κ. Γενικά, το Raft μπορεί να κάνει tolerate μέχρι και (N-1)/2 αποτυχίες και χρειάζεται τουλάχιστον (N/2)+1 μέλη να συμφωνούν με τιμές που προτείνονται από το cluster.

(πηγή https://docs.docker.com/engine/swarm/raft/)

Ένα ακόμα πλεονέκτημα του docker swarm είναι ότι για λόγους high availability ένας από τους managers του swarm (και Leader του Raft) κάνει deploy τα services με τέτοιο τρόπο ώστε όσο το δυνατόν λιγότερα services να βρίσκονται στον ίδιο κόμβο. Έτσι σε τυχόν αποτυχία ενός κόμβου αναλαμβάνει να σηκώσει ξανά τον μικρότερο δυνατό αριθμό από services. Για παράδειγμα, αν έχουμε 3 services και αντίστοιχα 3 κόμβους, τότε ο κάθε ένας κόμβος

αναλαμβάνει να τρέξει από 1 service. Αν προστεθούν και άλλα 3 services, ξανα με κυκλική διαδικασία αναλαμβάνει ο κάθε κόμβος απο 1 service ακόμα (σύνολο 2) κ.ο.κ.

Για τους λόγους του demo χρησιμοποιήθηκε επίσης το εργαλείο docker-machine για τη δημιουργία virtual μηχανημάτων με την οικογένεια docker εγκατεστημένη σε αυτά, τα οποία γίνονται manage μέσω των docker-machine εντολών που ακολουθούν την δική τους σύνταξη.

Υλοποίηση

Όπως προείπαμε για τους λόγους του demo χρησιμοποιήθηκαν τα docker-machines. Σε αυτό το κομμάτι θα παρουσιάσουμε τον τρόπο με τον οποίο δημιουργήθηκαν τα docker-machines, το docker swarm, οι Managers και Workers κόμβοι του καθώς και το εσωτερικό δίκτυο πάνω στο οποίο θα τρέχουν τα services.

Δημιουργία docker-machines, docker swarm, internal network

Από το host machine εκτελώντας την εντολή:

docker-machine create --driver virtualbox myMaster1

δημιουργούμε το πρώτο docker-machine με όνομα *myMaster1*. Ακολουθούν επίσης τα *myMaster2*, *myMaster3*, myWorker1, *myWorker2*.

Από το host machine εκτελώντας την εντολή:

docker-machine ssh myMaster1

Μπαίνουμε στο περιβάλλον του συγκεκριμένου docker-machine και εκτελώντας την εντολή:

docker swarm init -advertise-addr=192.168.99.100:2377

κάνουμε initialize το swarm. Επίσης, στο ίδιο περιβάλλον εκτελώντας τις εντολές:

docker swarm join-token manager (1)

docker swarm join-token worker (2)

παίρνουμε τις εντολές για να κανουμε join στο swarm κάποιον master και κάποιον worker, αντίστοιχα.

Στα αντίστοιχα περιβάλλοντα των μηχανήματα *myMaster2*,*myMaster3* εκτελούμε:

docker swarm join --token SWMTKN-1-34q6usqdrykc2o5zu1jeio4yurmxn4t7tkk1lff3k48x3iai7k-f3c3ms0j4bdmrw9kttz8m1hj6 192.168.99.101:2377

Το οποίο είναι και το αποτέλεσμα της εντολής (1).

Ενώ στα αντίστοιχα μηχανήματα *myWorker1, myWorker2* εκτελούμε:

docker swarm join --token SWMTKN-1-34q6usqdrykc2o5zu1jeio4yurmxn4t7tkk1lff3k48x3iai7k-cve2waz5fopdq0ub3zf2gcnj3 192.168.99.104:2377

Το οποίο είναι το αποτέλεσμα της εντολής (2).

Έτσι, έχουμε δημιουργήσει ένα swarm όπου τα docker-machines *myMaster1*,*myMaster2*,*myMaster3* είναι οι managers και τα *myWorker1, myWorker2* οι workers του swarm.

