服务迁移之路 | Spring Cloud向Service Mesh转变

一、导读

Spring Cloud基于Spring Boot开发，提供一套完整的微服务解决方案，具体包括服务注册与发现，配置中心，全链路监控，API网关，熔断器，远程调用框架，工具客户端等选项中立的开源组件，并且可以根据需求对部分组件进行扩展和替换。

Service Mesh，这里以Istio（目前Service Mesh具体落地实现的一种，且呼声最高）为例简要说明其功能。 Istio 有助于降低这些部署的复杂性，并减轻开发团队的压力。它是一个完全开源的服务网格，可以透明地分层到现有的分布式应用程序上。它也是一个平台，包括允许它集成到任何日志记录平台、遥测或策略系统的 API。Istio的多样化功能集使你能够成功高效地运行分布式微服务架构，并提供保护、连接和监控微服务的统一方法。

从上面的简单介绍中，我们可以看出为什么会存在要把Spring Cloud体系的应用迁移到Service Mesh这样的需求，总结下来，有四方面的原因：

1、功能重叠

来简单看一下他们的功能对比：

功能列表	Spring Cloud	Isito
服务注册与发现	支持，基于Eureka，consul等组件，提供server，和Client管理	支持，基于XDS接口获取服务信息，并依赖"虚拟服务路由表"实现服务发现
链路监控	支持，基于Zikpin或者Pinpoint或者Skywalking实现	支持，基于sideCar代理模型，记录网络请求信息实现
API网关	支持，基于zuul或者spring-cloud-gateway实现	支持，基于Ingress gateway以及egress实现
熔断器	支持，基于Hystrix实现	支持，基于声明配置文件，最终转化成路由规则实现
服务路由	支持，基于网关层实现路由转发	支持，基于iptables规则实现
安全策略	支持，基于spring-security组件实现，包括认证，鉴权等，支持通信加密	支持，基于RBAC的权限模型，依赖Kubernetes实现，同时支持通信加密
配置中心	支持，springcloud-config组件实现	不支持
性能监控	支持，基于Spring cloud提供的监控组件收集数据，对接第三方的监控数据存储	支持，基于SideCar代理，记录服务调用性能数据，并通过metrics adapter，导入第三方数据监控工具
日志收集	支持，提供client，对接第三方日志系统，例如ELK	支持，基于SideCar代理，记录日志信息，并通过log adapter，导入第三方日志系统
工具客户端集成	支持，提供消息，总线，部署管道，数据处理等多种工具客户端SDK	不支持

分布式事务	支持，支持不同的分布式事务模式：JTA，TCC，SAGA等，并且提供实现的SDK框架	不支持
其他	……	……

从上面表格中可以看到，如果从功能层面考虑，Spring Cloud与Service Mesh在服务治理场景下，有相当大量的重叠功能，从这个层面而言，为Spring Cloud向Service Mesh迁移提供了一种潜在的可能性。

2、服务容器化

在行业当前环境下，还有一个趋势，或者说是现状。越来越多的应用走在了通往应用容器化的道路上，或者在未来，容器化会成为应用部署的标准形态。而且无论哪种容器化运行环境，都天然支撑服务注册发现这一基本要求，这就导致Spring Cloud体系应用上容器的过程中，存在一定的功能重叠，有可能为后期的应用运维带来一定的影响，而Service Mesh恰恰需要依赖容器运行环境，同时弥补了容器环境所欠缺的内容（后续会具体分析）。

3、术业有专攻

从软件设计角度出发，我们一直在追求松耦合的架构，也希望做到领域专攻。例如业务开发人员希望我只要关心业务逻辑即可，不需要关心链路跟踪，熔断，服务注册发现等支撑工具的服务；而平台支撑开发人员，则希望我的代码中不要包含任何业务相关的内容。而Service Mesh的出现，让这种情况成为可能。

4、语言壁垒

目前而言Spring Cloud虽然提供了对众多协议的支持，但是受限于Java技术体系。这就要求应用需要在同一种语言下进行开发（这不一定是坏事儿），在某种情况下，不一定适用于一些工作场景。而从微服务设计考虑，不应该受限于某种语言，各个服务应该能够相互独立，大家需要的是遵循通信规范即可。而Service Mesh恰好可以消除服务间的语言壁垒，同时实现服务治理的能力。

