大规模软件开发试验

第一组
大规模软件开发的思考

指导老师： 袁昕

组　长　： 彭子帆

组　员　： 陈　诺 陈思启 代艺博
孔成俊 李林瀚 刘乐为
王　鹏 王　汀 王钟毓
席吉华（按照姓氏排序）

2020 年 6 月 21 日

目录

一、 压力测试

1.1 相关的说明

本次压力测试的系统为浙江大学大规模软件开发试验课程第一小组开发的校内论坛系统，该论坛系统可供师生在论坛网站上进行交流讨论，具有注册登录、发帖回帖、上传图片等功能。本次压测的功能主要有登陆、发帖以及查看最新帖子。

本次测试在 2020-06-20 22:39:00 开始，在 2020-06-20 23:19:00 停止。最大 VU（并发用户数）数为 100，最大 TPS（每秒事务数）为 2.10。

系统要达到的性能指标

1) 单机部署情况下 TPS 不小于 20

2) 系统运行高峰时期，响应时间小于 4 秒

3) TPS 能满足二八原则

1.2 压力测试的效果

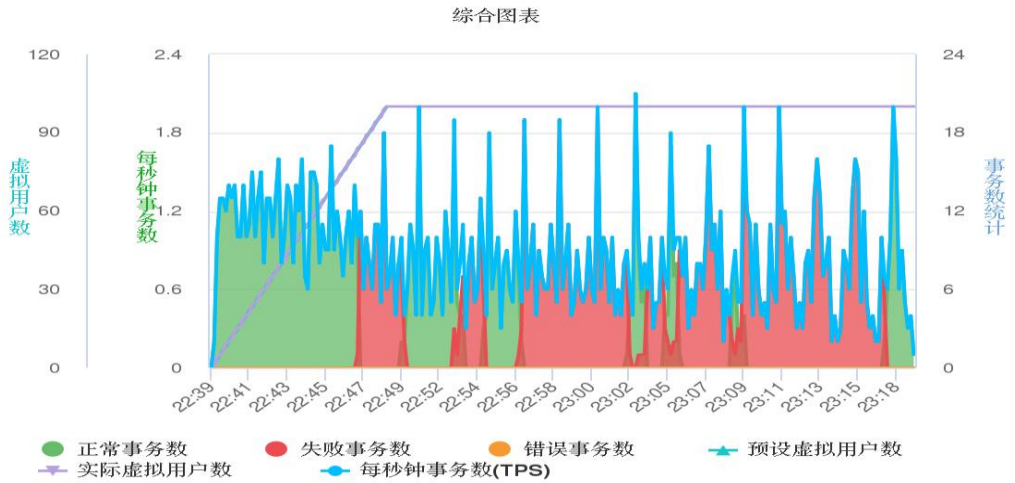
1.2.1 测试结果总结

本次压力测试具体各脚本概要情况如下表:

序号	脚本	总事务	成功事务	成功率	平均响应时间
1	test	2108	1030	48.86 %	77726ms

1.2.2 压力测试综合报表

本次压力测试最大并发用户数为 100，发起的总事务数为 2108，失败事务数为 1078，错误事务数为 0，每秒钟平均事务处理能力 TPS 为 0.88。



由于测试结果比较丰富，我们已经完成了完整的压力测试报告，已经放到附件中，和该报告一起提交。希望老师查阅!

二、 微服务

2.1 思考方向

微服务架构的核心思想是一个应用由多个小的、相互独立的、微服务组成。不同服务通过一些轻量级交互机制来通信，例如 RPC、HTTP 等。本次校园论坛项目预期将使用 Spring Cloud 进行微服务，但受限于时间，将仅给出思考方向和我们认为可能的解决方案。

2.2 解决方案

2.2.1 服务的注册与发现

Eureka Server 进行服务的注册与发现功能。

1. 在项目中引入 spring-cloud-starter-eureka-server 依赖内容。
2. 添加@EnableEurekaServer 注解到启动类上。
3. 在配置文件中添加 registerWithEUR 额卡和 fetchRegistry 信息。

Service Prodiver 为服务提供方，将自身服务注册到 Eureka 注册中心，从而使服务消费方能够找到。

1. 在项目中引用 spring-cloud-starter-eureka-server 依赖内容。
2. 添加@EnableDiscoveryClient 注解到启动类上。
3. 添加配置找到 Eureka 服务器。

