Spring Cloud

Milad Barzideh

فهرست مطالب

2	یکروسرویسها، ابر و چیزهای دیگر	۱ م
7	 ۱. میکروسرویس چیست؟	١
1	 . ۲ محاسبات ابری چیست؟	١
1	 .۳ چرا ابر؟	١
1	ننترل پیکربندیها	5 T
f	 ۱.۱ معماری مدیریت پیکربندی	٢
٠ (۲.۲ روشهای پیادهسازی	٢
٠ ١	 . Spring Cloud configuration server ییادهسازی ۳.'	٢

فهرست مطالب

فصل ۱

میکروسرویسها، ابر و چیزهای دیگر

میکروسرویس یا ریزخدمت کی از اصطلاحاتیست که این روزها در دنیای نرمافزار رایج شده است. میکروسرویسها، سرویسهایی توزیعشده و مجزا هستند که هر کدام از آنها وظیفهی کوچک و مشخصی را انجام میدهند. در این نوشتار، پس از معرفی میکرسرویسها و ذکر مشخصههای آن، با استفاده از فریمور کهای Spring Boot و Spring Cloud به صورت عملی، معماری میکروسرویسها را بررسی خواهیم کرد.

1.۱ میکروسرویس چیست؟

قبل از مطرح شدن میکروسرویسها بیشتر برنامههای تحتوب با استفاده از معماری یکپارچه ^۲ ساخته میشدند. در معماری یکپارچه، هر برنامه به عنوان یک محصول نرمافزاری قابل اجرا در نظر گرفته میشود و رابط کابری، منطق برنامه و دسترسی به پایگاه داده، همه در یک برنامهی کاربردی قرار گرفته میشود.

هر برنامه به عنوان یک واحد کاری در نظر گرفته می شود که معمولا چندین تیم توسعه روی بخشهای مختلف آن کار می کنند. یکی از مشکلاتی که در این حالت پیش می آید این است که با افزایش اندازه و پیچید گی برنامه، هزینه های ارتباطی و هماهنگی تیمها، بیشتر و سخت تر می شود. برای مثال با هر تغییری که تیمها ایجاد می کنند، کل برنامه دوباره باید تست و مستقر آشود.

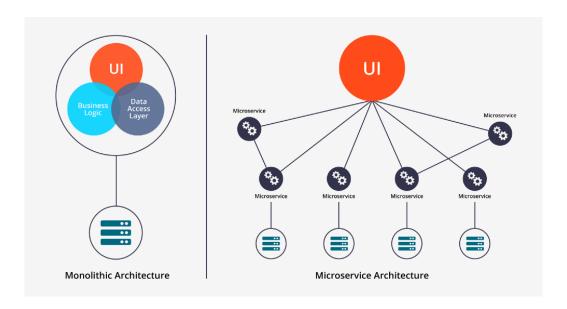
مفهوم میکروسرویس تقریبا از سال ۲۰۱۴ و در پاسخ به چالشهایی که برای مقیاسپذیر کردن برنامههای یکپارچه وجود داشت، وارد جامعهی نرمافزار شد. همانطور که گفته شد یک میکروسرویس، کوچک، مجزا و توزیعشده است. بنابراین این امکان را فراهم میکند که برنامههای بزرگ را به بخشهای کوچکتری تقسیم کرد تا مدیریت و توسعه ی آنها آسان تر داده شوند. مهم ترین نکتهای که باید به آن توجه داشته باشید، تجزیه و جداسازی عملکردهای برنامه است به صورتی که کاملا مستقل از همدیگر باشند.

همانطور که شکل ۱.۱ نشان میدهد هر تیم میتواند کدها و زیرساخت مربوط به سرویس خود را داشته باشد و آنها را به صورت مستقل نسبت به سرویسهای دیگر توسعه و ارزیابی کند.

[\]Microservice

⁷Monolithic

[&]quot;Redeployed



شکل ۱.۱: معماری یکپارچه در مقابل معماری میکروسرویس

معماری میکروسرویس ویژگیهای زیر را دارد:

- * منطق برنامه به قسمتهای کوچکتر شکسته می شود که هر قسمت مسئلهی مشخص و تعریف شده ای را حل می کند.
- * هر قسمت محدودهی مسئولیت مشخصی دارد و مستقل از سایر بخشها توسعه و استقرار می یابد. همچنین میکروسرویسها باید قابلیت استفادهی مجدد در برنامههای دیگر را داشته باشند.
- * میکروسرویسها با استفاده از قواعد مشخص و بکارگیری پروتکلهای ارتباطی سبکوزن مانند HTTP و JSON دادهها را مبادله می کنند.
- * پیادهسازی فنی هر میکروسرویس بیاهمیت است؛ زیرا برنامهها با پروتکلهای مستقل از تکنولوژی ٔ با هم ارتباط برقرار میکنند. این یعنی برنامهای که با معماری میکروسرویس ساخته شده می تواند از تکنولوژیها و زبانهای مختلفی استفاده کند.
- * میکروسرویسها برای سازمانها این امکان را فراهم میکنند که تیمهای توسعه ی کوچک با وظایف کاری مشخص داشته باشند.