Για τη δημιουργία του εσωτερικού δικτύου μέσα στο περιβάλλον οποιουδήποτε docker-machine που είναι manager του swarm εκτελούμε:

docker network create --driver=overlay --subnet=10.20.30.0/24 --ip-range=10.20.30.0/24 --gateway=10.20.30.254 my-network

Case1

Όπως προείπαμε, ο ρόλος του Leader στο docker swarm είναι να μπορεί να διατηρεί το ίδιο state σε όλο το swarm. Δηλαδη, όλα τα services να είναι alive ακόμα και μετά από κάποιο fail των ίδιων ή ακόμα και μετά από ένα fail ενός ολόκληρου κόμβου να μπορεί να γίνει ανάκαμψη των δικών του services με το να τρέξουν σε κάποιο άλλο κόμβο. Η περίπτωση που θα περιγράψουμε στη συνέχεια αποτελεί και την πλήρη αδυναμία ανάκαμψης από τυχόν fail του single point of reference που είναι το Key-Value Store.

Όπως μπορούμε να δούμε και στον φάκελο case1 που βρίσκεται εδώ https://github.com/sofiakarb/distributed_systems/tree/master/case1 έχουμε 2 docker-compose αρχεία, ένα για τον master κόμβο στον οποίο θα τρέξουν 2 services (ο Master του κατανεμημένου συστήματος και το Key-Value Store) και ένα για τον worker στον οποίο θα τρέξει ένα service (ο worker του κατανεμημένου συστήματος).

Σε αυτό το σημείο έχει ένα ενδιαφέρον να παρατηρήσουμε τα εξής:
Για το *docker-compose-manager.yaml* αρχείο βλέπουμε αρχικά το network “my-netwrork” στο οποίο θα τρέξουν τα 2 services (συν το service του *docker-compose-worker.yaml* αρχείου). Επίσης, παρατηρούμε ότι μπορούμε να θέσουμε constraints όπως το σε ποιο κόμβο να ξεκινήσει το service ανάλογα με τον ρόλο του κόμβου. Από τη στιγμή που δεν έχουμε θέσει mode (replicated ή global) για το πρώτο service, η default επιλογή είναι replicated με 1/1, δηλαδή το service να τρέξει σε ένα κόμβο. Όσων αφορά το δεύτερο service, έχουμε θέσει ξανά ένα περιοριστικό constraint για το που θα ξεκινήσει το service αυτό, θέτοντας το mode να είναι global που έχει την ίδια ιδιότητα με το replicas:1 (ή αλλιώς replicated 1/1). Ξανά, δηλαδή, μόνο ένα τέτοιο service στο swarm.

Αφού ξεκινήσαμε τα services εκτελώντας την παρακάτω εντολή μέσα από το περιβάλλον του docker-machine *myMaster1*:

docker stack deploy -c docker-compose-manager.yaml myMaster

Βλέπουμε ότι τα 2 services έχουν ξεκινήσει και πως το Key-Value Store έχει πλέον τα κλειδιά και τις τιμές που έχει θέσει ο master κόμβος του κατανεμημένου συστήματος. Αυτό μπορούμε να το δούμε εκτελώντας στο host machine την παρακάτω εντολή:

curl http://192.168.99.100:8500/v1/kv/?keys

Εκτελώντας την εντολή:
docker stop myMaster_exareme-keystore ή
docker kill myMaster_exareme-keystore ή
docker swarm leave -f (3)

το αποτέλεσμα θα είναι ο Leader του Raft πρωτοκόλλου να ξεκινήσει ξανά το service που μόλις έκανε fail είτε στον ίδιο κόμβο που έτρεχε το service αν αυτός είναι alive, είτε σε κάποιον άλλον (περίπτωση 3 όπου ο κόμβος δεν είναι alive).
Εκτελώντας ξανά την εντολή:
curl http://192.168.99.100:8500/v1/kv/?keys
Παρατηρούμε ότι το αποτέλεσμα δεν είναι το ιδανικό. Τα Key-Values έχουν χαθεί μετά το fail του service επομένως και το ίδιο το state πριν και μετα το fail έχει χαθεί. Άρα, το σύστημα πλέον **δεν** είναι σε consistent μορφή και το single point of reference δημιούργησε προβλήματα.

