几种常见的微服务架构方案——ZeroC IceGrid、Spring Cloud、基于消息队列、Docker Swarm

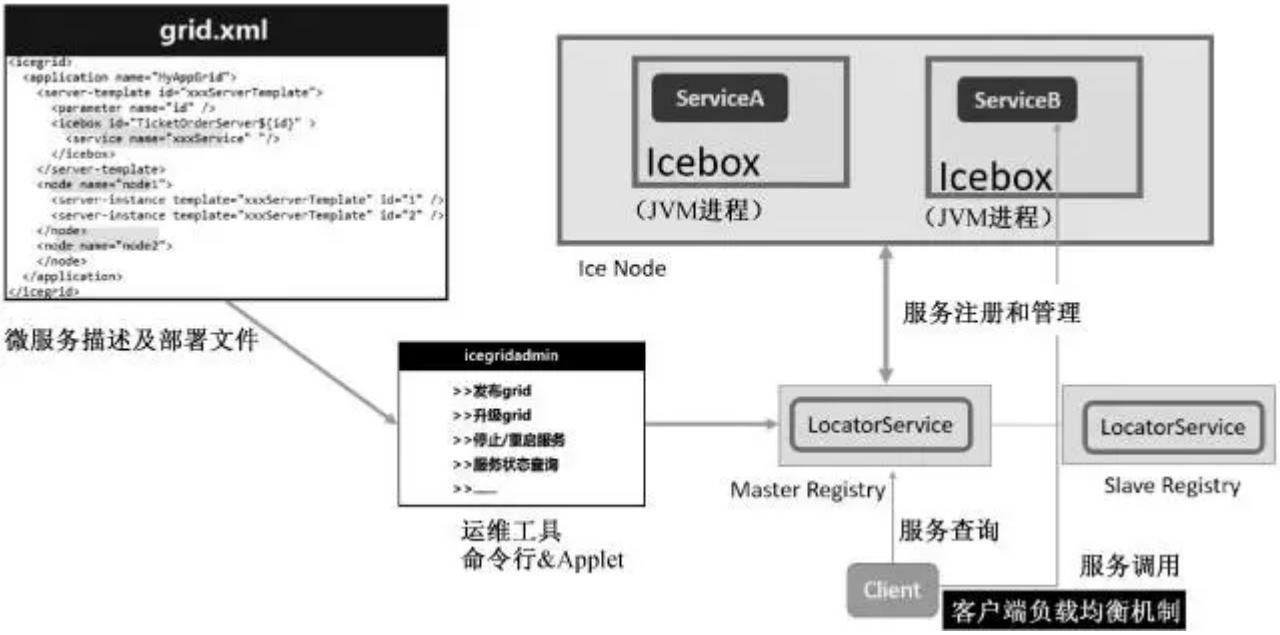
2017-10-07 Docker

微服务架构是当前很热门的一个概念，它不是凭空产生的，是技术发展的必然结果。虽然微服务架构没有公认的技术标准和规范草案，但业界已经有一些很有影响力的开源微服务架构平台，架构师可以根据公司的技术实力并结合项目的特点来选择某个合适的微服务架构平台，以此稳妥地实施项目的微服务化改造或开发进程。

本文盘点了四种常用的微服务架构方案，分别是ZeroC IceGrid、Spring Cloud、基于消息队列与Docker Swarm。

ZeroC IceGrid微服务架构

ZeroC IceGrid作为一种微服务架构，它基于RPC框架发展而来，具有良好的性能与分布式能力，如下所示是它的整体示意图。



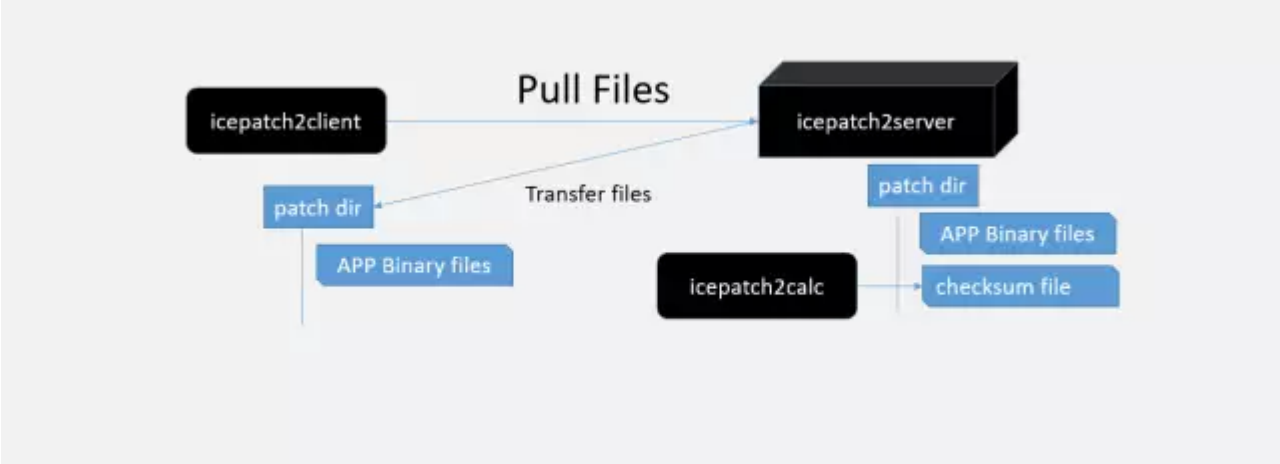
IceGrid具备微服务架构的如下明显特征。

首先，微服务架构需要一个集中的服务注册中心，以及某种服务发现机制。IceGrid服务注册采用XML文件来定义，其服务注册中心就是Ice Registry，这是一个独立的进程，并且提供了HA高可用机制；对应的服务发现机制就是命名查询服务，即LocatorService提供的API，可以根据服务名查询对应的服务实例可用地址。