^{*}Technology-neutral protocol

۲.۱ محاسبات ابری چیست؟

۳.۱ چرا ابر؟

به عنوان یک توسعه دهنده ی میکروسرویس، دیر یا زود باید تصمیم بگیرید سرویس تان را در یکی از جاهای زیر مستقر کنید:

- * سرور فیزیکی: سازمانهای کمی این کار را انجام میدهند؛ زیرا سرورهای فیزیکی محدودیت ایجاد میکنند. برای مثال شما نمی توانید ظرفیت سرور فیزیکی را سریعا افزایش دهید یا میکروسرویس را به دلیل هزینههایی زیادی که در پی دارد روی چندین سرور فیزیکی مستقر کنید.
- * ماشین مجازی: یکی از مهمترین فواید میکروسرویسها، توانایی آنها در افزودن یا کاهش نمونههای یک سرویس است (مقیاسپذیری). یک میکروسرویس را میتوان در یک ماشین مجازی قرار داد و به آسانی چندین نمونه از آن را روی زیرساختهای موجود مستقر کرد.
- * ظرف مجازی ^۵: به جای استقرار میکروسرویس روی یک ماشین مجازی کامل، بسیاری از توسعه دهندگان سرویسهای خود را به صورت ظرفهای داکر ^۶ در محیط ابری مستقر می کنند.

تفاوت داکر با ماشین مجازی

مزیت مهم میکروسرویسهای تحت ابر انعطاف پذیری بالای آنهاست. ارائه دهندگان محیط ابری اجازه می دهند در کمتر از چند دقیقه یک ماشین مجازی یا یک ظرف جدید را اضافه یا حذف کنید. همچنین برنامهها مقاومتر هستند، برای مثال اگر یکی از میکروسرویسها از کار بیفتد می توان از نمونههای دیگر آن میکروسرویس استفاده کرد و برنامه را بالا نگه داشت.

^aVirtual container

⁹Docker

فصل ۲

كنترل ييكربنديها

به عنوان یک توسعه دهنده می دانید که hard-code کردن مقادیر در کدهای برنامه چندان منطقی نیست، به همین دلیل معمولا توسعه دهندگان از یک فایل ثابت استفاده می کنند و همه اطلاعات پیکربندی برنامه را در آن قرار می دهند. با این حال قرار دادن این اطلاعات در کدهای برنامه مشکل ساز است، زیرا با هر تغییر در پیکربندی، کل برنامه دوباره باید کامپایل شود. برای جلوگیری از این کار، توسعه دهندگان اطلاعات مربوط به پیکربندی را از کدهای برنامه جدا می کنند.

این مشکل در برنامههای تحت ابر بیشتر خود را نشان می دهد، از آنجا که ممکن است این برنامهها از صدها میکروسرویس با چندین نمونه ی در حال اجرا تشکیل شده باشند، مدیریت پیکربندی برنامه بسیار مشکل است. برای همین توسعه ی میکروسرویسهای تحت ابر روی موارد زیر تاکید دارد:

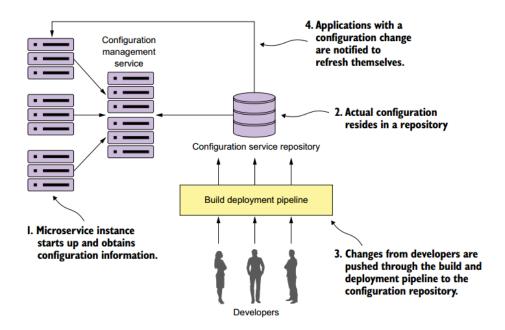
- ۱. جداسازی پیکربندی برنامه از کدهای اصلی برنامه
- ۲. تزریق اطلاعات مربوط به پیکربندی برنامه در زمان راهاندازی سرور با استفاده از یک مخرن مرکزی به میکروسرویسهای برنامه.

۱.۲ معماری مدیریت پیکربندی

همانطور که گفته شد لودشدن پیکربندی باید در راهاندازی ۱ میکروسرویسها انجام شود. شکل ۱.۲ نقش مهم سرویس پیکربندی را برای راهاندازی میکروسرویسها نشان میدهد:

- ۱. قبل از اینکه یک نمونه از میکروسرویس بالا بیاید باید یک endpoint را برای خواندن اطلاعات پیکربندی مربوط به خود صدا بزند. اطلاعات لازم برای فراخوانی endpoint هنگام راهاندازی سرویس به آن پاس داده می شود.
- مخزن پیکربندی می تواند با استفاده از گیت، پایگاه داده های رابطه ای یا کلید-مقدار پیاده سازی شود.
 - ۳. مدیریت دادههای پیکربندی باید مستقل از چگونگی استقرار برنامه باشد.