Case2

Στην περίπτωση αυτή έχει δημιουργηθεί ένα ακόμα service, το backup service. Τον κώδικα μπορείτε να δείτε εδώ https://github.com/sofiakarb/distributed_systems/blob/master/case2/backup.sh. Το Dockerfile για την δημιουργία του image *sofiakar/myservice:case2* μπορείτε να βρείτε εδώ https://github.com/sofiakarb/distributed_systems/blob/master/case2/Dockerfile. Στο *docker-compose-manager.yaml* που βρίσκεται εδώ https://github.com/sofiakarb/distributed_systems/blob/master/case2/docker-compose-manager.yaml μπορούμε να δούμε το καινούριο service με τα δικά του χαρακτηριστικά με άξιο αναφοράς να είναι το *mode:global* (να τρέξει ακριβώς ένα service, ακόμα και μετά απο κάποιο fail). Απο την άλλη βλέπουμε πως στο https://github.com/sofiakarb/distributed_systems/blob/master/case2/docker-compose-worker.yaml yaml αρχείο του worker έχουμε βάλει σαν constraints το service να ξεκινήσει σε κόμβο με ρόλο wokrer, αλλά και το όνομα του κόμβου να είναι συγκεκριμένο. Με την εντολή

docker node update --label-add name={name} {node_hostname}

οπου name το όνομα που εμείς επιλέγουμε και node_hostname το hostname του κόμβου, μπορούμε να τροποποιήσουμε το όνομα ενός κόμβου.

Η διαφορά με το προηγούμενο case, είναι πως τώρα ανά τακτά χρονικά διαστήματα το backup service αντλεί τις πληροφορίες από το Key-Value Store και τις αποθηκεύει σε ένα αρχείο μέσα στο container. Αυτό δίνει την δυνατότητα, να γίνονται sync ανα τακτά χρονικά διαστήματα το Key-Value Store με το αρχείο, σε περίπτωση που το Key-Value Store κάνει fail, αλλά και το ίδιο το αρχείο με το Key-Value Store σε περίπτωση που νέες εγγραφές υπάρξουν. Κάνοντας λοιπόν την ίδια διαδικασία με παραπάνω, βλέπουμε ότι το Key-Value Store δεν χάνει το consistency ακόμα και μετα το fail που έγινε είτε στο ίδιο το service Key-Value Store (*docker stop myMaster_backup ή docker kill myMaster_backup*) είτε στον κόμβο που εκείνο τρέχει (*docker swarm leave -f*).

Υπάρχει όμως μια λεπτή γραμμή στην οποία τα 2 services (backup service και Key-Value Store service) να κάνουν fail ταυτόχρονα. Αυτό θα μπορούσε να γίνει με το να είχαν γίνει deploy στο ίδιο κόμβο (τροποποιώντας τα constraints) ο οποίος έκανε fail ή με ταυτόχρονο fail για κάποιο άλλο λόγο και ας ζουν σε διαφορετικούς κόμβους (αυτό παρουσιάζεται και στο demo).

Case3

Αυτήν την τελευταία οριακή περίπτωση έρχεται να λύσει το case3. Μπορείτε να δείτε τον τροποποιημένο κώδικα για το backup service εδώ https://github.com/sofiakarb/distributed_systems/blob/master/case3/backup.sh

Αντίστοιχα στο *docker-compose-manager.yaml* αρχείο που βρίσκεται εδώ https://github.com/sofiakarb/distributed_systems/blob/master/case3/docker-compose-manager.yaml άξιο αναφοράς αυτή τη φορά είναι το volume που θα δημιουργηθεί κατα το start-up του service.

```
volume:
          - /home/myMaster/backup/:/home/backup/
```

Αυτό ουσιαστικά σημαίνει ότι στο docker-machine πάνω στο οποίο τρέχει το service backup, θα γίνει map ο φάκελος */home/myMaster/backup/* με τον φάκελο */home/backup/* ο οποίος θα υπάρχει και ζει μέσα στο container του backup service. Μέσα στον φάκελο */home/backup/* υπάρχει το json αρχείο με τα Key Values που το service έχει αντλήσει απο το Key-Value Store. Επειδή το συγκεκριμένο αποτελεί volume κάθε φορά που γίνεται αλλαγή στο αρχείο json μέσα στο container του backup service (*/home/backup/)*, αυτόματα αλλάζει και το αρχείο στο path (*/home/myMaster/backup/)*μέσα στο docker-machine. Σε αυτό το case3, επειδή ο τρόπος failure μας ενδιαφέρει θα δούμε τις περιπτώσεις ξεχωριστά.

