如何正确使用 Spring Cloud？

如何更快地交付软件，每周、每天甚至每个小时向用户发布新特性？如何让新员工在入职后就能部署代码？在如此快的节奏下如何保证质量？快，我们应用开发面临的主要挑战，交付越快就越能紧密地收集到用户反馈，从而更有效地满足用户需求。

微服务、DevOps、云计算，业界应对“快”挑战的三大兵器，但其中任何一件都不是能轻松玩转的。微服务，在带来好处的同时，也引入了大量复杂度；DevOps，不仅要求团队文化、组织架构和研发流程做出调整，还对应用开发提出了新的要求；虚拟机、容器、镜像等新技术亟待学习，我们能快速跨越云计算这套技术栈吗？

Spring Cloud，它将帮我们填平横跨在应用开发与微服务、DevOps、云计算之间的沟壑，让我们轻松拥抱云上微服务，但你知道它是如何做到的吗？你对它有全面的了解吗？你知道如何正确使用它吗？新概念新技术层出不穷，让人云里雾里，你是否想拨开云雾对它们有更清晰的认知？磨刀不误砍柴工，赶快来听一听吧！

1. 微服务、DevOps、云计算之间的关系

随着互联网重构各行各业的速度不断加快，我们应用开发主要遭遇了哪些新的挑战？快，天下武功唯快不破，应用开发的速度也越来越快。软件研发流程运转越快，企业就能更快地交付软件，就能更紧密地收集用户反馈，从而更有效地满足用户需求，打造出更加优秀的产品，从而构筑起强大的竞争力，但：

- 如何做到每周、每天甚至每个小时向客户发布新特性呢？
- 如何让新员工在入职第一天甚至面试阶段就部署代码呢？
- 在代码部署之后，如何确信应用运行正常而安然入睡呢？

这就是我们应用开发面临的主要挑战，微服务、DevOps、云计算是业界应对此挑战的三大法宝：

- 微服务：互联网业务的复杂度和规模都在快速地增长，单体式应用很难满足这种发展需求，将应用程序分解成独立的微服务，每个微服务都能很好地完成一个小任务，化整为零。每个微服务可以选择最合适的技术和语言来开发，由小型团队创建和管理，彼此沟通成本较低，从而做到快速迭代。
- DevOps：微服务解决了单个应用规模大、迭代速度慢、扩展变更难等问题，提升了应用开发的速度，但整体研发流程的效率还有待于提升，传统职能矩阵式的研发方式就不太适应了，需要以产品为线索打破不同职责团队之间的隔阂，开发运维一体化就应运而生了，做到持续集成和持续交付。
- 云计算：微服务关注软件的开发期，云计算关注软件的运行期，DevOps 就是连通开发期和运行期。互联网业务的访问量都是海量的，而且存在波峰波谷，这就需要

应用能够弹性伸缩，也就要求基础资源可以自动地创建和销毁，从而引出了虚拟机和容器等云计算技术。

上述三件宝物中的任何一件都不是轻轻松松就能玩转的，接下来我们分头来看一看，在掌握使用它们时会遇到哪些问题或困难：

- 首先，应用架构从“单体式”演进到“微服务”，在拥抱“微服务”上我们会遇到什么困难呢？任何事物都有两面性，“微服务”也不例外，它在带来一些好处的同时，也引入了一些复杂度。相对于单体式架构，微服务在迭代速度、部署频率、系统性能、系统扩展性、技术栈多样性等方面更有利于快速交付，但在架构复杂度、部署难度、运维难度、问题定位、管理成本等方面都有不少新的挑战，那我们如何扬长避短真正拥抱“微服务”呢？
- 再者，DevOps，开发运维一体化，在此基础上达到更高效地持续集成和持续交付。除了对企业文化、组织结构和流程系统提出了新的要求，DevOps 也要求我们应用开发做出新的调整。应用是否足够标准化？是否能够适用部署流水线？是否能够做到一键部署等等？
- 最后，云计算，这是一种全新的 IT 基础设施，我们可以像使用水电一样使用计算、存储和网络等资源。虚拟机、容器、镜像等新技术需要我们熟悉和掌握，我们大家能否快速跨越云计算这套技术栈呢？我们的应用能否快速地迁移上云呢？怎样充分使用云计算的特性，让应用更具弹性，这些都是我们将要面临和解决的新挑战。

