COME CREARE UN REPLICASET DI MONGODB CON SELF-HEALING USANDO I SERVIZI DI AWS

MongoDB

beSharp | 19 Aprile 2019

MongoDB è un database NOSQL documentale immediato da configurare e da utilizzare e sta avendo una grandissima diffusione nei nuovi progetti di sviluppo. In questo articolo vedremo come sia possibile creare un sistema di **self healing** per i cluster di Mongo creati su AWS in modo da prevenire un possibile downtime.

MongoDB consente di creare un **cluster distribuito in alta affidabilità** tramite i Replica Sets e lo Sharding:

- **I Replica Sets** sono insiemi di nodi identici su cui vengono replicati i documenti JSON inseriti nel database.
- **Lo Sharding** consente di scrivere ogni insieme di documenti JSON (Collections) su un server differente, garantendo quindi un miglior throughput sia in lettura che in scrittura.

La configurazione in Sharding non è tollerante ad alcun fallimento, infatti in caso di down di una macchina le Collection presenti sulla stessa diventano inaccessibili. Per ovviare a questo problema si opta per una **configurazione mista basata su Sharding e Replication** dove ogni nodo dello sharding è in realtà un Replica Set.

Ogni Replica Set è composto da un Master che si occupa di eseguire le operazioni di scrittura e lettura e da almeno due Repliche che non accettano direttamente scritture ma vengono tenute sincronizzate con i dati presenti nel Master e possono eseguire solo operazioni in lettura. Ogni nodo comunica agli altri continuamente (ogni 2 secondi) il proprio stato tramite un sistema di heartbeats e in caso di fallimento del master i nodi secondari eseguono una votazione con cui viene eletto il nuovo Master.

Affinché la votazione avvenga correttamente tuttavia devono partecipare alla votazione **più del 50% dei nodi che compongono il cluster.**

Nel caso di un deploy su AWS è pertanto consigliabile **distribuire equamente le EC2 su almeno tre Availability Zones** (AZ) se si desidera che il Replica Set resti funzionante anche in caso di down di una AZ.

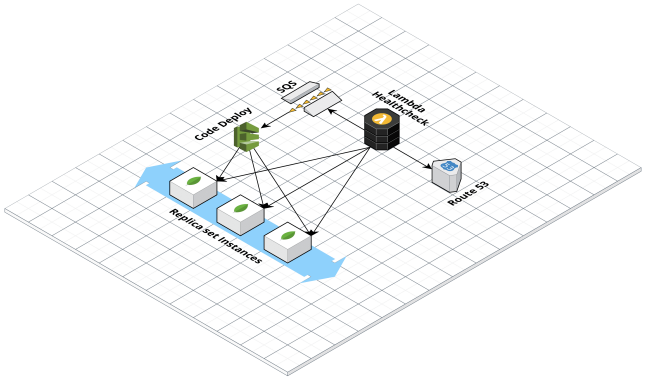
Per database di dimensioni contenute e non *read intensive,* oppure per database con Sharding, molto spesso si usa una configurazione con tre nodi per il Replica Set. In questa configurazione in caso di fallimento di un nodo ne rimangono sempre disponibili altri due per la votazione, uno dei quali verrà promosso a master.

Tuttavia, soprattutto in condizioni di traffico intenso, il fallimento di un nodo deve essere trattato come un emergenza e risolto il prima possibile in quanto qualunque ulteriore fallimento comporterebbe una interruzione del servizio.

Per risolvere in modo automatico questo tipo di situazione e consentire a chi si occupa di gestire il db di portare a termine l'analisi post mortem dell'istanza mongodb fallita abbiamo implementato un **sistema di self-healing per mongodb** in grado di ricreare automaticamente un istanza identica a quella non più raggiungibile e riconfigurare il replica set per usare la nuova EC2 al posto di quella corrotta.