Περίπτωση 1) Ταυτόχρονο Fail των services με *docker stop ή docker kill*

Στην περίπτωση αυτή, επειδή οι κόμβοι θα είναι alive, τα services θα ξεκινήσουν να τρέχουν ξανά στους ίδιους. Το map που έχει γίνει μέσω του volume στο *docker-compose-manager.yaml*, θα υπάρχει και κατά το ξεκίνημα του backup service αμέσως μετά το fail. Επομένως, παρόλο που ξεκινώντας το container, το αρχείο json *κανονικά* δεν υπάρχει, μέσω του volume όπου γίνονται αμέσως map οι 2 folders, οι πληροφορίες που υπάρχουν στο path του docker-machine */home/myMaster/backup/* "αντικατοπτρίζονται" και στο path μέσα στο container */home/backup/* . Αυτό σημαίνει ότι το state δεν έχει χαθεί ακόμα και σε ταυτόχρονο fail των 2 services.

Περίπτωση 2) Fail του bakcup service με *docker swarm leave*

Στην περίπτωση αυτή, ο κόμβος στον οποίο ζούσε το backup service δεν θα είναι πια alive, επομένως και το service θα ξεκινήσει να τρέχει σε έναν από τους άλλους 2 κόμβους. Για τον λόγο αυτό υλοποιήθηκε ενα script του οποίου τον κώδικα μπορείτε να βρείτε εδώ https://github.com/sofiakarb/distributed_systems/blob/master/case3/copy_backup_file.sh κατά το οποίο γίνεται copy το αρχείο που υπάρχει στον φάκελο */home/myMaster/backup/* στο docker-machine στου οποίου τον κόμβο τρέχει το backup service, σε όλα τα docker-machines όπου οι κόμβοι τους είναι managers του swarm. Με αυτό τον τρόπο οι πληροφορίες έχουν διανεμηθεί σε όλους τους managers, και έτσι οπουδηποτε και να ξεκινήσει το backup service θα μπορέσει να βρει τις πληροφορίες και να κρατήσει το ίδιο state πριν αλλά και μετά το fail.

Ανακεφαλαίωση

Με τον τρόπο που παρουσιάστηκε παραπάνω, σχεδιάσαμε ένα distributed backup system το οποίο πλέον διατηρεί την κατάσταση του συστήματος σε όλες τις περιπτώσεις fail, είτε ενός μόνο service, είτε ταυτόχρονου fail από services, είτε κάνοντας fail ένας ολόκληρος κόμβος. Δείξαμε πως ένα single point of failure που στην περίπτωση αυτή ήταν το Key-Value Store, μπορεί να δημιουργήσει προβλήματα σε ένα σύστημα που χρειάζεται τις πληροφορίες καθόλη τη διάρκεια της ζωης του και πως δεν είναι πάντα εύκολο να διατηρηθεί το ίδιο state πριν και μετά απο failures(case1,case2). Επίσης, χρησιμοποιήθηκε η οικογένεια του docker, για δημιουργία Images, services καθώς και volumes. Και τέλος, το docker swarm το οποίο πέρα απο το Raft Consensus που χρησιμοποιούν οι Managers κόμβοι που είναι μέλη στο swarm, δίνει ένα high availability στον τρόπο που ο Leader κόμβος κάνει deploy τα services με τέτοιο τρόπο ώστε ο κάθε κόμβος να τρέχει τον μικρότερο δυνατό αριθμό από services (φυσικά πάντα συμφωνώντας στα constraints που έχουμε επιβάλει στα docker-compose αρχεία) έτσι ώστε σε περίπτωση fail ενός ολόκληρου κόμβου να μπορέσει ο Leader να σηκώσει τον μικρότερο δυνατό αριθμό από services.