2. Spring、Spring Boot、Spring Cloud 之间的关系

现在我们已经初步清楚遇到了什么问题，接下来我们一起来看一下，Spring Cloud 是如何帮助我们应对这些挑战的？它会给应用开发带来哪些变化呢？我们人类文明的发展主要体现在分工合作上，就像水电成为基础设施之后，我们每家每户就不会自己生产水和电了，技术的发展也符合这个趋势，云计算就是将一些通用的、标准的技术纳入到基础设施范畴，让我们可以聚焦在业务领域，通过分工协作来提升交付的效率。Spring Cloud 也是遵循上述规则，从不同维度帮我们应用开发做了许多辅助工作，让应用开发变得更加简单。

Spring 作为应用开发框架已经存在很多年了，随着JAVA开发技术的不断发展丰富，Spring 的体量变得越来越大，配置使用也变得越来越复杂，尤其是云计算和微服务等新技术的出现，Spring Boot 和 Spring Cloud 应运而生了，新概念容易让人困惑，我们有必要厘清它们彼此的关系：

2.1 Spring

Spring Framework，一个开源 Java/Java EE 全功能栈的应用开发框架，包含了展示层、领域层、数据层等主流的技术框架：

1) 核心容器（Core Container）

- Spring-Core：核心工具类，Spring 其他模块大量使用 Spring-Core；
- Spring-Beans：Spring 定义 Bean 的支持；
- Spring-Context：运行时 Spring 容器；
- Spring-Context-Support：Spring 容器对第三方包的集成支持；
- Spring-Expression：使用表达式语言在运行时查询和操作对象；

2) AOP

- Spring-AOP：基于代理的 AOP 支持；
- Spring-Aspects：基于 AspectJ 的 AOP 支持；

3) 消息（Messaging）

- spring-Messaging：对消息架构和协议的支持。

4) Web

- Spring-Web：提供基础的 Web 集成的功能，在 Web 项目中提供Spring的容器；
- Spring-WebMVC：提供基于 Servlet 的 Spring MVC；
- Spring-WebSocket：提供 WebSocket 功能；
- Spring-WebMVC-Portlet：提供 Portlet 环境支持；

5）数据访问/集成（Data Access/Integration）

- Spring-JDBC：提供以JDBC访问数据库的支持；
- Spring-TX：提供编程式和声明式的事务支持；
- Spring-ORM：提供对对象/关系映射技术的支持；
- Spring-OXM：提供对对象/XML映射技术的支持；
- Spring-JMS：提供对JMS的支持；

除此之外，围绕着 Spring Framework 已经形成一个庞大的生态圈，包含 Spring Data、Spring Batch、Spring Security、Spring Integration、Spring AMQP、Spring Mobile、Spring for Android、Spring Web Flow、Spring Shell 等顶级子项目，通过复用让开发者可以快速开发业务应用。

2.2 Spring Boot

Spring Boot 对 Spring 做了一层简单包装，简化了 Spring 应用的开发，通过编写少量的代码就能搭建一个简单的应用。它秉持默认约定大于配置的理念，简化了原本复杂的配置文件，为 Spring 和第三方库提供开箱即用的设置，提供一套快速开发单个微服务的脚手架，让我们上手和开发更加方便高效。

2.3 Spring Cloud

一套分布式服务治理的框架，为我们提供分布式服务所依赖的配置中心、服务注册发现、断路器、负载均衡、微代理、消息总线、数据监控等套件。Spring Boot 是 Spring 的

快速配置脚手架，我们可以基于 Spring Boot 快速开发单个微服务，Spring Cloud 是基于 Spring Boot 的微服务开发套装；Spring Boot 专注于快速地开发单个微服务，Spring Cloud 关注全局的服务治理框架。

Spring 没有重复发明轮子，它只是将目前各家公司比较成熟、经过考验的服务框架组合起来，通过 Spring Boot 封装屏蔽了复杂的配置和实现原理，最终给开发者提供了一套简单易懂、易部署和易维护的分布式系统开发工具包。微服务是可以独立部署、水平扩展、独立访问的服务单元，Spring Cloud 就是这些微服务的大管家，微服务架构下组件的数量会非常多，Spring Cloud 需要提供各种方案来管理整个生态。Spring Boot 可以不依赖 Spring Cloud 独立使用，但是 Spring Cloud 离不开 Spring Boot，它们的依赖关系就是：

```
Spring -> Spring Boot > Spring Cloud
```

3. Spring 集成了哪些常用组件？

从 2004 年发布 1.0 版本开始，Spring 目前已经演进至 5.x 版本了，为不同时期的应用开发提供了强有力的支撑。现在我们正面对微服务、DevOps、云计算这些新的挑战，Spring 家族的新生力量 Spring Cloud 又将给我们提供哪些方面的支撑呢？概括起来说，我觉得主要分为四类：

