Selçuk Usta

1.5K Followers About Follow

Redis’in High Availability Çözümü: Sentinel

Selçuk Usta Apr 14, 2018 · 5 min read

High Availability de nedir?

Matrix’in meşhur sahnesini hatırlamayan var mı? Avluda Neo ile (N) tane Ajan Smith’in sonsuza kadar süreceğini düşündüğümüz kavga sahnesinden bahsediyorum evet.

Aynı görev ve sorumlulukla donatılmış bir grup **(cluster)** Ajan Smith, aldığı onca ağır hasara — hatta bazılarının yok olmasına — rağmen görevini %99.9 kesintisiz bir şekilde yerine getirebiliyordu.

High Availability kavramını bu analoji ile aklımın bir köşesine çok uzun zaman önce yazdım. Dilim döndüğünce de bu şekilde ifade etmeyi doğru buluyorum. Tabii ki tartışmaya oldukça açık.

temel tanım ile bir "key-value store" ürünü. Yazılım dünyasında sıklıkla duyduğumuz — tabii ki duyduğumuz, biz böyle şeyler söylemeyiz — *Cache'ten geliyordur, N saniye sonra düzelir* ifadesinin faili olur kendisi.

Diğer bir çok database, caching sistemlerinde olduğu gibi Redis'te de cluster yapısı hali hazırda mevcut. Aslına bakarsanız Redis'in geliştiricisi (Github ismiyle *antirez*) ürününü, sağladığı in-memory sorted ve selection algoritmalarıyla (ZCARD, SCAN, vb. gibi) ön planda tutmayı tercih ediyor.

> *Yukarıdaki notu özellikle belirtmek istiyorum çünkü Redis'in **en iddialı** olduğu tarafın bu olmadığını bilmek gerek. Eğer amaç yalnızca bir anahtar ve anahtara karşılık gelen bir değeri N süreliğine tutmak ve kullanmak ise daha iyi alternatifler olduğunu düşünüyorum.*

Neden High Availability'e İhtiyaç Duyarız?

Yazının başındaki örneğe geri dönelim. Amacımız Neo'yu yenmek. Bunun için de elimizdeki en iyi ajan olan Smith'i, o zamana kadar mükemmel analiz ettiğimiz Neo'nun karşısına tek başına gönderiyoruz, biliyoruz ki bütün yeteneği ve dayanıklılığıyla sorunsuz bir şekilde görevini yerine getirebilir.

Ancak o da ne, bir sorun var! Neo hiç yorulmuyor, kaynaklarını oldukça verimli kullanıyor. Bizim kaynaklarımız ise tükenmek üzere. Bu istisnai duruma karşı bir senaryomuz yok, durumu tolere edemiyoruz **(fault-tolerance)** ve elimizdeki tek sistemi de kaybediyoruz. Artık tamamen savunmasız haldeyiz.

Peki, elimizde 3 tane aynı yetenekte ajanımız olsaydı?

Böylece kaynağı tükenen ajan kenara çekilip dinlense, bir başkası onun yerine geçip sistemi savunmaya devam etse ve bu yükü devretme **(failover)** işlemi sistemli bir şekilde ilerlese. Sonsuza kadar — teoride — ayakta kalabilir miydik?

> *Hata toleransı yüksek gruplar (fault-tolerance cluster), yük devretme otomasyonları (failover) ile %99.9 ayakta kalma (up-time) sözü verirler.*

Redis'te söz veren ürünlerden bir tanesi.

Redis'in Çözümü Nedir?

olarak yürütmesi anlamına gelmez.

Bu noktada Redis, 2.6 sürümüyle ortaya koyduğu ve **Sentinel** adını verdiği uygulama ile failover'ın otomatik olarak yönetilmesini sağlayabiliyor.

> ***NOT:*** *2.6 ile gelen Sentinel 1 sürümünün kullanılması önerilmiyor. 2.8 ve sonrasında gelen Sentinel 2 sürümü tercih edilmelidir.*

Redis Cluster yapısı en yalın haliyle 1 adet master, N adet slave uygulamadan oluşur. Tüm yazma eylemleri master uygulama üzerinden yürütülür, slave'ler read-only yapılardır. Master üzerindeki verileri replika yöntemi ile üzerlerinde taşırlar.

Sentinel ise, cluster üzerindeki uygulamaları izleme, raporlama ve olası bir hata/istisna durumunda cluster'ın sağlıklı bir şekilde yoluna devam edebilmesinden sorumlu apayrı bir yapıdır.

Sentinel ne iş yapar?

Master uygulamada bir sorun olduğunu varsayarsak bu şu anlama gelir: Artık Redis'e yeni bir veri yazılamayacak!

Tam bu esnada sistem kritik bir karara ihtiyaç duyar: Yeni master kim olacak?

