Spring Cloud

Milad Barzideh

MB

فهرست مطالب

فصل ۱

میکروسرویس‌ها، ابر و چیزهای دیگر

میکروسرویس یا ریزخدمت[۱] یکی 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 فریمورک‌های Spring Boot و Spring Cloud به صورت عملی، معماری میکروسرویس‌ها را بررسی خواهیم کرد.

۱.۱ 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 کاربری، منطق برنامه و دسترسی به پایگاه داده همه در یک برنامه‌ی کاربردی قرار داده می‌شود.

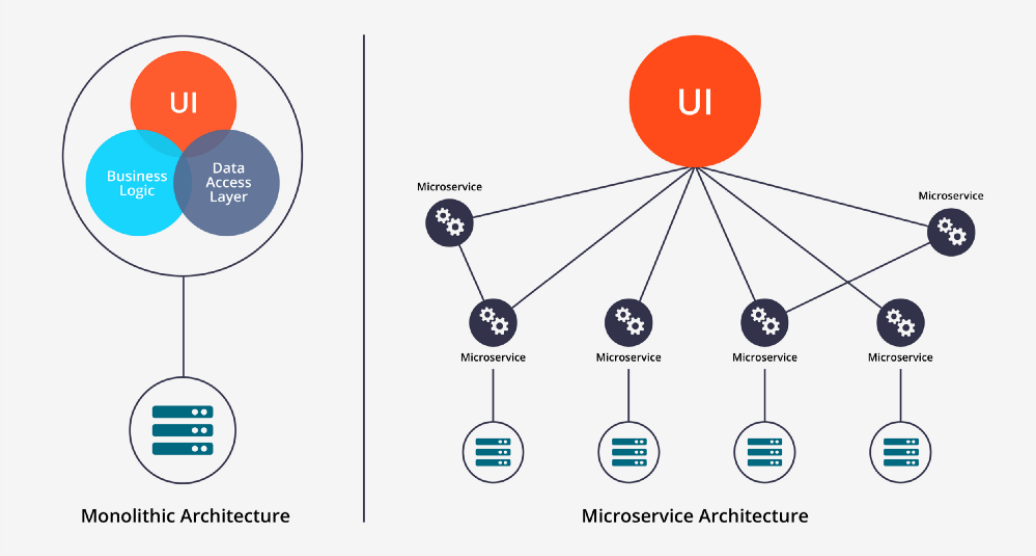
هر برنامه به عنوان یک واحد کاری در نظر گرفته می‌شود که معمولا چندین تیم توسعه روی بخش‌های مختلف آن کار می‌کنند. یکی از مشکلاتی که در این حالت پیش می‌آید این است که با افزایش اندازه و پیچیدگی برنامه، هزینه‌های ارتباطی و هماهنگی تیم‌ها، بیشتر و سخت‌تر می‌شود. برای مثال با هر تغییری که تیم‌ها ایجاد می‌کنند، کل برنامه باید دوباره تست و مستقر[۳] شود.

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 توسعه‌ی آن‌ها آسان‌تر شود. مهم‌ترین نکته‌ای که باید به آن توجه داشت 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.

---

[۱] Microservice
[۲] Monolithic
[۳] Redeployed



شکل ۱.۱: معماری یکپارچه در مقابل معماری میکروسرویس

معماری میکروسرویس ویژگی‌های زیر را دارد:

* منطق برنامه به قسمت‌های کوچک‌تر شکسته می‌شود و هر قسمت مسئله‌ی مشخص و تعریف شده‌ای را حل می‌کند.
* هر قسمت مسئولیت مشخصی دارد و مستقل از سایر بخش‌ها توسعه و استقرار می‌یابد. همچنین میکروسرویس‌ها باید قابلیت استفاده‌ی مجدد در برنامه‌های دیگر را داشته باشند.
* میکروسرویس‌ها با استفاده از قواعد مشخص و بکارگیری پروتکل‌های ارتباطی سبک‌وزن[۴] مانند HTTP و JSON داده‌ها را مبادله می‌کنند.
* پیاده‌سازی فنی هر میکروسرویس بی‌اهمیت است؛ زیرا برنامه‌ها با پروتکل‌های مستقل از تکنولوژی[۵] با هم ارتباط برقرار می‌کنند. این یعنی برنامه‌ای که با معماری میکروسرویس ساخته شده می‌تواند از تکنولوژی‌ها و زبان‌های مختلفی استفاده کند.
* میکروسرویس‌ها برای سازمان‌ها این امکان را فراهم می‌کنند که تیم‌های توسعه‌ی کوچک با وظایف کاری مشخص داشته باشند.