2.2.2 服务的分布式一致性

Consul 使用 Raft 算法保证一致性，支持多数据中心，内外网的服务采用不同的端口进行监听。

1. 安装并配置 Consul
2. 添加 spring-cloud-starter-consul-discovery 依赖
3. 配置 application.yml 文件及服务注册功能

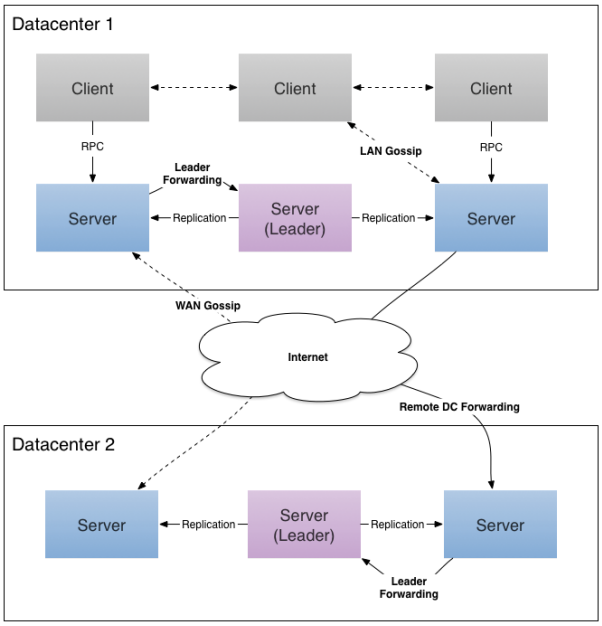


图 1 Consul 服务架构图

2.2.3 Eureka 提供服务与 Ribbon 服务消费

Ribbon 提供客户端软件的负载均衡算法。其提供四种负载均衡策略，分别为简单轮循负载均衡、随机负载均衡、加权响应时间负载均衡、区域感知轮询负载均衡。

1. 添加 spring-cloud-starter-ribbon 依赖
2. 添加 restRemplate 注解支持负载均衡
3. 在 application.yml 指定服务的注册中心地址、服务端口和服务名称

2.2.4 断路器监控

在分布式架构中，如果某一服务单元发生故障，将通过断路器的故障监控向调用方返回错误响应，而不是长时间的等待。

1. 添加 spring-cloud-starter-hystrix 断路器依赖。
2. 在启动类中通过@EnableHystrix 开启 Hystrix 断路器监控，并设置断路器信息。

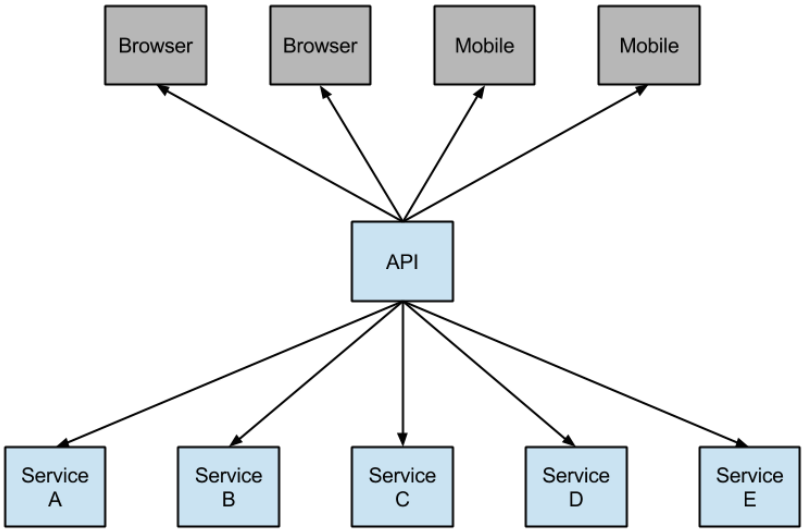


图 2 微服务示意图

2.2.5 服务网关

在服务网关中，包含如权限检验、限流以及监控等功能。通过将权限校验的逻辑写在网关的过滤器中，后端服务不需要关注权限校验的代码。而想修改权限校验的逻辑，只需要修改网关中的权限校验过滤器，不需要升级已存在的微服务。用户访问将直接请求网关，网关将对请求进行智能路由转发。

1. 添加 org.springframework.cloud 依赖项
2. 创建服务网关项目，并添加@EnableZuulProxy 注解开启服务网关
3. 设置 application.yml 文件，并添加端口号