基于以上四点原因，当下环境，除了部分大多已经提前走在了Service Mesh实践的道路上互联网大厂以外（例如蚂蚁金服的SOFASTACK），也有大部分企业已经开始接触Service Mesh，并且尝试把Spring Cloud构建的应用，迁移到Service Mesh中。

二、Spring Cloud向Service Mesh的迁移方案

Spring Cloud向Service Mesh迁移，从我们考虑而言大体分为七个步骤，如图所示：

1. Spring Cloud架构解析

Spring Cloud架构解析的目的在于确定需要从当前的服务中去除与Service Mesh重叠的功能，为后续服务替换做准备。我们来看一个典型的Spring Cloud架构体系，如图所示：

2020年2月(1)
2019年11月(1)
2019年10月(2)
2019年9月(1)
2019年8月(1)
2019年7月(6)
2019年6月(4)
2019年5月(3)

最新评论

1. Re:服务迁移之路 | Spring Cloud向Mesh转变
@ 余大彬? ...
--
2. Re:服务迁移之路 | Spring Cloud向Mesh转变
haha
3. Re:服务迁移之路 | Spring Cloud向Mesh转变
hal>

阅读排行榜

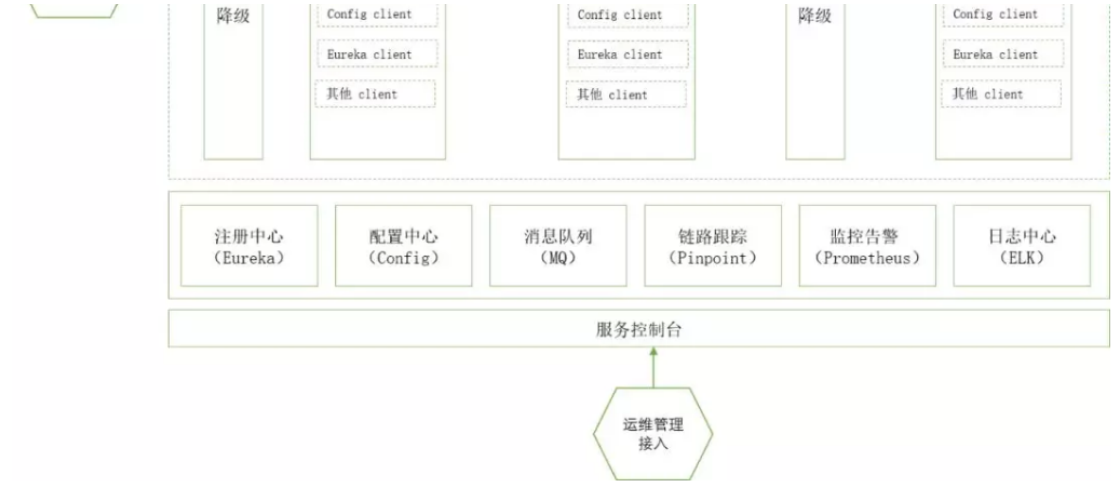
1. 微服务网关实战——Spring Cloud (638)
2. 服务迁移之路 | Spring Cloud向Se转变(3128)
3. API网关——Kong实践分享(872)
4. 容器云未来：Kubernetes、Istio 和54)
5. 开源分布式事务中间件Seata使用指

评论排行榜

1. 服务迁移之路 | Spring Cloud向Se转变(3)