- 在单个微服务的构建上，它提供了一套应用开发框架，主体是基于 Spring Framework 这个生态的开源产品。
- 在水平维度服务集成上，它以 Starter 的方式集成了大量常用组件和微服务全家桶，达到开箱即用，降低我们开发微服务的难度，提升效率，避免重复投入。
- 在垂直维度资源调度上，它可以跟 Cloud Foundry、Kubernetes、Docker 等平滑集成，让应用上云更加简单，让资源调度变得更加智能高效，让应用具备更大的弹性。
- 在研发流程全线连通上，它可以跟 DevOps 相关系统做一些配合和优化，以便应用能够更加顺畅地通过各个研发环节，让持续集成、持续交付更加高效。

接下来，我们将展开每个点来看一看。首先，我们来看一下它究竟集成了一些什么样的常用组件：

- 监控服务类，包括主机监控（Vector）、应用监控（Actuator）等；
- 存储服务类，包括关系型数据库（MySQL）、文档型数据库（MangoDB）、内存型数据库（Redis）等；
- 消息服务类，包括 ActiveMQ、RocketMQ、Kafka 等；
- 安全服务类，包括 OAuth2.0、JWT 等。

4. Spring Cloud 微服务全家桶有哪些？

除了常用组件之外，Spring Cloud 还集成了微服务全家桶，开箱即用：

- 服务注册发现类，包括：Eureka、Consul、Zookeeper、Etcd 等；

> 服务注册：每个微服务组件都向注册中心登记自己提供的服务，包括服务的主机、端口号、版本号、通讯协议等信息。注册中心按照服务类型分类组织服务清单，同时以心跳检测的方式监测清单中服务是否可用，若不可用需要从服务清单中剔除，以达到排除故障服务的效果。
>
> 服务发现：在服务治理框架下，服务间的调用不再通过具体的实例地址来实现，而是通过服务名发起请求调用实现。服务调用方通过服务名从注册中心的服务清单中获取服务实例的列表清单，通过指定的负载均衡策略取出一个服务实例位置来进行服务调用。

- 服务调用框架类，包括：Ribbon、Feign 等；

> 客户端负载均衡，将负载均衡逻辑集成到消费方，消费方从注册中心获知服务有哪些实例可用，然后再按照某种策略从这些地址中选择一个服务实例进行访问。

- 服务容错组件类，包括：Hystrix 等；

> 服务熔断：某个目标服务不可用或大量访问超时，系统将断开该服务的调用，对后续的调用请求，系统不再继续转发至此服务，直接返回失败应答，从而快速地释放资源；如果目标服务情况好转，则恢复调用。服务降级：在应急屏蔽或服务熔断情况下，直接返回本地缺省的应答。

- 统一配置中心类，包括：Spring Cloud Config、Spring Cloud Bus 等；

> 在服务构建阶段，配合构建流水线，为服务软件包或镜像提供配置；在服务运维阶段，动态调整服务配置，满足运维的灵活性需求；在服务开发阶段，提供配置 API 将配置外置化，供其他系统调用。

- 服务网关代理类，包括：Zuul、Spring Cloud Gateway 等；

> 主要提供服务路由功能，即接收消费方的所有请求，按照某种策略将请求转发至服务提供方。同时，在服务网关中完成一系列横切面的功能，例如：权限校验、请求计量、流量控制、服务质量、请求管理、响应管理、版本管理等。

- 调用链路监测类，包括：Spring Cloud Sleuth，封装了 Dapper、Zipkin 和 HTrace 等；

> 微服务架构下组件的数量众多，一个业务请求可能需要调用多个服务，调用的复杂性决定了错误和异常难以定位。我们需要知道每个请求到底有哪些服务参与，调用顺序是怎么样的，从而清楚每个调用步骤，出现问题也能很快定位。

通常我们会采用 Eureka 作为服务注册中心，实现服务注册与发现；通过 Ribbon 和 Feign 实现服务的消费以及客户端负载均衡；通过 Spring Cloud Config 实现应用不同环境的外部化配置以及版本管理。为了让服务集群更为健壮，借助 Hystrix 的融断机制来避免微服务架构中个别服务出现异常时引起故障蔓延和雪崩。服务网关 Zuul 为微服务架构门户，将权限控制、计量计费等较重的非业务逻辑内容迁移到服务路由层面，使得服务集群主体能够具备更高的可复用性和可测试性。

5. Spring Cloud 如何融合 DevOps?

接下来，我们来了解一下 Spring Cloud 在与 DevOps 融合方面可以做哪些事情，它是如何让应用持续交付更加快捷的？我们都知道，DevOps 打造了一套持续交付的流程，包括：开发、编译、测试、发布、运营等节点。如何让应用更顺畅地通过上述各个节点呢？Spring Cloud 可以在每个研发节点上做一些配合和优化：