2.2.6 高可用的分布式配置中心

由于分布式系统服务数量巨大，将通过分布式配置中心组建对服务配置文件进行管理，。服务端负责将 git svn 中存储的配置文件发布成 REST 接口，客户端从服务端的 REST 接口中获取配置。

1. 添加 org.springframework.cloud 依赖
2. 在启动类中添加@EnableConfigServer 注解开启 SpringCloudConfig 服务端
3. 在配置文件中设置仓库的地址、分支和路径。

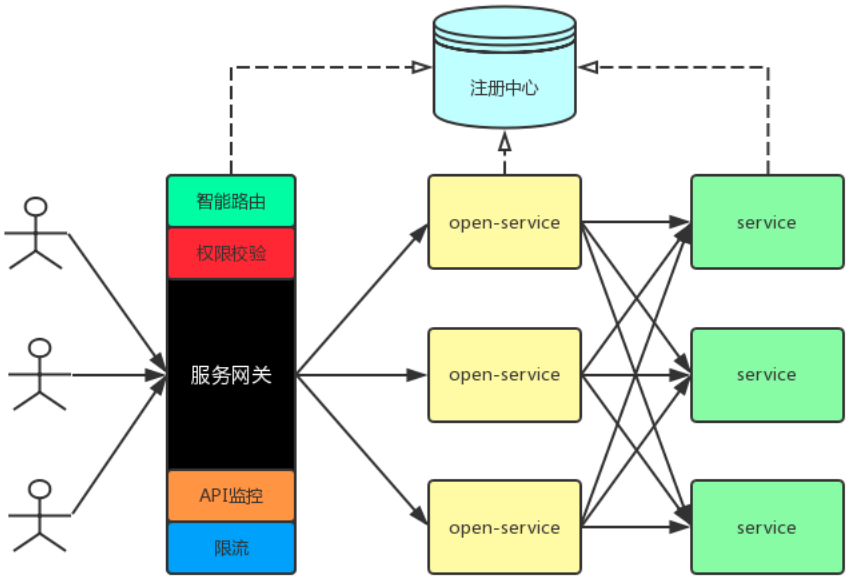


图 3 服务网关设计图

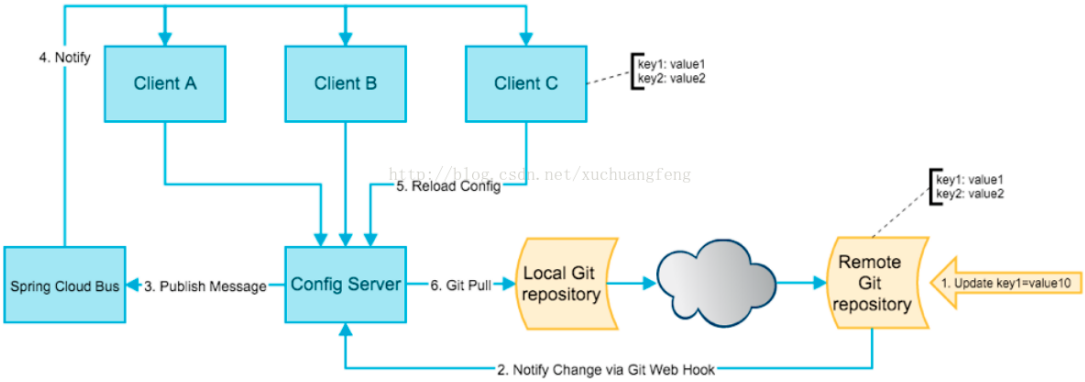


图 4 SpringCloudConfig 工作流程示意图

2.1.7 基于 Hystrix 进行容错处理

基于 Hystrix 实现容错处理只需要完成以下两步。

1. 对于非 Feign 的普通方法，加上@HystrixCommand 注解
2. 定义回退方法

图 5 示范了如何 findById 方法是如何整合进 Hystrix 中的。

```java
@HystrixCommand(fallbackMethod = "findByIdFallback")
@GetMapping(value = "/user/{id}")
public User findById(@PathVariable Long id) {
    return restTemplate.getForObject("http://user-service/" + id, User.class);
}

public User findByIdFallback(Long id){
    User user = new User();
    user.setId(-1L);
    user.setUsername("Default User");

    return user;
}
```