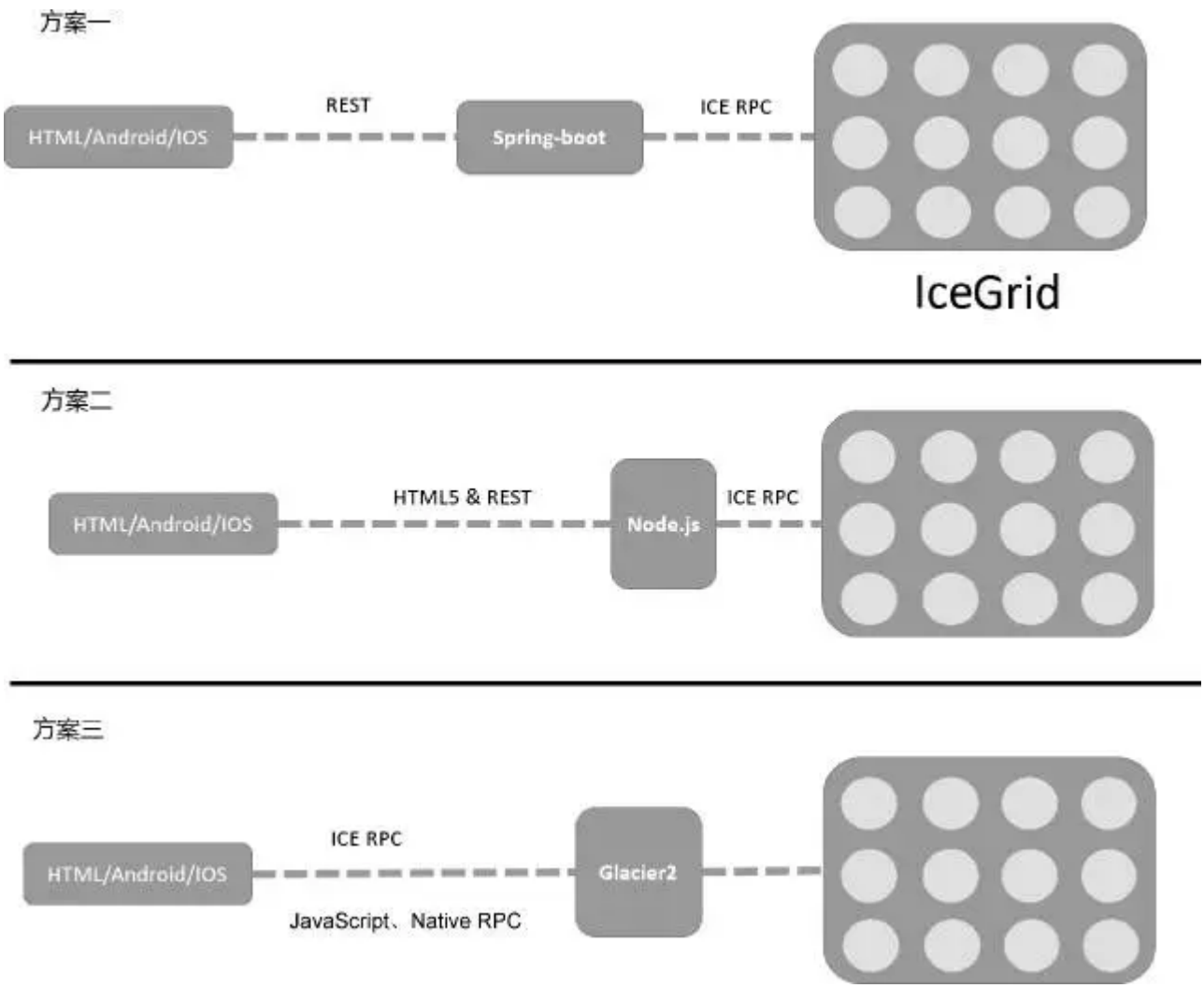
其次，微服务架构中的每个微服务通常会被部署为一个独立的进程，当无状态服务时，一般会由多个独立进程提供服务。对应在IceGrid里，一个IceBox就是一个单独的进程，当一个IceBox只封装一个Servant时，就是一个典型的微服务进程了。

然后，微服务架构中通常都需要内嵌某种负载均衡机制。在 IceGrid 里是通过客户端API内嵌的负载均衡算法实现的，相对于采用中间件Proxy转发流量的方式，IceGrid的做法更加高效，但增加了平台开发的工作量与难度，因为采用各种语言的客户端都需要实现一遍负载均衡的算法逻辑。

最后，一个好的微服务架构平台应该简化和方便应用部署。我们看到 IceGrid提供了grid.xml来描述与定义一个基于微服务架构的Application，一个命令行工具一键部署这个Application，还提供了发布二进制程序的辅助工具——icepatch2。下图显示icepatch2的工作机制，icepatch2server类似于FTP Sever，用于存放要发布到每个Node上的二进制代码与配置文件，而位于每个Node上的icepatch2client则从icepatch2server上拉取文件，这个过程中采用了压缩传输及差量传输等高级特性，以减少不必要的文件传输过程。客观地评价，在Docker技术之前，icepatch2这套做法还是很先进与完备的，也大大减少了分布式集群下微服务系统的运维工作量。