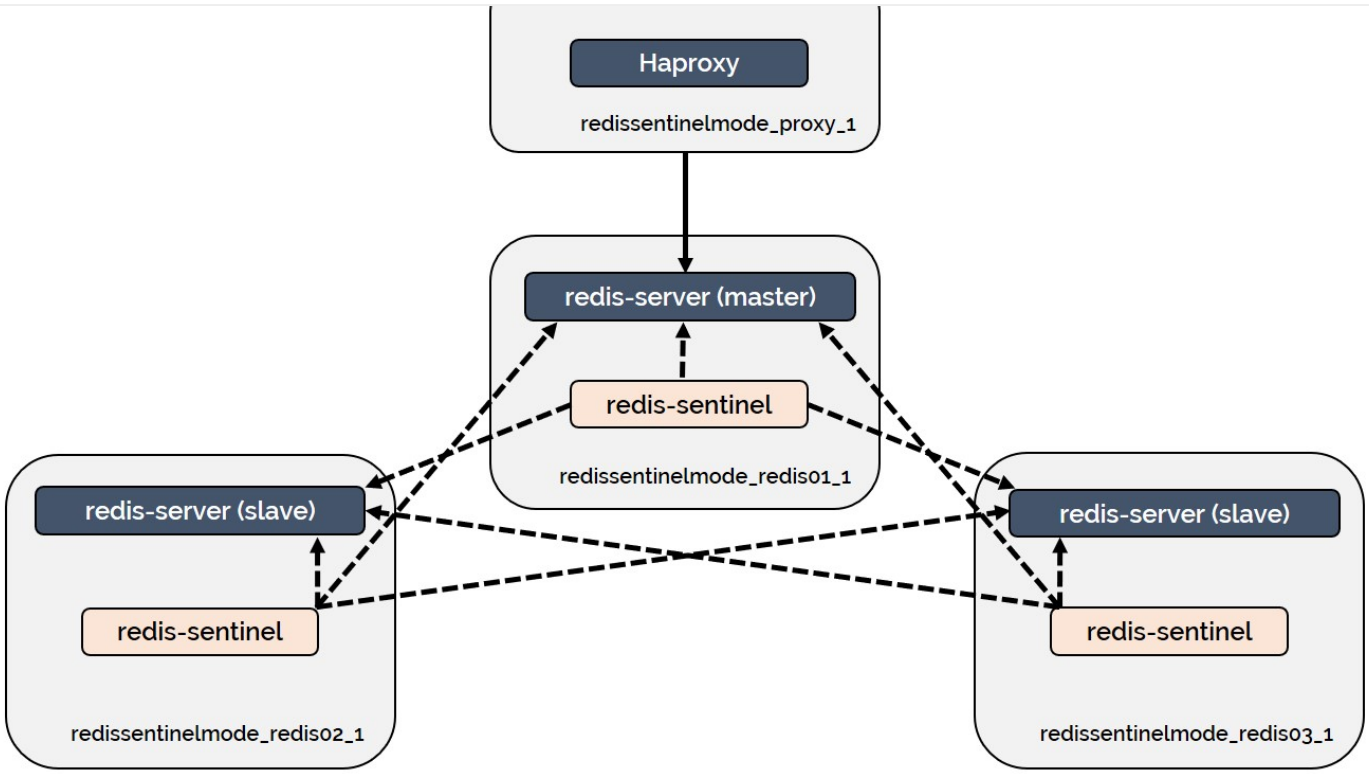
Sentinel tam bu noktada devreye girer, master'da bir sorun olduğunu farkeder, gerekli konfigürasyonlar yapılmış ise diğer sentinel'ler ile haberleşir ve yeni master'ın atanmasını sağlar. Ya da rol bağımsız olarak, sorunlu uygulamayı tespit eder, cluster dışında tutar; bir süre sonra sağlıklı bir duruma geldiğini tespit ettiğinde tekrar içeri alır ve rol ataması gerçekleştirir.

Sentinel ile ilgili büyük bir dokümantasyonu bu adreste bulabilirsiniz. Detaylıca okumakta fayda var diye düşünüyorum.

Buraya kadar teorik bazı açıklamalarla kafalarda soru işareti oluşturmaya çalıştım, eğer yeterli soru işaretine sahipsek, gelin uygulayarak bu sistemi gözlemleyelim.

Deneme — Yanılma

Önce topolojiyi görmekte fayda var:



Örnek uygulamada 3 node'umuz var. Bunlardan ikisi slave, biri master olarak konumlanmış durumda. Node'lar içerisinde hem redis-server hem de redis-sentinel çalışacak.

> *Yine tam bu noktada bir not düşmekte fayda var. Bu mimari, sistem kaynaklarınıza ve ihtiyaçlarınıza göre değişecektir. Yukarıdaki topolojide gözlemciler ve asıl uygulamalar aynı makineler üzerinde. Haliyle fiziksel bir sorun meydana geldiğinde her ikisi de kullanılamaz duruma gelecektir. Dilerseniz Sentinel'leri ayrı makinelerde de konumlandırabilirsiniz.*

Sentinel sayısı için genellikle önerilen ve "best-practice" olarak kabul gören minimum sayı 3 ve 3'ten büyük tek sayılar. Böylece oy çokluğu ile kararsızlık senaryosu devre dışı bırakılabilir.