۲.۱ محاسبات ابری چیست؟

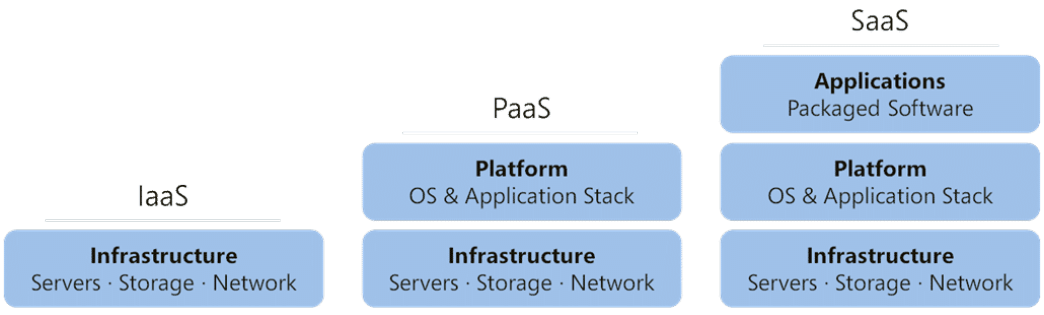
دنیای محاسبات به‌سرعت به سمت توسعه‌ی نرم‌افزارهایی پیش می‌رود که به‌جای اجرا بر روی کامپیوترهای منفرد، به عنوان یک سرویس در دسترس میلیون‌ها مصرف‌کننده قرار داده می‌شوند. از

[۴] Lightweight
[۵] Technology-neutral protocol

این نظر، محاسبات ابری از دید کاربران نهایی ساختاری شبیه به یک توده‌ی ابر دارد که به واسطه‌ی آن می‌توانند به برنامه‌های کاربردی از هر جایی از دنیا دسترسی داشته باشند.
محاسبات ابری یک الگوی محاسباتی است که در آن تعداد بسیار زیادی از سیستم‌ها به صورت شبکه‌های خصوصی و یا عمومی به یکدیگر متصل شده‌اند تا زیرساخت پویا و مقیاس‌پذیری را برای برنامه‌های کاربردی، ذخیره داده‌ها و فایل‌ها فراهم آورند. با ظهور این تکنولوژی، هزینه‌ی محاسبات، میزبانی برنامه‌های کاربردی، ذخیره‌سازی محتوا و تحویل سرویس‌ها به طور قابل توجهی کاهش یافته است. ایده‌ی محاسبات ابری در اصل بر مبنای "استفاده مجدد از قابلیت‌های فناوری" است.

۳.۱ معماری لایه‌ای محاسبات ابری

رایانش ابری می‌تواند برای تحویل سرویس‌های موجود در لایه‌های مختلف، از سخت‌افزار گرفته تا برنامه‌ی کاربردی مورد استفاده قرار گیرد. در عمل ارائه‌دهندگان محاسبات ابری سرویس‌های مختلف آن را در سه گروه دسته‌بندی کرده‌اند: نرم‌افزار به عنوان سرویس[۶]، پلتفرم به عنوان سرویس[۷] و زیرساخت به عنوان سرویس[۸]. این گروه‌ها با همدیگر لایه‌های مختلف نمایش داده شده در شکل ۲.۱ را نشان می‌دهند.



شکل ۲.۱: لایه‌های مختلف محاسبات ابری

۱.۳.۱ نرم‌افزار به عنوان سرویس

نرم‌افزار به عنوان سرویس شامل یک برنامه‌ی کامل است که به‌صورت یک سرویس برحسب تقاضا فراهم می‌شود. یک نمونه‌ی واحد از نرم‌افزار روی ابر اجرا می‌شود و به چندین کاربر نهایی یا مشتری سازمانی سرویس می‌دهد.

۲.۳.۱ پلتفرم به عنوان سرویس

پلتفرم به عنوان سرویس یک لایه از نرم‌افزار را به صورت بسته‌بندی شده و به عنوان یک سرویس فراهم می‌کند به‌طوری که بتوان از آن برای ایجاد سرویس‌های سطح بالاتر استفاده کرد.

[۶] Software as a service
[۷] Platform as a service
[۸] Infrastructure as a service

۳.۳.۱ زیرساخت به عنوان سرویس

زیرساخت به عنوان سرویس، قابلیت‌های محاسباتی و ذخیره‌سازی اولیه را به عنوان سرویس‌های استاندارد در شبکه ارائه می‌دهد. سرورها، سیستم‌های ذخیره‌سازی، سوئیچ‌ها، روترها و دیگر سیستم‌ها با هم به عنوان مجموعه‌ای از منابع در دسترس هستند تا بار کاری و دیگر نیازمندی‌های برنامه‌های کاربردی که به توان بالایی نیاز دارند را مدیریت کنند.

۴.۱ چرا ابر؟

به عنوان یک توسعه‌دهنده‌ی میکروسرویس، دیر یا زود باید تصمیم بگیرید سرویس‌تان را در یکی از مکان‌های زیر مستقر کنید:

* سرور فیزیکی: سازمان‌های کمی این کار را انجام می‌دهند؛ زیرا سرورهای فیزیکی محدودیت ایجاد می‌کنند. برای مثال شما نمی‌توانید ظرفیت سرور فیزیکی را سریعا افزایش دهید یا یک 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[۹]: به جای استقرار میکروسرویس روی یک ماشین مجازی کامل، بسیاری از توسعه‌دهندگان سرویس‌های خود را به صورت ظرف‌های داکر[۱۰] در محیط ابری مستقر می‌کنند.