如果基于IceGrid开发系统，则通常有三种典型的技术方案，下图展示了这三种技术方案。



其中方案一是比较符合传统Java Web项目的一种渐进改造方案，Spring Boot里只有Controller组件而没有数据访问层与Service对象，这些Controller组件通过Ice RPC方式调用部署在IceGrid里的远程的Ice微服务，面向前端包装为REST服务。此方案的整体思路清晰，分工明确。Leader在开源项目中给出了这种方式的一个基本框架以供参考：https://github.com/MyCATApache/mycat-ice。

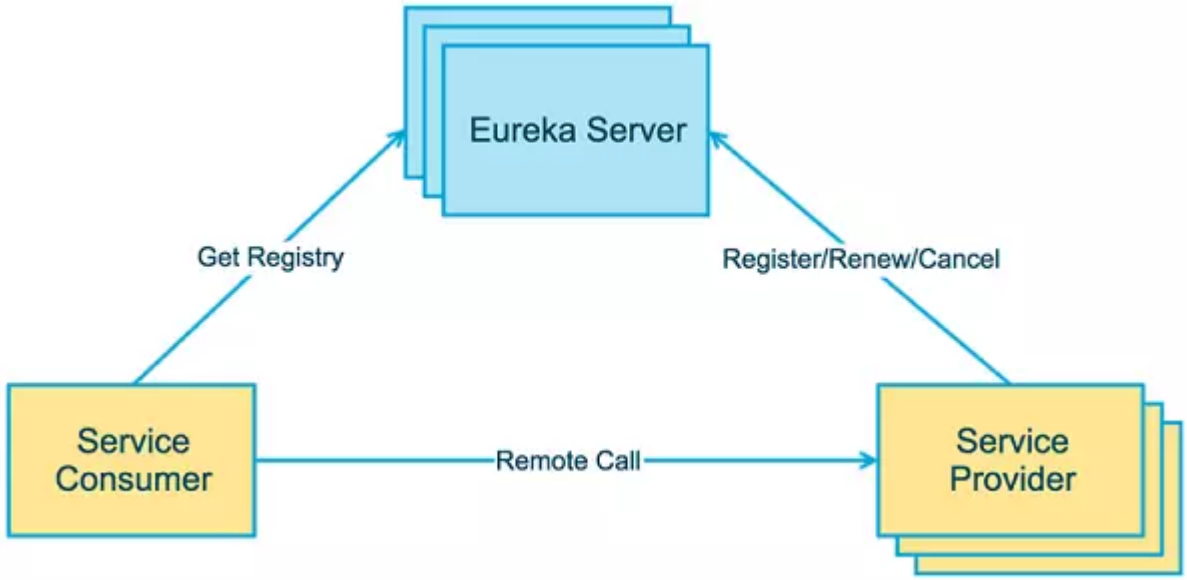
方案二与方案三则比较适合前端JavaScript能力强的团队，比如很擅长Node.js的团队可以考虑方案二，即用JavaScript来替代Spring Boot实现REST服务。主要做互联网App的系统则可以考虑方案三，浏览器端的JavaScript以HTML5的WebSocket技术与Ice Glacier2直接通信，整体高效敏捷。

IceGrid在3.6版本之后还增加了容器化的运行方式，即Ice Node与Ice Registry可以通过Docker容器的方式启动，这就简化了IceGrid在Linux上的部署。对于用Java编写的Ice微服务架构系统，我们还可以借助Java远程类加载机制，让每台Node自动从某个远程HTTP Server下载指定的Jar包并加载相关的Servant类，从而实现类似Docker Hub的机制。下图显示了前面提到mycat-ice开源项目时给出的具体实现方案。