Gli strumenti di AWS utilizzati per implementare questa soluzione sono i seguenti:

- **Autoscaling group:** un servizio di AWS per gestire e scalare gruppi di istanze; è possibile definire il tipo di istanza da utilizzare e il numero di istanze desiderate e AWS si occuperà di lanciarle, monitorare lo status e se necessario terminarle e lanciarne di nuove.
- **Route 53:** il servizio DNS di AWS
- **Lambda Functions:** Funzioni serverless che possono essere lanciate su AWS come risposta all'attivazione di un trigger oppure con una cadenza definita usando AWS CloudWatch
- **SQS Queues:** Code di messaggi in alta affidabilità gestite da AWS.
- **AWS CodeDeploy:** Servizio gestito di AWS per il deploy di codice arbitrario tramite un agent installato sulle macchine, si integra con gli autoscaling group: il deploy viene eseguito nel momento in cui una nuova istanza viene avviata



Creazione del Replica Set

Per procedere nella **creazione del Replica set con self healing** è innanzi tutto necessario creare un AMI di base per le macchine del replica set con tutti i pacchetti necessari installati: AWSCLI,

Chrony, Ruby e ovviamente MongoDB (master e client) nella versione desiderata è inoltre necessario istallare il **CodeDeploy agent.**

Per configurare il replica set è necessario creare un **keyfile** che deve essere presente su tutte le macchine cluster e che mongo utilizza per autenticare le connessioni dell' heartbeat. Il file deve contenere caratteri alfanumerici generati in modo pseudo-random e può essere creato col comando

```
openssl rand -base64 756 > <path-to-keyfile>
chmod 400 <path-to-keyfile>
```

Il keyfile è una forma minimale di sicurezza ed è accettabile solo su una rete sicura (subnet private in VPC), per l'ambiente di produzione è comunque preferibile usare un certificato x.509. Il file di configurazione di MongoDB sarà simile al seguente:

```
storage:
  dbPath: /mnt/mongo/data
  journal:
    enabled: true

systemLog:
  destination: file
  logAppend: true
  path: <logs-path>

security:
  keyFile: <keyfile-path>

replication:
  replSetName: test-rs

net:
  port: 27017
  bindIpAll: true
```

La creazione iniziale del cluster MongoDB può essere completamente automatizzata usando **CloudFormation**, oppure può essere eseguita manualmente utilizzando l'ami appena creata e uno script di configurazione da eseguire sulle singole macchine del cluster.

Una delle macchine va configurata inizialmente per essere il master e inizializzare il Replica Set. Per fare ciò è necessario prima avviare Mongo senza i campi security e replication nel file di configurazione e poi creare un utente admin col comando:

```
db.createUser({user: "<ADMIN_USER>", pwd: "<PWD>", roles:[{role: "root", db: "admin"
}]});
```

A questo punto Mongo va riavviato col file di configurazione contenente security e replication.

Dopo aver acceso anche le altre macchine, che inizialmente saranno repliche, è necessario creare degli *A name* su una private hosted zone di R53 per tutte le macchine del replica set. Nelle configurazioni della VPC è inoltre necessario selezionare le opzioni enableDnsHostnames e enableDnsSupport per attivare la risoluzione dei DNS interni di R53.

Fatto ciò è finalmente possibile connettersi in SSH al master e tramite la cli di mongo attivare il replica set col comando:

```
rs.initiate( {
   _id : "test-mongo-rs",
   members: [
      { _id: 0, host: "mongodb-1.test.it:27017" },
      { _id: 1, host: "mongodb-2.test.it:27017" },
      { _id: 2, host: "mongodb-3.test.it:27017" }
   ]
});
```

Ora che il Replica Set è configurato e funzionante è giunto il momento di occuparci della parte di self healing.

Set up del Self Healing

Per configurare il selfhealing è innanzitutto necessario **registrare le macchine ad un autoscaling group con healthcheck di tipo EC2 e numero di macchine fisso** (Max: 3, Min: 3 e Desired: 3). In questo modo in caso di fallimento di una macchina AWS provvederà a creare una nuova istanza partendo da un AMI definita nel Launch template dell'Autoscaling Group. Lo step successivo è la creazione di una funzione Lambda che funzioni da "helthcheck" per le macchine del Replica Set. Nel nostro caso si è scelto di creare una lambda in python con Pymongo installato come un Lambda Layer. La funzione Lambda si connette al Replica Set tramite *pymongo* e controlla lo status di tutti i nodi tramite il comando *replSetGetStatus* che ritorna un esploso dello stato del Replica Set:

```
import boto3
from pymongo import MongoClient
sqs = boto3.client('sqs')
mongoserver_uri = os.environ['MONGO_URI']
hostzone = os.environ['ZONE_ID']
mongo_connection = MongoClient(mongoserver_uri)
dns_aws = boto3.client('route53')
ec2 = boto3.resource('ec2')

rs_status = mongo_connection.admin.command('replSetGetStatus')
```

Tutti i nodi non funzionanti avranno **"health": 0.** Per evitare falsi positivi abbiamo configurato il codice in modo da ripetere il controllo varie volte in un intervallo di tempo di un minuto. Se entro questo intervallo il nodo non ritorna healty la Lambda scrive un messaggio su una coda SQS contenente il DNS name della macchina fallita e un timestamp.

