N KOLAY BILGI TEKNOLOJILERI GRUBU

Java Uygulamaları için Kubernetes Deneyimleri

Kubernetes'te Java uygulamaları için sınırlar koymalı mıyız, koymamalı mıyız?

Cevap açık gibi görüse de – çünkü Kubernetes YAML bildirimlerimizi doğrulayan birçok araç vardır ve CPU veya bellek limitleri belirlemesek de kesinlikle bir uyarı yazdırırlar - ancak, toplulukta bununla ilgili bazı "sıcak tartışmalar" var.

CPU Limiti :

Örneğin herhangi bir CPU sınırı belirlemenizi önermeyen bir yazı : https://home.robusta.dev/blog/stop-using-cpu-limits

Örneğin bir CPU sınırı belirlemenizi şiddetle öneren bir diğer yazı : https://dnastacio.medium.com/why-you-should-keep-using-cpu-limits-on-kubernetes-60c4e50dfc61

CPU sınırlarını göz önünde bulunduralım veya bulundurmayalım, benzer bir tartışma bellek sınırları için de vardır : https://home.robusta.dev/blog/kubernetes-memory-limit

Özellikle Java uygulamaları bağlamında Kubernetes ortamında benim kişisel tecrübem önerim her zaman hafıza limitinin ayarlanması yönündedir . Ayrıca CPU limit talebi ile de görüşüm de gelen istek sayısı ile aynı yönde arttırılması yönündedir.

Java uygulamaları bağlamında, belleği -Xmx, -XX:MaxMetaspaceSize or -XX:ReservedCodeCacheSize JVM parametreleriyle sınırlayabilmemiz önemlidir

Kubernetes açısından bakıldığında ise , pod lar talep ettiği kaynakları alır. CPU limitin bellek ile bir ilgisi yoktur...

Deneyimlerim beni ilk tavsiyeye yönlendiriyor – CPU limitlerinizi çok düşük tutmayın. Ama bir CPU sınırı ayarlasanız bile, kesinlikle aynı zamanda uygulamanızı da etkilememesi gerektiğini göz önüne alın.

Örneğin, muhtemelen bildiğiniz gibi, Java uygulamanız normal çalışmada çok fazla CPU tüketmese bile, hızlı başlamak için çok fazla CPU gerektirir. Örneğin MongoDB'yi Kubernetes'e bağlayan basit bir Spring Boot **uygulaması için, sınırsız ve hatta 0,5 fazla Core ayırma arasındaki fark çok fazladır.**

Normalde 10 saniyenin altında başlayan uygulama CPU limit 'i sadece 500 millicore kadar fazla arttırıldığında bile 30 sn de başlayabilir..

```
2023-02-09 15:17:50 - INFO Exposing 14 endpoint(s) beneath base path '/actuator'
2023-02-09 15:17:51 - INFO Tomcat started on port(s): 8081 (http) with context path ''
2023-02-09 15:17:51 - INFO Started SpringBootOnKubernetesApp in 7.29 seconds (process running for 8.477)
```

```
2023-02-09 15:19:26 - INFO Exposing 14 endpoint(s) beneath base path '/actuator'
2023-02-09 15:19:26 - INFO Tomcat started on port(s): 8081 (http) with context path ''
2023-02-09 15:19:26 - INFO Started SpringBootOnKubernetesApp in 30.784 seconds (process running for 33.549)
```

Bellek Limiti :

Şimdi sadece hafıza limitine odaklanalım. Kubernet'lerde bir Java uygulaması çalıştırırsanız, maksimum kullanımı iki düzeyde sınırlandırırsınız: Conatiner ve JVM. Ancak, JVM için herhangi bir ayar belirtmezseniz de varsayılan değerler vardır.

Uygulama için `-Xmx`.JVM heap parametrelerini ayarlamamanız durumunda maksimum yığın boyutunu da kullanılabilir; ki bu da RAM'in yaklaşık toplamın %25'i olarak ayarlanması demektir. Bu değer, container içinde görünen toplam belleğe göre ayarlanır. Container düzeyinde bir sınır belirlemediğinizde ise JVM, düğümün tüm belleğini görecektir.

Uygulamayı Kubernetes üzerinde çalıştırmadan önce, en azından beklenen ortalama yükte ne kadar bellek tükettiğini ölçmelisiniz.

Neyse ki, container içinde çalışan Java uygulamaları için bellek yapılandırmasını optimize edebilecek araçlar da vardır.