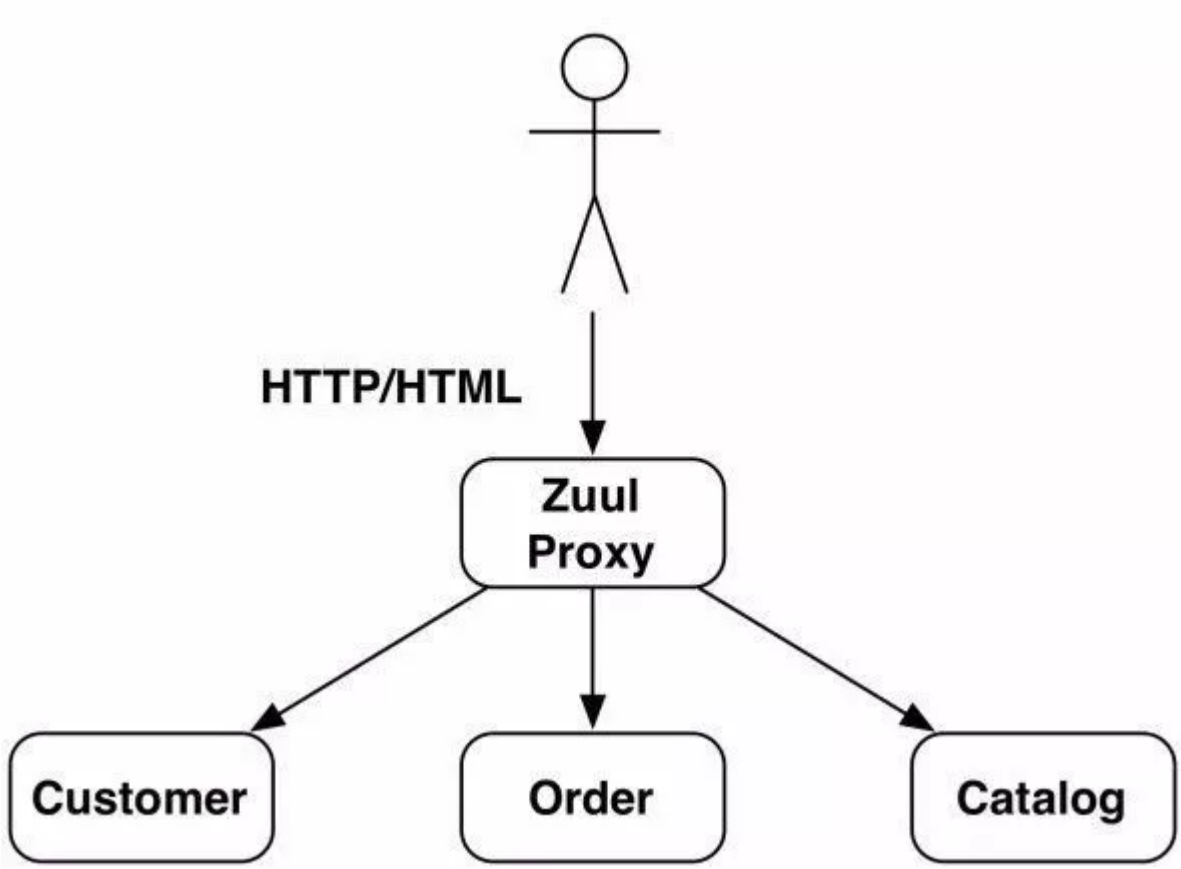
Spring Cloud微服务架构

Spring Cloud是基于Spring Boot的一整套实现微服务的框架，因此它只能采用Java语言，这是它与其他几个微服务框架的最明显区别。Spring Cloud是一个包含了很多子项目的整体方案，其中由Netflix开发后来又并入Spring Cloud的Spring Cloud Netflix是Spring Cloud微服务架构的核心项目，即可以简单地认为Spring Cloud微服务架构就是Spring Cloud Netflix，后面我们用Spring Cloud时如果不特意声明，就是指Spring Cloud Netflix。

首先，Spring Cloud中的服务注册中心是Eureka模块，它提供了一个服务注册中心、服务发现的客户端，还有一个简单的管理界面，所有服务使用Eureka的服务发现客户端来将自己注册到Eureka中，如下所示为相关示意图，你会发现它很像之前第4章中的某个图。

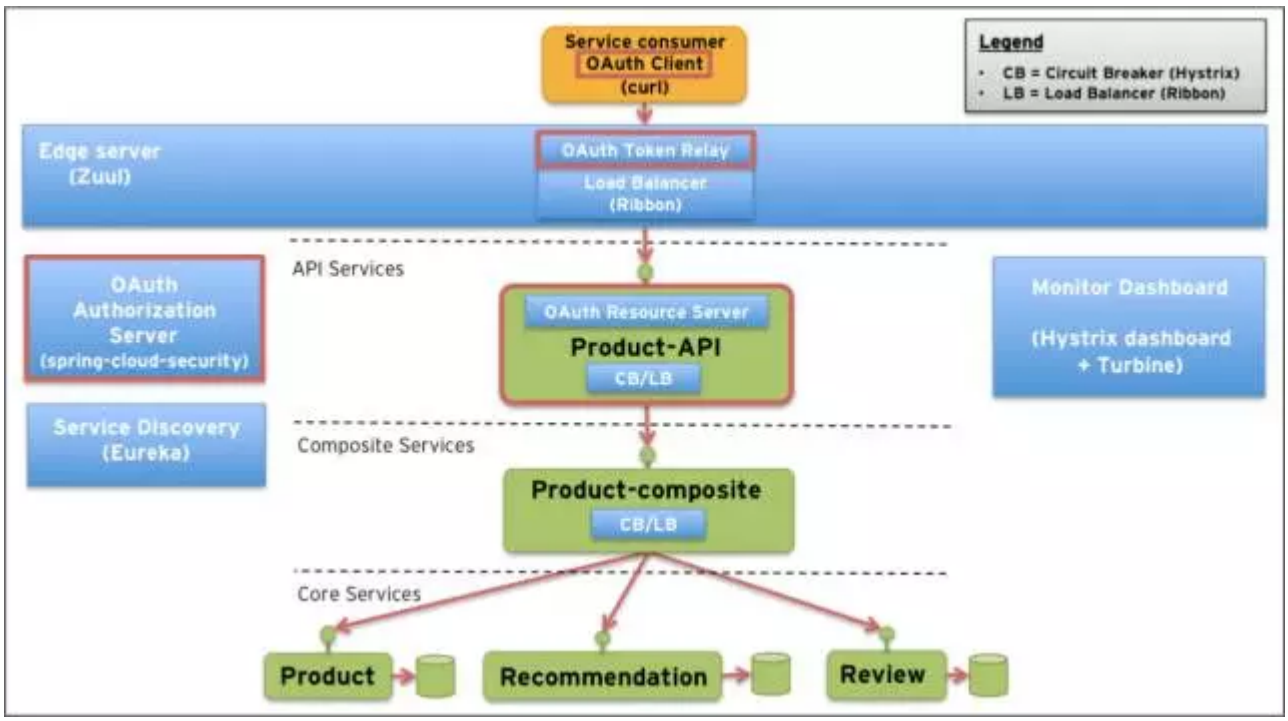


那么Spring Cloud是如何解决服务的负载均衡问题的呢？由于Spring Cloud的微服务接口主要是基于REST协议实现的，因此它采用了传统的HTTP Proxy机制。如下图所示，Zuul类似一个Nginx的服务网关，所有客户端请求都通过这个网关来访问后台的服务。