Bu 3'lünün önünde ise, cluster'a gelen istekleri karşılamak ve yönlendirmek üzere bir HAProxy konumlandırıyoruz. Bunun sebebini ise şöyle açıklayabiliriz;

- Node'ları health-check yapacak ve olası bir sorun durumunda bilgilendirecek bir yapı olması

İkinci maddeyi açmakta fayda var. Örneğin mevcuttaki master'ın IP'si 192.168.1.25 olsun. Slave'lerin ise 24 ve 26 ile bittiğini farz edelim. Uygulama geliştirirken de bağlantı cümlesi olarak `192.168.1.25:6379` adresini kullandık. Ancak oldu ki master değişti. Bu durumda artık uygulama üzerinden Redis'e yazamayız. Herhangi sorun olmasa dahi, sadece 25'e bir bağlantı kurduğumuz için diğer sunuculardan okuma gerçekleştiremeyiz. HAProxy ile `haproxy.domain.name` adresini kullanarak gelen istekleri Redis cluster'ına yönlendirme yapma şansı elde ediyoruz.

Örnek uygulamanın tamamına yazının sonundaki Github referansından ulaşabilirsiniz. Ben "can alıcı" olduğunu düşündüğüm kısımlardan bahsetmeye çalışacağım.

`./redis/redis-entrypoint.sh`

```
1   #!/bin/sh
2   tee /etc/redis/sentinel.conf <<EOF
3   port 26379
4   daemonize yes
5   logfile "/var/log/sentinel.log"
6   pidfile "var/run/sentinel.pid"
7   dir "/var/lib/redis/sentinel"
8
9   sentinel monitor docker-cluster $MASTER_HOST 6379 $SENTINEL_QUORUM
10  sentinel down-after-milliseconds docker-cluster $SENTINEL_DOWN_AFTER
11  sentinel parallel-syncs docker-cluster 1
12  sentinel failover-timeout docker-cluster $SENTINEL_FAILOVER
13  EOF
14
15  redis-sentinel /etc/redis/sentinel.conf
16
17  if [ "$IS_SLAVE" == true ]; then
18          redis-server --slaveof $MASTER_HOST 6379
19  else
20      redis-server
21  fi
```

9. satır en önemli kısım olarak ifade dilebilir. `docker-cluster` isimli bir cluster üzerindeki `redis01` (docker-compose.yml dosyasından environment variable olarak

quorum ifadesi ise, otomatik failover senaryosunun başlayabilmesi için kaç Sentinel işleminin master'ı ulaşılamaz olarak kabul etmesi gerektiğini temsil ediyor. Burada dikkat edilmesi gereken nokta şu:

> *Quorum, olası bir hatanın tespiti için kullanılır. Failover gerçekleşmesi için herhangi bir Sentinel lider olarak seçilmeli ve failover'ı başlatma yetkisi kendisine verilmelidir. Bu da oy çokluğu ile gerçekleştirilebilir.*

Bizim senaryomuzda `2` ifadesi ile, failover'ın başlayabilmesi için 2 Sentinel'in master'ı ulaşılamaz kabul etmesi gerektiğini söylüyoruz. **Buna rağmen Sentinel'ler birbiri ile haberleşemezse failover senaryosu başlamaz.**

10. satırdaki `down-after-milliseconds` ile de kaç milisaniye sonunda bir Sentinel'in ilgili node'u ulaşılamaz olarak kabul etmesi gerektiğini ifade ediyoruz.

15. satırda ise her bir container'ın içerisinde mutlaka, yukarıdaki konfigürasyonlarla ayarlanmış bir redis-sentinel işleminin başlaması gerektiğini ifade ediyoruz.

18. satırda eğer node IS_SLAVE environment variable'ı taşıyor ve değeri `true` ise o node'daki redis kopyasını slave mode olarak çalıştırmamıza olanak sağlıyor.

Sonuç

`docker-compose up -d` komutu ile uygulamayı çalıştırıp, stack ayağa kalktıktan sonra `docker exec -it redis01 sh` komutunu çalıştırıp container içerisine girmenizi ve `redis-cli info replication` komutunu çalıştırmanızı tavsiye ediyorum. Böylece master'a bağlı kaç adet slave'in olduğunu ve istatistikleri öğrenebilirsiniz. Ya da *redis01* yerine *redis02* veya *redis03* yazarak aynı istatistikleri slave ağzından dinleyebilirsiniz.

Bir süre bekleyip `docker-compose pause redis01` komutu çalıştırdığınız zaman ise redis02 ya da redis03'e gidip istatistiklere baktığınızda içlerinden birinin master rolüne yükseldiğini göreceksiniz:

Düzenleme: Yukarıdaki senaryoyu görselleştirdiğim bir animasyon bu alanda bulunuyordu ancak host edilen site üzerinden silinmiş. Animasyonu sabit diskimde de tutmamışım, dolayısıyla bu alan boşa çıkmış durumda.

selcukusta/redis-sentinel-with-haproxy

redis-sentinel-with-haproxy — Build fault-tolerance Redis cluster with Sentinel on Docker

github.com

Medium Turkish Türkçe Redis Software Development High Availability