مطالعه‌ی بیشتر

تفاوت داکر با ماشین مجازی

مزیت مهم میکروسرویس‌های تحت ابر انعطاف‌پذیری بالای آن‌هاست. ارائه‌دهندگان محیط ابری اجازه می‌دهند 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 می‌توان از نمونه‌های دیگر آن میکروسرویس استفاده کرد و برنامه را بالا نگه داشت.

[۹] Virtual container
[۱۰] Docker

فصل ۲

کنترل پیکربندی‌ها

به عنوان یک توسعه‌دهنده می‌دانید که hard-code کردن مقادیر در کدهای برنامه چندان منطقی نیست، به همین دلیل معمولا توسعه‌دهندگان از یک فایل ثابت استفاده می‌کنند و همه اطلاعات پیکربندی برنامه را در آن قرار می‌دهند. با این حال قرار دادن این اطلاعات در کدهای برنامه مشکل‌ساز است، زیرا با هر تغییر در پیکربندی، کل برنامه دوباره باید کامپایل شود. برای جلوگیری از این کار، توسعه‌دهندگان اطلاعات مربوط به پیکربندی را از کدهای برنامه جدا می‌کنند.

این مشکل در برنامه‌های تحت ابر بیشتر خود را نشان می‌دهد، از آن‌جا که ممکن است این برنامه‌ها از صدها میکروسرویس با چندین نمونه‌ی در حال اجرا تشکیل شده باشند، مدیریت پیکربندی برنامه بسیار مشکل است. برای همین توسعه‌ی 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 مخزن 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:

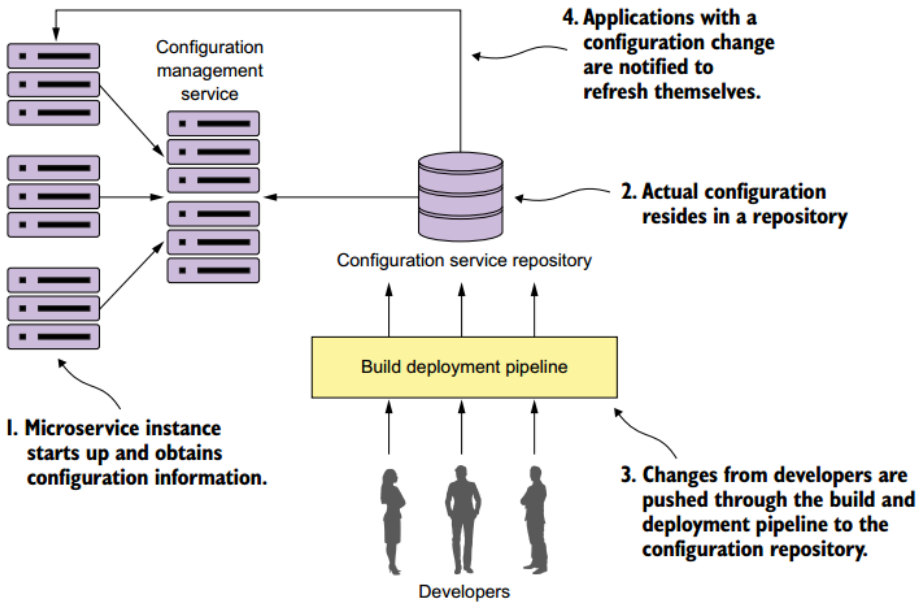
۱. قبل از اینکه یک نمونه از میکروسرویس بالا بیاید باید یک endpoint را برای خواندن اطلاعات پیکربندی مربوط به خود صدا بزند. اطلاعات لازم برای فراخوانی endpoint هنگام راه‌اندازی سرویس به آن پاس داده می‌شود.

۲. مخزن پیکربندی می‌تواند با استفاده از گیت، پایگاه‌داده‌های رابطه‌ای یا کلید-مقدار پیاده‌سازی شود.

۳. مدیریت داده‌های پیکربندی باید مستقل از چگونگی استقرار برنامه باشد.

---

[۱] 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 می‌شود. این روش نسبت به روش‌های دیگر ساده‌تر است، به خوبی با Spring Boot ادغام می‌شود و از گزینه‌های بیشتری برای ذخیره داده‌های پیکربندی پشتیبانی می‌کند.

۳.۲ پیاده‌سازی Spring Cloud configuration server

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

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 key-value 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 management, 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</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.
     sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.
     reporting.outputEncoding>
  <java.version>1.8</java.version>
  <spring-cloud.version>Finchley.RELEASE</spring-cloud.
     version>
</properties>

```

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
  </dependency>

  <!--refresh config-->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-monitor</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rabbit</
       artifactId>
  </dependency>
</dependencies>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```

در فایل Maven بالا، ابتدا نسخه‌ای از Spring Boot که در میکروسرویس‌ها از آن استفاده می‌شود، مشخص شده است. قسمت مهم بعدی (خطوط ۱۶-۱۱) مشخص کردن نسخه‌ی 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 ...