Zuul从Eureka那里获取服务信息，自动完成路由规则的映射，无须手工配置，比如上图中的URL路径/customer/*就被映射到Customer这个微服务上。当Zuul转发请求到某个指定的微服务上时，会采用类似ZeroC IceGrid的客户端负载均衡机制，被称为Ribbon组件，下图给出了Zuul与Eureka的关系及实现服务负载均衡的示意图。

如下所示是Spring Cloud微服务架构平台的全景图。我们看到它很明显地继承了Spring Framework一贯的思路——集大成！



从图中来看，Spring Cloud 微服务架构平台集成了以下一些实际项目开发中常用的技术与功能模块。

基于Spring Security的OAuth模块，解决服务安全问题。

提供组合服务（Composite Services）的能力。

电路断路器 Hystrix，实现对某些关键服务接口的熔断保护功能，如果一个服务没有响应（如超时或者网络连接故障），则Hystrix可以在服务消费方中重定向请求到回退方法（fallback

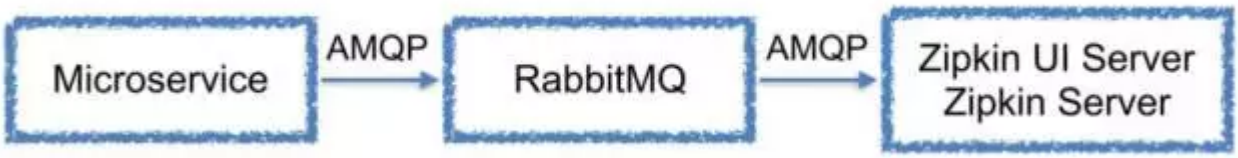
method）。如果服务重复失败，则Hystrix会快速失败（例如直接调用内部的回退方法，不再尝试调用服务），直到服务重新恢复正常。

监控用的Dashboard，可以简化运维相关的开发工作量。

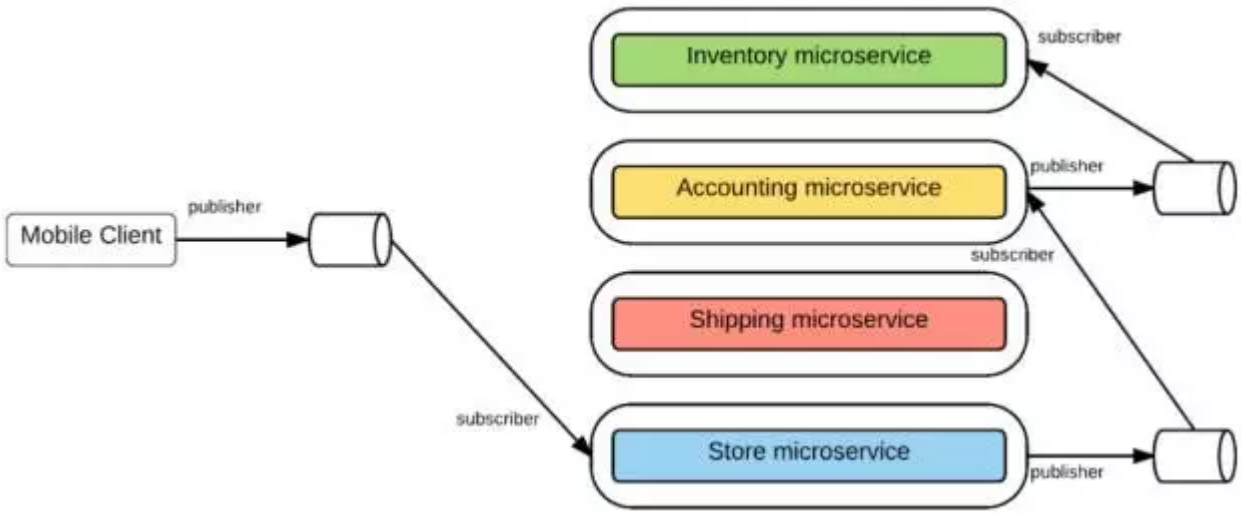
总体来说，Spring Cloud是替代Dubbo的一种好方案，虽然Spring Cloud是基于REST通信接口的微服务架构，而Dubbo以RPC通信为基础。对于性能要求不是很高的Java互联网业务平台，采用Spring Cloud是一个门槛相对较低的解决方案。