图 5 通用方法整合 Hystrix 示例

2.2.8 使用 Spring Cloud Bus 自动刷新配置

Spring Cloud Bus 基于轻量级地消息代理（例如 RabbitMQ、Kafka 等）连接分布式系统的节点，就可以通过广播的方式来传播状态的更改（例如配置的更新）或者其他的管理指令。

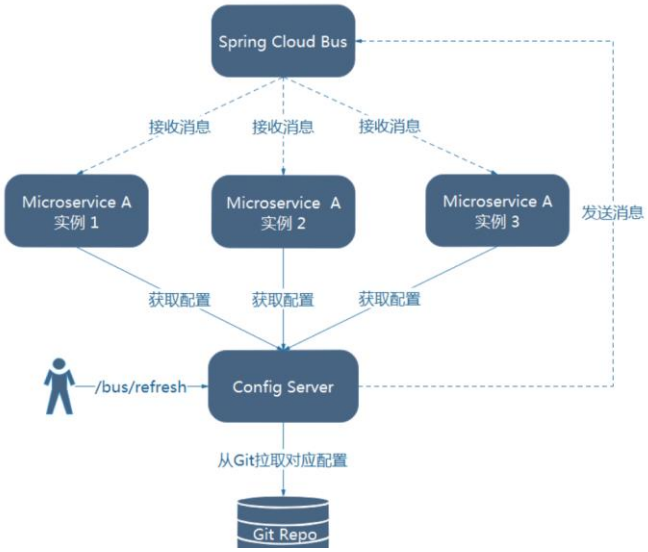


图 6 Spring Cloud Bus 架构示例

它将 Config Server 加入消息总线之中，并使用 Config Server 的/bus/refersh 端点来实现配置的刷新。这样，各个微服务只需要关注自身的业务逻辑，而无需再自己手动刷新配置。

三、 监控

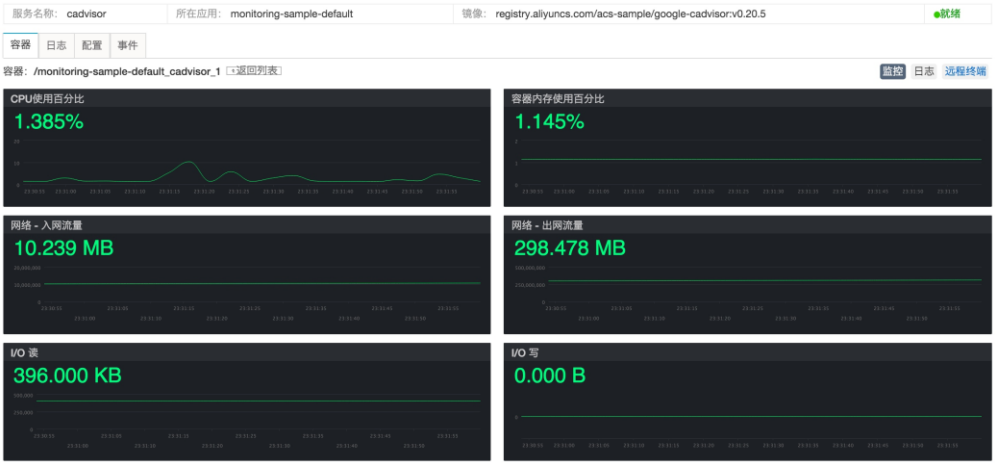
3.1 思考方向

我们小组在服务器监控方面主要考虑了两方面的工作。一方面，由于我们小组运用的是阿里云服务器，作为一个成熟的云服务器提供商，阿里云本身已经提供了一些简单的监控功能。另一方面，我们也可以考虑使用开源的一些监控工具，进一步精细化我们服务器的监控工作。

3.2 解决方案

3.2.1 阿里云自带监控

本次大规模软件开发试验采用的是阿里云服务器，进入阿里云控制台，点击云监控菜单下的云服务器 ECS，即可实时监控服务器的状态信息。包括 CPU 使用率情况、内存使用率、网络流入带宽、网络流出带宽、服务器 TCP 连接数等数据状况。



阿里云服务器 ECS 提供网络安全、服务器安全等基础防护措施，并支持多种实时预警保障业务安全。

具体功能包括:

1. DDoS 基础防护：

提供最高 5G 的 DDoS 防护能力，可防御 SYN flood、UDP flood、ICMP flood、ACK flood 常规 DDoS 攻击

2. 服务器安全功能（安骑士）：

包含暴力破解密码拦截、木马查杀、异地登录提醒、高危漏洞修复的防入侵功能

3. 免费提供云监控，并支持多种实时预警

站点监控：

提供对 http、ping、dns、tcp、udp、smtp、pop、ftp 等服务的可用性和响应时间的统计、监控、报警服务

云服务监控：

提供对云服务的监控报警服务，对用户开放自定义监控的服务，允许用户自定义个性化监控需求。

4. 报警及联系人管理：

提供对报警规则，报警联系人的统一、批量管理服务。支持多报警方式：短信、邮件、旺旺、接口回调。

在控制台总览页，可以查看到待处理的告警事件数量及其紧急程度、检测到的告警事件总数、已处理事件的数量等信息。点击后即可进入对应的功能模块进行处理。

阿里云容器服务不但提供了核心的容器和宿主机监控能力，也支持定制或集成自己的监控解决方案，我们可以在阿里云容器服务上打造自己的 Docker 监控框架。

Docker 本身提供了 Docker stats 命令和 stats API。我们可以通过 docker stats [CONTAINER] 或 docker stats -a 列出指定容器或所有容器的性能信息，如果你想要在图形界面中直接查看这些信息，一种可行的方案是使用 Cadvisor 工具，阿里云容器服务的特性支持 Cadvisor 容器在集群每个节点上的“全局”部署，在选择 Cadvisor 服务后，我们可以在控制台中看到容器的端口地址，通过这些地址，我们可以在浏览器里打开 cAdvisor 控制台，方便地获得指定节点 OS 和 Docker 容器的监控信息。

3.2.2Kurbernete/k8s

3.2.2.1 简介

首先，它是一个全新的基于容器技术的分布式架构领先方案。Kubernetes(k8s)是 Google 开源的容器集群管理系统（谷歌内部:Borg）。在 Docker 技术的基础上，为容器化的应用提供部署运行、资源调度、服务发现和动态伸缩等一系列完整功能，提高了大规模容器集群管理的便捷性。

Kubernetes 是一个完备的分布式系统支撑平台，具有完备的集群管理能力，多扩多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和发现机制、内建智能负载均衡器、强大的故障发现和自我修复能力、服务滚动升级和在线扩容能力、可扩展的资源自动调度机制以及多粒度的资源配额管理能力。同时 Kubernetes 提供完善的管理工具，涵盖了包括开发、部署测试、运维监控在内的各个环节。

使用 Web 服务，用户希望应用程序能够 7*24 小时全天运行，开发人员希望每天多次部署新的应用版本。通过应用容器化可以实现这些目标,使应用简单、快捷的方式更新和发布，也能实现热更新、迁移等操作。使用 Kubernetes 能确保程序在任何时间、任何地方运行，还能扩展更多有需求的工具/资源。Kubernetes 积累了 Google 在容器化应用业务方面的经验，以及社区成员的实践，是能在生产环境使用的开源平台。

3.2.2.2 优势

在 Kubernetes 集群中，它解决了传统 IT 系统中服务扩容和升级的两大难题。你只需为需要扩容的 Service 关联的 Pod 创建一个 Replication Controller 简称（RC），则该 Service 的扩容及后续的升级等问题将迎刃而解。在一个 RC 定义文件中包括以下 3 个关键信息。

1. 目标 Pod 的定义
2. 目标 Pod 需要运行的副本数量（Replicas）
3. 要监控的目标 Pod 标签（Label）

在创建好 RC 后，Kubernetes 会通过 RC 中定义的的 Label 筛选出对应 Pod 实例并实时监控其状态和数量，如果实例数量少于定义的副本数量，则会根据 RC 中定义的 Pod 模板来创建一个新的 Pod，然后将新 Pod 调度到合适的 Node 上启动运行，知道 Pod 实例的数量达到预定目标，这个过程完全是自动化。

总体来说，Kubernetes 主要有以下几个优势:

1. 容器编排
2. 轻量级
3. 开源

4. 弹性伸缩
5. 负载均衡

3.2.3 总体架构

K8s 集群由两节点组成：Master 和 Node。在 Master 上运行 etcd,Api Server,Controller Manager 和 Scheduler 四个组件。后三个组件构成了 K8s 的总控中心，负责对集群中所有资源进行管控和调度.在每个 node 上运行 kubectl,proxy 和 docker daemon 三个组件,负责对节点上的 Pod 的生命周期进行管理，以及实现服务代理的功能。另外所有节点上都可以运行 kubectl 命令行工具。