[\]Bootstrapping



شکل ۱.۲: معماری مفهومی مدیریت پیکربندیها

۴. وقتی اطلاعات پیکربندی تغییر یابد؛ سرویسهایی که از آن اطلاعات استفاده میکنند باید از تغییر آگاه شوند و دادههای خود را بروز کنند.

۲.۲ روشهای پیادهسازی

برای پیادهسازی مدیریت پیکربندی پروژههای متنباز زیادی وجود دارد. شکل ۲.۲ برخی از آنها را نشان میدهد.

در اینجا از Spring Cloud configuration server استفاده می شود. این روش نسبت به روشهای دیگر ساده تر است، به خوبی با Spring Boot ادغام می شود و از گزینه های بیشتری برای ذخیره داده های پیکربندی پشتیبانی می کند.

۳.۲ پیادهسازی Spring Cloud configuration server

configuration server یک برنامهی REST-based است که روی Spring Boot ساخته می شود. کدهای ۲.۱ فایل pom.xml را برای ساخت این سرویس نشان می دهند.

Listing 2.1: pom.xml

```
1
2 <!-- remove boilerplate codes -->
3
4 <parent>
```

Project Name	Description	Characteristics
Etcd	Open source project written in Go. Used for service discovery and key-value management. Uses the raft (https://raft.github.io/) protocol for its distributed computing model.	Very fast and scalable Distributable Command-line driven Easy to use and setup
Eureka	Written by Netflix. Extremely battle-tested. Used for both service discovery and keyvalue management.	Distribute key-value store. Flexible; takes effort to set up Offers dynamic client refresh out of the box
Consul	Written by Hashicorp. Similar to Etcd and Eureka in features, but uses a different algorithm for its distributed computing model (SWIM protocol; https://www.cs.cornell.edu/~asdas/research/dsn02-swim.pdf).	Fast Offers native service discovery with the option to integrate directly with DNS Doesn't offer client dynamic refresh right out of the box
ZooKeeper	An Apache project that offers distributed locking capabilities. Often used as a configuration management solution for accessing key-value data.	Oldest, most battle-tested of the solutions The most complex to use Can be used for configuration manage- ment, but should be considered only if you're already using ZooKeeper in other pieces of your architecture
Spring Cloud configuration server	An open source project that offers a general configuration management solution with different back ends. It can integrate with Git, Eureka, and Consul as a back end.	Non-distributed key/value store Offers tight integration for Spring and non-Spring services Can use multiple back ends for storying configuration data including a shared filesystem, Eureka, Consul, and Git

شکل ۲.۲: روشهای پیادهسازی مدیریت پیکربندی

```
<groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.4.RELEASE
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
10
  cproperties>
11
    project.build.sourceEncoding>UTF-8
12
       sourceEncoding>
    project.reporting.outputEncoding>UTF-8
13
       reporting.outputEncoding>
    <java.version>1.8</java.version>
    <spring-cloud.version>Finchley.RELEASE</spring-cloud.</pre>
       version>
  </properties>
```

كنترل پيكربندىها

```
<dependencies>
    <dependency>
19
       <groupId>org.springframework.cloud</groupId>
20
       <artifactId>spring-cloud-config-server</artifactId>
    </dependency>
22
23
    <!--refresh config-->
    <dependency>
25
       <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-config-monitor</artifactId>
27
    </dependency>
28
    <dependency>
29
      <groupId>org.springframework.cloud</groupId>
30
       <artifactId>spring-cloud-starter-stream-rabbit
31
          artifactId>
    </dependency>
  </dependencies>
  <dependencyManagement>
    <dependencies>
36
      <dependency>
         <groupId>org.springframework.cloud</groupId>
         <artifactId>spring-cloud-dependencies</artifactId>
         <version>${spring-cloud.version}
40
         <type>pom</type>
41
         <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
46
  <build>
47
    <plugins>
48
      <plugin>
         <groupId>org.springframework.boot</groupId>
50
         <artifactId>spring-boot-maven-plugin</artifactId>
51
      </plugin>
52
    </plugins>
53
  </build>
55
 </project>
```

در فایل Maven بالا، ابتدا نسخهای از Spring Boot که در میکروسرویسها از آن استفاده Spring بالا، ابتدا نسخهای از کطوط 8-1) مشخص کردن نسخه Spring می شود، مشخص شده است. قسمت مهم بعدی (خطوط 8-1) مشخص کردن نسخه Spring Cloud است. Spring Cloud شامل مجموعهای از پروژههای مستقل است که هر کدام نسخههای

مربوط به خود را دارند. با استفاده از این تعریف، تضمین میشود که از زیرپروژههای سازگار با هم در Spring Cloud استفاده شود. قسمت بعدی مشخص کردن وابستگیهای ...