基于消息队列的微服务架构

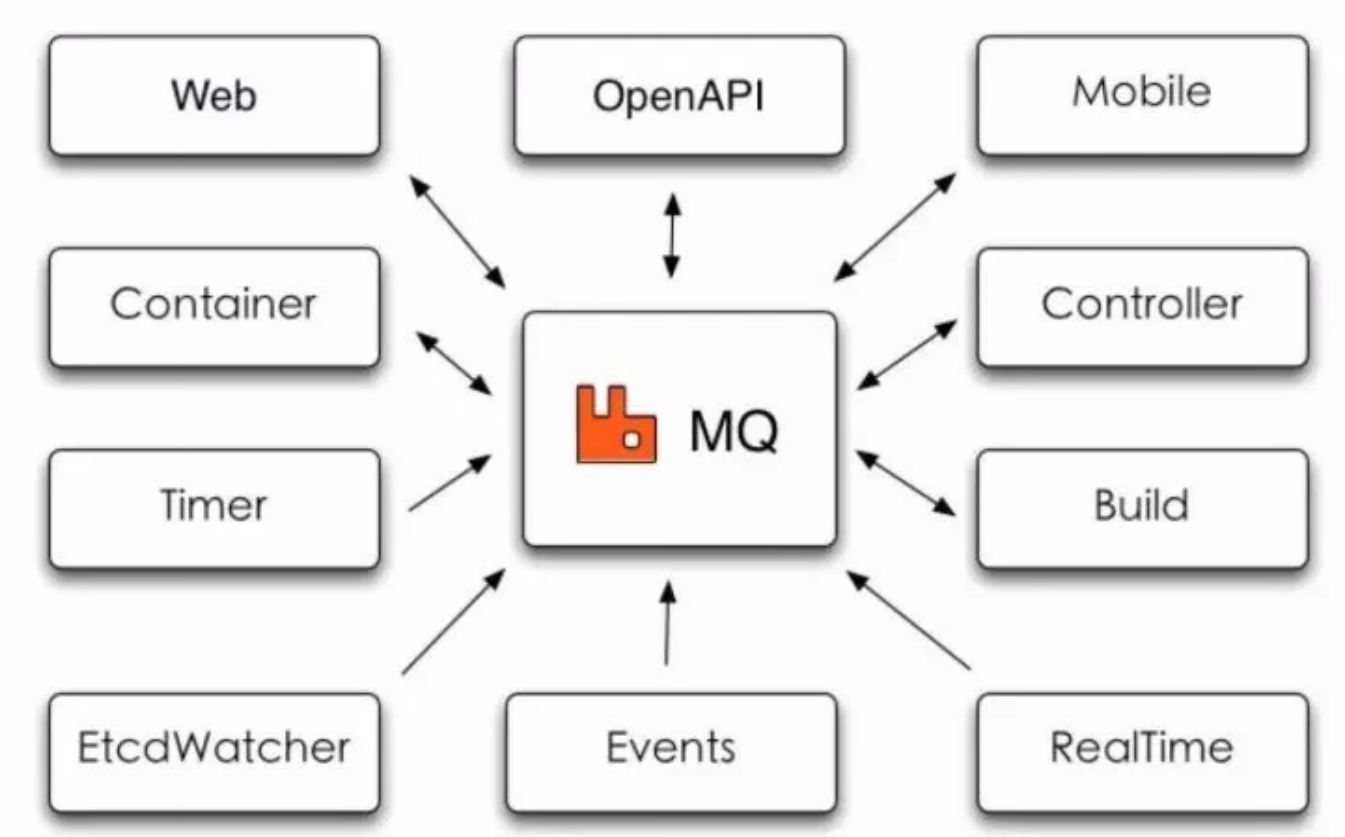
除了标准的基于RPC通信（以及类RPC的通信如Http Rest、SOAP等）的微服务架构，还有基于消息队列通信的微服务架构，这种架构下的微服务采用发送消息（Publish Message）与监听消息（Subscribe Message）的方式来实现彼此之间的交互。下图是这种微服务架构下各个组件之间的交互示意图，我们看到消息中间件是关键，它负责连通各个微服务与UI组件，担任了整个系统互联互通的重任。



基于消息队列的微服务架构是全异步通信模式的一种设计，各个组件之间没有直接的耦合关系，也不存在服务接口与服务调用的说法，服务之间通过消息来实现彼此的通信与业务流程的驱动，从这点来看，基于消息队列的微服务架构非常接近Actor模型。实际上，分布式的Actor模型也可以算作一种微服务架构，并且在微服务概念产生之前就已经存在很久了。下面是一个购物网站的微服务设计示意图，我们看到它采用了基于消息队列的微服务架构。