API Server 作为集群的核心，负责集群各功能模块之间的通信。集群内的功能模块通过 Api Server 将信息存入到 etcd,其他模块通过 Api Server 读取这些信息，从而实现模块之间的信息交互。Node 节点上的 Kubelet 每隔一个时间周期，通过 Api Server 报告自身状态，Api Server 接收到这些信息后，将节点信息保存到 etcd 中。Controller Manager 中的 node controller 通过 Api server 定期读取这些节点状态信息，并做响应处理。Scheduler 监听到某个 Pod 创建的信息后，检索所有符合该 pod 要求的节点列表，并将 pod 绑定到节点列表中最符合要求的节点上。如果 scheduler 监听到某个 Pod 被删除，则调用 api server 删除该 Pod 资源对象。kubelet 监听 pod 信息，如果监听到 pod 对象被删除，则删除本节点上的相应的 pod 实例，如果监听到修改 Pod 信息，则会相应地修改本节点的 Pod 实例。

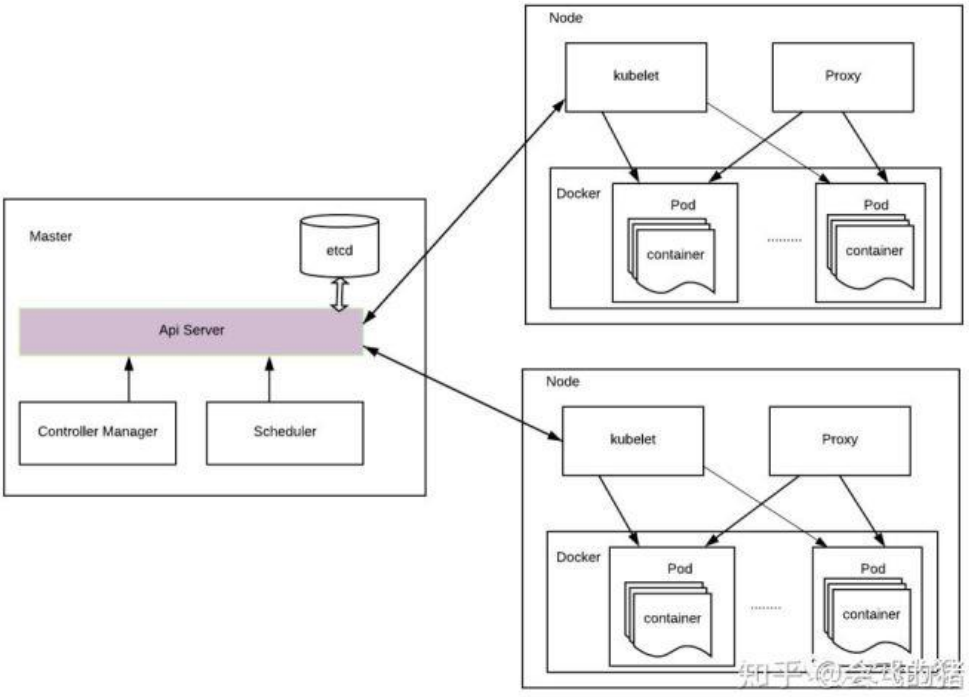


图 5.1 k8s 架构一览

Kubernetes 主要由以下几个核心组件组成：

etcd 保存了整个集群的状态；

apiserver 提供了资源操作的唯一入口，并提供认证、授权、访问控制、API 注册和发现等机制；

controller manager 负责维护集群的状态，比如故障检测、自动扩展、滚动更新等；

scheduler 负责资源的调度，按照预定的调度策略将 Pod 调度到相应的机器上；

kubelet 负责本 Node 节点上的 Pod 的创建、修改、监控、删除等生命周期管理，同时 Kubelet 定时“上报”本 Node 的状态信息到 Api Server 里；

Container runtime 负责镜像管理以及 Pod 和容器的真正运行（CRI）；

kube-proxy 负责为 Service 提供 cluster 内部的服务发现和负载均衡。

四、 弹性

4.1 思考方向

弹性负载均衡（Elastic Load Balance，简称 ELB）是将访问流量根据转发策略分发到后端多台服务器流量分发控制服务。弹性负载均衡可以通过流量分发扩展应用系统对外的服务能力，通过消除单点故障提升应用系统的可用性。