Örneğin, Paketo Buildpacks yerleşik bir bellek hesaplayıcıyla birlikte gelir. `-Xmx` JVM değeri ile aşağıdaki gibi hesaplar :

`Heap = Total Container Memory - Non-Heap - Headroom`.

Öte yandan, yığın olmayan değer (heap dışı değer) aşağıdaki formül kullanılarak hesaplanır:

`Non-Heap = Direct Memory + Metaspace + Reserved Code Cache + (Thread Stack * Thread Count)`.

Paketo Buildpacks, şu anda Spring Boot uygulamaları oluşturmak için varsayılan bir seçenektir

(komutla `mvn spring-boot:build-image`).

Örnek uygulamamız için deneyelim. Bellek sınırını 512M olarak ayarlayacağımızı varsayarsak, `-Xmx`130M düzeyinde hesaplayacaktır.

```
[sample-spring-boot-on-kubernetes] Calculated JVM Memory Configuration: -XX:MaxDirectMemorySize=10M
(-Xmx130208K) -XX:MaxMetaspaceSize=86879K -XX:ReservedCodeCacheSize=240M -Xss1M (Total Memory: 512M)
 Thread Count: 50, Loaded Class Count: 12925, Headroom: 0%)
```

Uygulamam için uygun mu? Uygulamamın yoğun trafik altında nasıl performans gösterdiğini doğrulamak için en azından bazı yük testleri yapmalıyız. Ancak bir kez daha – limitleri çok düşük ayarlamayın.

Örneğin, 1024M limiti ile `-Xmx`650M'ye eşittir.

```
[sample-spring-boot-on-kubernetes] Calculated JVM Memory Configuration: -XX:MaxDirectMemorySize=10M
(-Xmx654496K) -XX:MaxMetaspaceSize=86879K -XX:ReservedCodeCacheSize=240M -Xss1M (Total Memory: 1G) Thr
ead Count: 50, Loaded Class Count: 12925, Headroom: 0%)
```

Gördüğünüz gibi bellek kullanımını JVM parametreleriyle hallediyoruz. Bu “Out of Memory” öldürmelerini engeller.

Bu nedenle, isteklerin limitle aynı seviyede ayarlanmasının pek bir anlamı yoktur. Normal kullanımdan biraz daha yükseğe ayarlamanızı tavsiye ederim – %20 daha fazla diyelim.

Canlılık (Liveness) ve Hazırlık(Readiness) Evreleri :

Kubernetes ortamında canlılık ve hazır olma arasındaki farkı anlamak çok önemlidir.

Her ikisi de dikkatli bir şekilde uygulanmazsa, örneğin gereksiz yeniden başlatmalara neden olarak bir hizmetin genel işleyişini bozabilirler.

Üçüncü evre olan başlangıç (**StartUp**) evresi, Kubernetes'de görece yeni bir özelliktir. Canlılık veya hazır olma evrelerinde initialDelaySeconds ayarlamaktan kaçınmamıza olanak tanır. Yani bu ayarı artık kullanmayız ve bu sayede, uygulama başlatma işleminiz hızlanır.

N KOLAY BILGI TEKNOLOJILERI GRUBU

Özellikle uygulama başlatma işleminiz eğer çok zaman alıyorsa bu söylediğim özellik çok yararlıdır…

https://dnastacio.medium.com/the-art-and-science-of-probing-a-kubernetes-container-db1f16539080

Bir uygulama herhangi bir nedenle kullanılamıyorsa, kapsayıcıyı yeniden başlatmak bazen mantıklı olabilir.

Öte yandan, bir konteynerin gelen trafiği idare edip edemeyeceğine karar vermek için bir hazırlık araştırması(Readiness) kullanılır.

Bir pod ‘un hazır olmadığı algılanırsa, yük dengelemeden çıkarılır.

Web uygulamaları için en tipik canlılık veya hazır olma araştırması, bir HTTP uç noktası aracılığıyla gerçekleştirilir.

Hazırlık araştırmasının(Readiness) başarısız olması, pod un yeniden başlatılmasına neden olmaz.

Container’ın yeniden başlatılıp başlatılmayacağına karar vermek için bir canlılık araştırması(Liveness) kullanılır.

Bu nedenle Canlılık araştırmasının müteakip başarısızlıkları pod un yeniden başlatılmasına neden olduğundan, bence uygulama; entegrasyonlarınızın kullanılabilirliğini kontrol etmemelidir. Bunu izleme ile görüp müdahale etmeliyiz; ama tabi ki yoruma da açık bir konu.. Hayır, kontrol edilmelidir diyen de olacaktır.