Fatto questo il codice provvederà a de-registrare la macchina fallita dall'Autoscaling Group, se questa non è già stata terminata, utilizzando *boto3:*

```
autoscaling = boto3.client('autoscaling')

response = autoscaling.detach_instances(
    InstanceIds=[
        'string',
    ],
    AutoScalingGroupName='mongo-autoscaling',
    ShouldDecrementDesiredCapacity=False
)
```

Per evitare esecuzioni multiple la lambda è configurata per mandare il messaggio su sqs e deregistrare l'istanza solo se la coda SQS è vuota (ApproximateNumberOfMessages = 0).

Una volta preparata la lambda è il momento dell'ultimo passaggio: la **configurazione di CodeDeploy** che ha il compito di configurare le nuove istanze al momento dell'accensione da parte dell' autoscaling group. Iniziamo col creare una nuova applicazione nella dashboard di CodeDeploy e un nuovo deployment group mettendo come target l'autoscaling group appena creato. CodeDeploy esegue sulla macchina una serie operazioni di script configurabili dall'utente, definite in un file chiamato *appspec.yml* che deve essere salvato in un bucket S3 insieme agli script e ai file che devono essere trasferito e/o eseguiti sulla macchina.

La struttura del file è la seguente:

```
version: 0.0
os: linux
files:
  - source: /content
    destination: /home/ubuntu

hooks:
  ApplicationStop:
    - location: scripts/applicationstop.sh
      timeout: 300
      runas: root

  BeforeInstall:
    - location: scripts/beforeinstall.sh
      timeout: 300
      runas: root

  AfterInstall:
    - location: scripts/afterinstall.sh
      timeout: 300
      runas: root

  ApplicationStart:
    - location: scripts/applicationstart.sh
      timeout: 300
      runas: root
```

```
  ValidateService:
    - location: scripts/validateservice.sh
      timeout: 300
      runas: root
```

La struttura della cartella da caricare su s3 è la seguente e da utilizzare come input di codedeploy è la seguente:

```
content/
    mongod.conf
    mongo-keyfile
scripts/
    afterinstall.sh
    applicationstart.sh
    applicationstop.sh
    beforeinstall.sh
    validateservice.sh
```

Nel nostro caso la cartella content contiene il keyfile e il file di configurazione di Mongo mentre la cartella scripts contiene gli script bash che si occupano di configurare la nuova istanza. In particolare gli script devono leggere dalla coda SQS il messaggio contenente il DNS name della macchina fallita, configurare Mongo sulla macchina appena accesa, tramite la CLI di MongoDB riconfigurare il Replica Set per usare il nuovo nodo e quindi avviare il servizio di mongo.

Non appena un'istanza fallisce (oppure il processo di mongo diventa unresponsive) si innesca dunque la seguente catena di eventi:

- **La lambda di helthcheck rileva il nodo come unhealthy**, esegue il detach della macchina dall' autoscaling group e scrive un messaggio su una coda sqs contenente il dns name del nodo fallito.
- **L'autoscaling group reagisce all'assenza di una macchina** avviando una nuova EC2.
- **CodeDeploy esegue gli script di configurazione** salvati su S3 che configurano la nuova macchina in modo da farle prendere il posto di quella fallita.

A questo punto MongoDB inizia a replicare i dati sulla nuova macchina, tuttavia questa operazione per database grandi può richiedere diverse ore. Per evitare questo problema conviene **configurare MongoDB per salvare i dati su uno specifico EBS** e durante la creazione della nuova istanza smontare l'EBS in questione dall'istanza fallita e fare il mount sull'istanza appena creata.

Una volta concluse tutte questa configurazioni avremo finalmente il nostro **replica set in alta affidabilità e con automatic self-healing!**

Volete condividere con noi le vostre osservazioni, i vostri dubbi, i vostri spunti oppure volete una mano per creare una configurazione come questa? ... il nostro team non vede l'ora di approfondire il topic con voi!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it