弹性负载均衡的一些优点：

1. **高性能**

   集群支持最高 1 亿并发连接，满足用户的海量业务访问需求。

2. **高可用**

   采用集群化部署，支持多可用区的同城双活容灾，无缝实时切换。

3. **灵活扩展**

   根据应用流量自动完成分发，与弹性伸缩服务无缝集成，灵活扩展用户应用的对外服务能力。

4. **简单易用**

   快速部署 ELB，实时生效，支持多种协议、多种调度算法可选，用户可以高效地管理和调整分发策略。

5. **可靠性**

   仅增强型负载均衡具有，公网私网均支持跨可用区双活容灾，支持一致性 Hash，流量分发更均衡。

6. **运维**

   仅增强型负载均衡具有，支持按监听器粒度监控性能指标，更方便客户业务统计。

弹性负载均衡的应用：

1. **会话保持**

   对于业务量访问较大的业务，可以通过 ELB 设置相应的转发策略，将访问量均匀的分到多个后端处理。例如大型门户网站，移动应用市场等。同时还可以开启会话保持功能，保证同一个客户请求转发到同一个后端，从而提升访问效率，如 图 1 所示。

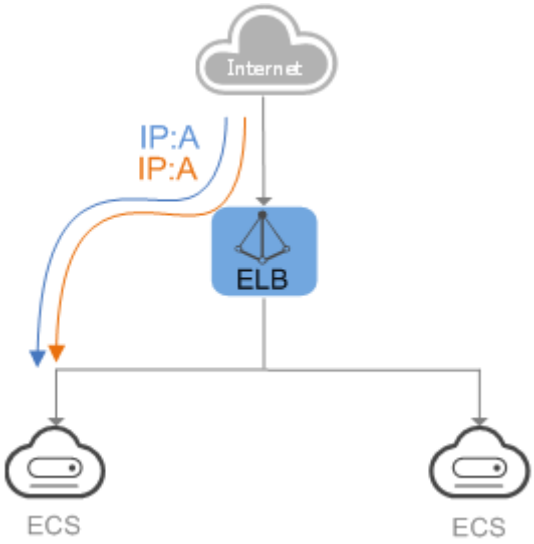


图 1

2. 灵活扩展

对于存在潮汐效应的业务，结合弹性伸缩服务，可以随时在 ELB 上添加和移除后端，更好的提升业务的灵活扩展能力，如 图 2 所示。例如电商，手游，直播网站等。

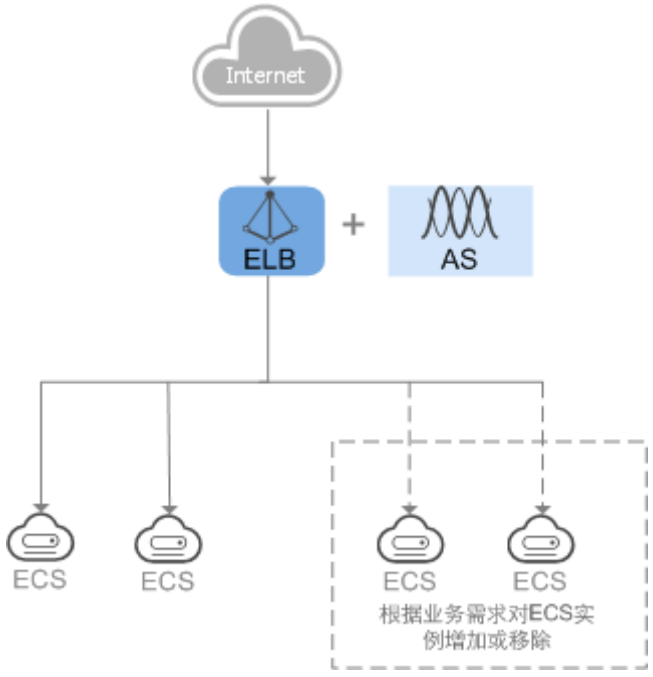


图 2

3. 消除单点故障

对于可靠性有较高要求的业务，可以在负载均衡器上添加多个后端 云服务器 。负载均衡器会通过健康检查及时发现并屏蔽有故障的 云服务器 ，并将流量转发到其他正常运行的后端 云服务器 ，确保业务不中断，如 图 3 所示。例如官网，计费业务，Web 业务等。