推荐排行榜

1. 微服务网关实战——Spring Cloud (
)
2. 容器云未来：Kubernetes、Istio 和
)
3. 应用量化时代 | 微服务架构的服务治



从图中我们可以简要的分析出，一个基于Spring Cloud的微服务架构，主要包括四部分内容：服务网关，应用服务，外围支撑组件，服务管理控制台。

- **服务网关**

  服务网关涵盖的功能包括路由，鉴权，限流，熔断，降级等对入站请求的统一拦截处理。具体可以进一步划分为外部网关（面向互联网）和内部网关（面向服务内部管理）。

- **应用服务**

  应用服务是企业业务核心。应用服务内部由三部分内容构成：业务逻辑实现，外部组件交互SDK集成，服务内部运行监控集成。

- **外围支撑组件**
  外围支撑组件，涵盖了应用服务依赖的工具，包括注册中心，配置中心，消息中心，安全中心，日志中心等。

- **服务管理控制台**
  服务管理控制台面向服务运维或者运营人员，实现对应用服务运行状态的实时监控，以及根据情况需要能够动态玩成在线服务的管理和配置。

这里面哪些内容是我们可以拿掉或者说基于Service Mesh（以Istio为例）能力去做的？分析下来，可以替换的组件包括网关（gateway或者Zuul，由Ingress gateway或者egress替换），熔断器（hystrix，由SideCar替换），注册中心（Eureka及Eureka client，由Polit，SideCar替换），负责均衡（Ribbon，由SideCar替换），链路跟踪及其客户端（Pinpoint及Pinpoint client，由SideCar及Mixer替换）。这是我们在Spring Cloud解析中需要完成的目标：即确定需要删除或者替换的支撑模块。

2. 服务改造

服务单元改造的目的在于基于第一步的解析结果，完成依赖去除或者依赖替换。根据第一步的分析结果服务单元改造分为三步：

1. 删除组件，包括网关，熔断器，注册中心，负载均衡，链路跟踪组件，同时删除对应client的SDK；
2. 替换组件，采用httpClient 的SDK支持http协议的远程调用（原来在Ribbon中），由原来基于注册中心的调用，转变成http直接调用；
3. 配置信息变更，修改与删除组件管理的配置信息以及必要的组件交互代码（根据实际应用情况操作）；

当然服务单元改造过程中，还会涉及到很多的细节问题，都需要根据应用特点进行处理，这里不做深入分析。

3. 服务容器化

服务容器化是目前应用部署的趋势所在。服务容器化本身有很多不同的方式，例如基于Jenkins的pipeline实现，基于docker-maven-plugin + dockerfile实现，当然还有很多不同的方式。这里以Spring Cloud一个demo服务通过docker-maven-plugin+dockerfile实现说明为例：

简易的一个服务的Dockerfile如下所示：

```
ROM openjdk:8-jre-alpine
ENV TZ=Asia/Shanghai \
```

```
    SPRING_OUTPUT_ANSI_ENABLED=ALWAYS \
    JAVA_OPTS="" \
    JHIPSTER_SLEEP=0
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
CMD echo "The application will start in ${JHIPSTER_SLEEP}s..." && \
    sleep ${JHIPSTER_SLEEP} && \
    java ${JAVA_OPTS} -Djava.security.egd=file:/dev/./urandom -jar /app.jar
    # java ${JAVA_OPTS} -Djava.security.egd=environment:/dev/./urandom -jar /app.@project.packaging@

EXPOSE 8080
ADD microservice-demo.jar /app.jar
```

文件中定义了服务端口以及运行命令。

Maven-docker-plugin的插件配置如下所示：

```
<build>
    <finalName>microservice-demo</finalName>
    <plugins>
        ......
        <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <version>1.2.0</version>
            <executions>
                <execution>
                    <id>build-image</id>
                    <phase>package</phase>
                    <goals>
                        <goal>build</goal>
                    </goals>
                </execution>
                <execution>
                    <id>tag-image</id>
                    <phase>package</phase>
                    <goals>
                        <goal>tag</goal>
                    </goals>
                    <configuration>
                        <image>${project.build.finalName}:${project.version}</image>
                        <newName>${docker.registry.name}/${project.build.finalName}:${project.version}</newName>
                    </configuration>
                </execution>
                <!--暂时不添加推送仓库的配置-->
            </executions>
            <configuration>
                <dockerDirectory>${project.basedir}/src/main/docker</dockerDirectory>
                <imageName>${project.build.finalName}:${project.version}</imageName>
                <resources>
                    <resource>
                        <targetPath>/</targetPath>
                        <directory>${project.build.directory}</directory>
                        <include>${project.build.finalName}.${project.packaging}</include>
                    </resource>
                </resources>
            </configuration>
        </plugin>
    </plugins>
</build>
```

通过增加docker-maven-plugin，在执行mvn package的时候可以加载Dockerfile，自动构建服务的容器镜像（需要说明的前提是本地安装docker运行环境，或者通过环境变量在开发工具中配置Docker的远程连接环境），从而完成服务容器化改造。