- 开发环节，我们大家应该都试用过 Spring Initializer 创建过 Spring Boot 项目工程，除此之外我们还可以借助 Maven Archetype 来快速生成项目工程。Archetype 是 Maven 工程的模板工具包，一个 Archetype 定义了某种类型项目的基本骨架，借助它尽可能快地给用户提供示例工程。
- 测试环节，微服务通常对外提供 RESTful API，供各种类型客户端调用，而以往我们需要借助文档来记录这些 API 信息，以便其他人员查阅和测试。如果 API 发生了改变，那我们就需要同步更新文档，这会降低持续交付的效率，而 Swagger 可以帮我们自动生成 API 在线文档，与代码实现保持同步。在此基础上，我们还可以对 API 进行自动化测试。通过 Spring Boot 集成 Swagger，让接口测试变得更加自动化。
- 发布环节，使用 Spring Boot 开发的项目，可以嵌入应用服务器 Jetty、Tomcat 等，最适合发布成 Docker 镜像。我们可以提供制作镜像的 Dockerfile 模板，不再使用 Docker Maven 插件，而是在当前项目的根目录下新建一个 Dockerfile 文件，代码编写完了之后，用户只需要修改一些参数，直接手动执行命令将应用打成镜像，这样就不用深入学习容器相关技术了。
- 运营环节，我们可以接入统一配置中心、日志云、全链路追踪等平台，方便问题的定位解决。

6. Spring Cloud 如何适配云基础设施?

刚才我们已经详细介绍了 Spring Cloud 在开发框架、服务集成和融合 DevOps 等方面的内容，接下来我们看看 Spring Cloud 在适配云平台方面可以做哪些事情。通常为了让各种云计算技术栈对应用开发者透明，降低应用上云的难度，我们需要适配虚拟机、容器等不同类型的云基础设施。

虚拟机是物理机的抽象，包括操作系统、内存和存储等，在制作 VM 镜像的时候可以按需安装软件，例如：WEB 服务器和数据库等，这些软件也会出现在以该镜像启动的虚拟机中，VM 与它所在的主机，以及主机上其他 VM 相互隔离。容器在系统中只需要代码运行库环境所需的空间即可，同一个操作系统上的容器共享主机内核。由于实现原理不同，运行虚拟机和容器时所需的资源也不同。虚拟机本质上是一个完整的计算机，比容器需要更多的资源，而容器只是操作系统的一部分。一般容器集群资源密集度较低，因此在单个服务器上运行多容器要比在单个服务器上运行多 VM 更适合。

那我们的应用适合部署在哪种基础设施上呢？通常公共应用适合采用虚拟机部署，例如：消息队列、注册中心、数据库等等，这类应用对运行资源要求比较高，本身没有太多个组件构成；业务应用适合采用容器部署，我们可以将业务应用拆解成大量职责单一的微服务组件，每个组件对资源要求相对确定，而且在不同访问量下需要弹性伸缩。

因此，我们 DevOps 系统需要抽象出操作不同基础设施的管理接口，基于这些管理接口实现应用生命周期管理、服务治理等功能，这块可以借鉴国外厂商指定了一个云应用管理标准 CAMP （Cloud Application Management for Platform），它定义了应用构建、运行、管理、监控和更新等 API，以及资源模型和交互协议等。

7. Spring Cloud 填空式应用开发模式

在单个微服务构建上，Spring 已经从展示层、领域层和数据源层给我们提供了丰富的组件，我们只需要关注业务逻辑代码的编写；在服务集成上，Spring Cloud 已经提供了微服务全家桶，我们只需要引用这些服务，不需要自行搭建这些公共服务；在对接 DevOps 和 云基础设施上，我们可以做一些优化和适配。在它的支持下，我们可以更加轻松地拥抱微服务、DevOps 和云计算，更好地应对新技术挑战。遵循分工协作的理念，Spring Cloud 给我们提供了一种填空式的开发模式，在这种开发模式下，我们只需要聚焦业务和用户，从而以更快地迭代速度打磨出优秀的产品。

8. 总结

Spring 家族变得越来越庞大，包括 Spring Framework、Spring Boot、Spring Cloud 等，如果我们对它没有一个全局的认知，那我们很容易迷失在技术细节当中，也用不好这款产品。本文是作者参与公司微服务框架研发过程中积累的经验认知，可以作为 Spring Cloud 知识体系的索引，后续可以根据它深入学习某个特性。