网易的蜂巢平台就采用了基于消息队列的微服务架构设计思路，如下图所示，微服务之间通过RabbitMQ传递消息，实现通信。

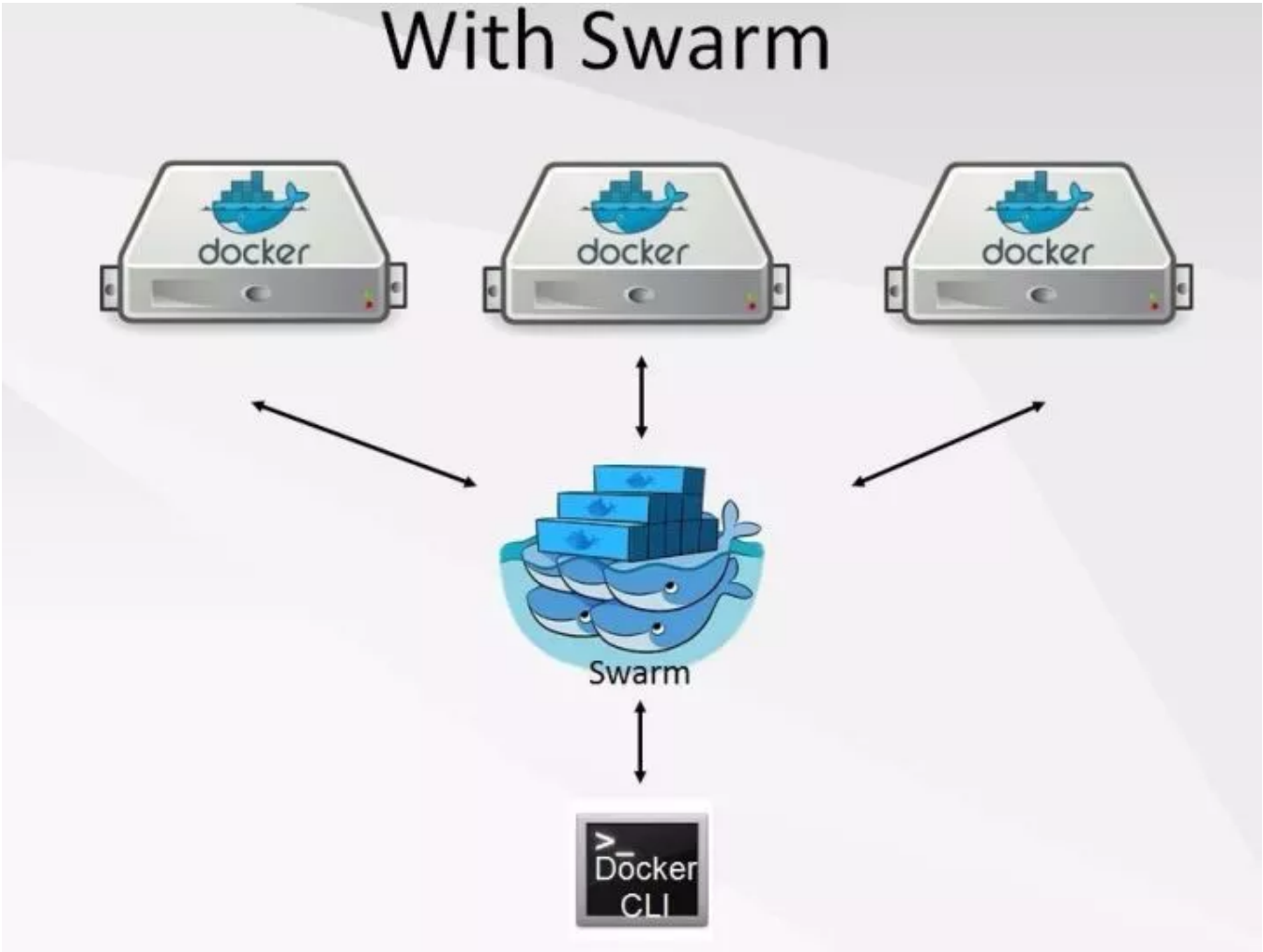


与上面几种微服务架构相比，基于消息队列的微服务架构并不多，案例也相对较少，更多地体现为一种与业务相关的设计经验，各家有各家的实现方式，缺乏公认的设计思路与参考架构，也没有形成一个知名的开源平台。因此，如果需要实施这种微服务架构，则基本上需要项目组自己从零开始去设计实现一个微服务架构基础平台，其代价是成本高、风险大，因此决策之前需要架构师“接地气”地进行全盘思考与客观评价。

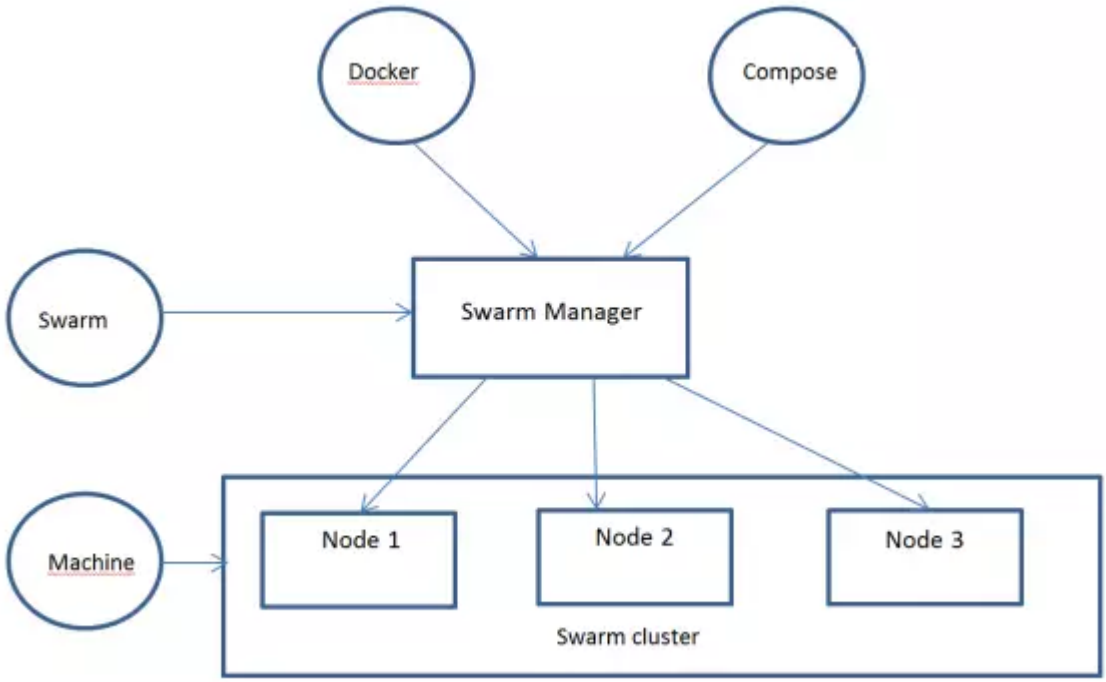
Docker Swarm微服务架构

Docker Swarm其实是Docker公司“高仿”Google开源的Kubernetes微服务架构平台的一个产品，但一直无法跟上对手的脚步，在业界始终缺乏影响力。2016年发布Docker 1.12时，Docker Swarm就被强行集成到了Docker Engine中而不再作为单独的工具发布了，这类似当年微软推广IE浏览器的做法。不过即使这样，也难以掩盖Docker Swarm还没成名就已经陨落的事实。