4. 容器环境构建

容器环境决定这Service Mesh的部署形态，这里不详细描述容器环境的部署过程。感兴趣的朋友，可以参考https://github.com/easzlab/kubeasz 开源项目，提供了Kubernetes基于ansible的自动化部署脚本。我们也建议选择Kubernetes来构建容器环境。这里说明容器环境构建的考虑因素：

· 集群部署方案
集群部署方案主要考虑多集群，跨数据中心，存储选择，网络方案，集群内部主机标签划分，集群内部网络地址规划等多方面因素。

· 集群规模
集群规模主要考虑etcd集群大小，集群内运行实例规模（用来配置ip范围段），集群高可用节点规模等因素。

基于以上两点来考虑容器化环境的部署方案，关键是合理规划，避免资源浪费。

5. Service Mesh环境构建

Service Mesh环境构建依赖于容器环境构建，主要考虑两个方面，以Isito为例：

· 部署插件
Istio部署插件需要根据需要的场景，考虑采用的插件完整性，例如prometheus，kiali，是否开启TLS等，具体安装选项可以参考https://preliminary.istio.io/zh/docs/reference/config/installation-options/。

· 跨集群部署
依据容器环境考虑是否需要支持Isito的跨集群部署方案.

6. 服务注入

服务注入用于将容器化的服务接入到Service Mesh的平台中，目前主要有两种方式。以Isito为例说明，主要包括自动注入和手动入住。选择手动注入的目的在于可以根据企业内部上线流程，对服务接入进行人为控制。而自动注入则能够更加快捷，方便。到此实际上已经完成服务迁移工作。

7. 服务管理控制台

由于Service Mesh目前而言，多是基于声明式的配置文件，达到服务治理的效果，因此无法实时传递执行结果。基于这种原因，需要一个独立的Service Mesh的管理控制台，一方面能够查看各个服务的运行状态以及策略执行情况，另外一方面能够支持服务运行过程中策略的动态配置管理。目前而言，可以在Isito安装过程中选择kiali作为一个控制台实现，当然未来也会有大量的企业提供专门的服务。

通过以上七个步骤，能够在一定程度上帮助企业应用，从Spring Cloud迁移到Service Mesh上，但迁移过程中必然存在不断踩坑的过程，需要根据应用特点，事前做好评估规划。

三、迁移优缺点分析

Spring Cloud迁移到Service Mesh是不是百利而无一害呢?

首先，从容器化的环境出发，后续Knative，Kubernetes，Service Mesh必然会构建出一套相对完整的容器化PaaS解决方案，从而完成容器化PaaS支撑平台的构建。Service Mesh将为容器运行态提供保驾护航的作用。

其次，就目前Service Mesh的落地实现而言，对于一些特定需求的监测粒度有所欠缺，例如调用线程栈的监测（当然，从网络层考虑，或者不在Service Mesh的考虑范围之内），但是恰恰在很多服务治理场景的要求范围之中。我们也需要针对这种情况，考虑实现方案。

最后，大家一直诟病的性能和安全问题。目前已经有所加强，但是依然被吐槽。

整体而言，Spring Cloud是微服务实现服务治理平台的现状，而Service Mesh却是未来，当然也不能完全取而代之，毕竟设计思路和侧重点不同，是否迁移需要根据业务场景而定。

分类: Service Mesh

标签: servicemesh, 微服务, spring cloud

好文要顶 关注我 收藏该文

博云技术社区
关注 - 0
粉丝 - 5
+加关注

0 1

« 上一篇: 基于事件驱动机制，在Service Mesh中进行消息传递的探讨
» 下一篇: 微服务网关实战——Spring Cloud Gateway

posted @ 2019-05-20 17:20 博云技术社区 阅读(3128) 评论(3) 编辑 收藏

评论列表

#1楼 2019-05-20 17:30 余大彬

hahaha

支持(0) 反对(0)

#2楼 2019-05-20 17:32 余大彬

haha

支持(0) 反对(0)

#3楼 [楼主] 2019-05-20 17:32 博云技术社区

@ 余大彬
?

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册， 访问 网站首页。