Docker Swarm的最初目标是将一些独立的Docker主机变成一个集群，如下图所示，我们通过简单的Docker命令行工具就能创建一个Swarm集群。



后来随着Kubernetes微服务架构平台越来越火，Docker公司开始努力让Swarm向着Kubernetes的方向靠拢，即变成一个基于容器技术的微服务平台。下面给出了Swarm集群的结构图。

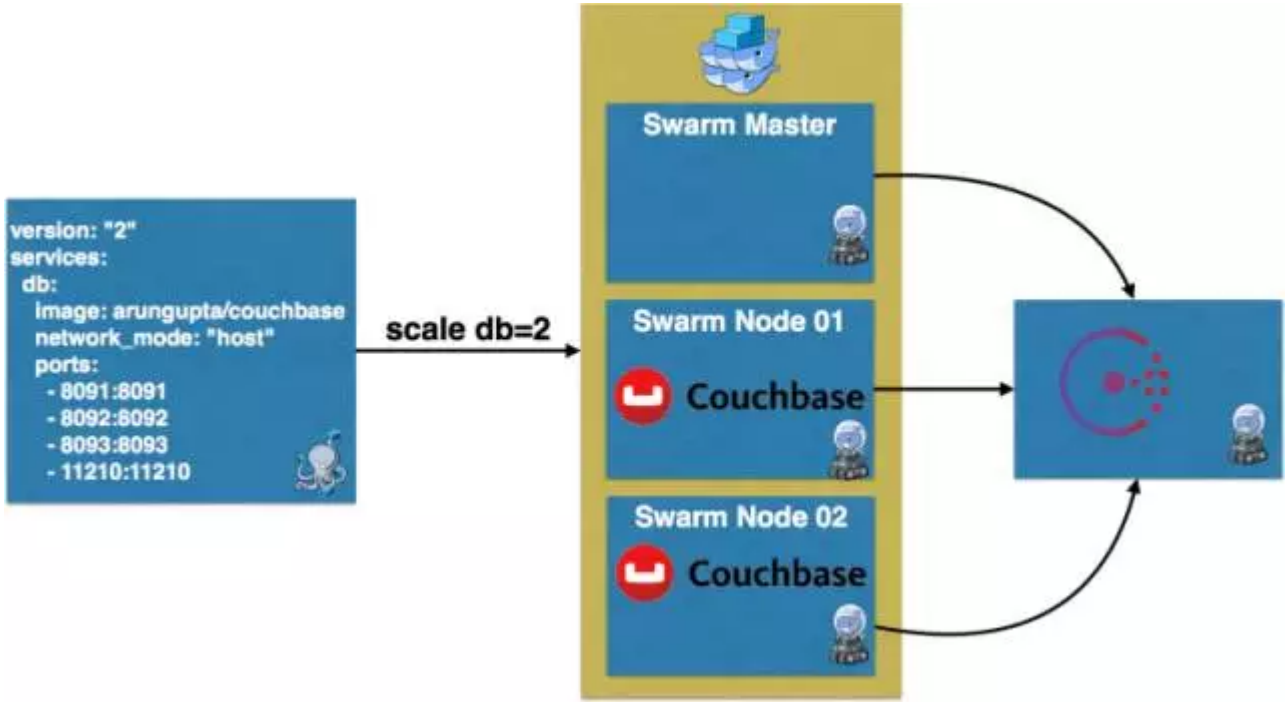


从图中我们看到一个Swarm集群中有两种角色的节点。

Swarm Manager：负责集群的管理、集群状态的维持及调度任务（Task）到工作节点（Swarm Node）上等。

Swarm Node：承载运行在Swarm集群中的容器实例，每个Node主动汇报其上运行的任务（Task）并维持同步状态。

上图中的Docker Compose是官方编排（Orchestration）项目，它提供了一个YAML格式的文件，用于描述一个容器化的分布式应用，并且提供了相应的工具来实现一键部署的功能。下图给出了两节点的Couchbase集群对应的YAML文件定义，此Couchbase集群随后被部署到了Swarm集群中的两个Node节点上。



注意上图左边YAML文件中的Services定义，Swarm manager节点给每个Service分配唯一的DNS名字，因此可以通过最古老又简单的DNS轮询机制来实现服务的发现与负载均衡，这明显借鉴了Kubernetes的做法。

由于Docker Swarm高仿了前辈Kubernetes的设计，而且在微服务架构中并没有太多的影响力，所以我们在此并未做深入介绍。

原文链接：http://www.broadview.com.cn/article/348

Spring Cloud实战训练营

本次培训内容包括：微服务架构及概述、微服务架构项目实战目标、Spring Boot概述、Spring Cloud简介与入门、Eureka、Ribbon、Feign、Hystrix、Zuul、Spring Cloud Config、Spring Cloud Sleuth等，点击识别下方二维码加微信好友了解**具体培训内容**。