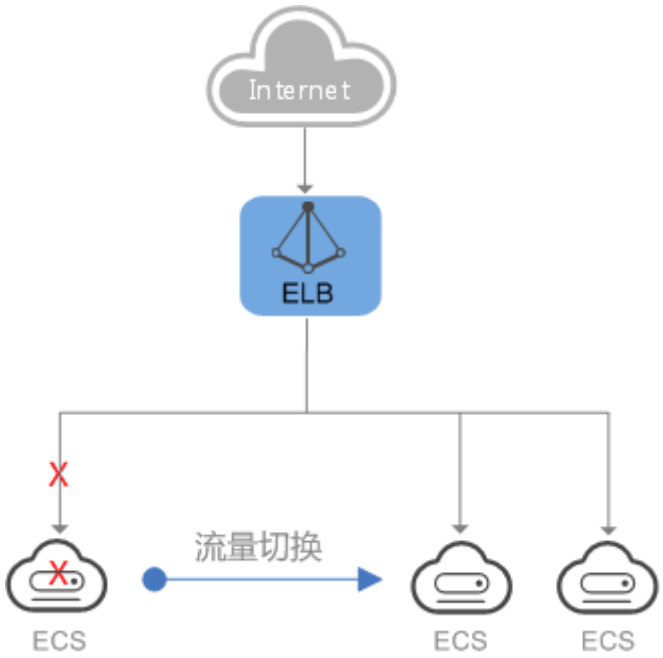


图 3

4. 高可用保障

对于可靠性和容灾有很高要求的业务，弹性负载均衡可将流量跨可用区进行分发，建立实时的业务容灾部署。即使出现某个可用区网络故障，负载均衡器仍可将流量转发到其他可用区的后端云服务器进行处理，如图 4 所示。例如银行业务，警务业务，大型应用系统等。

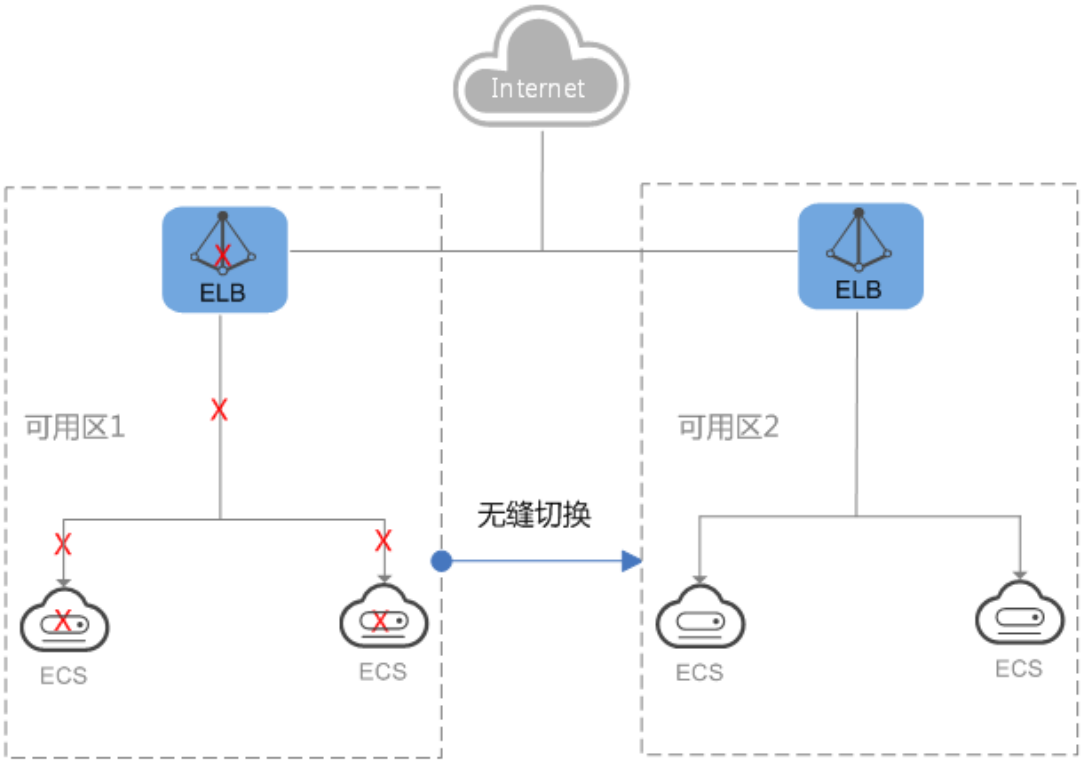
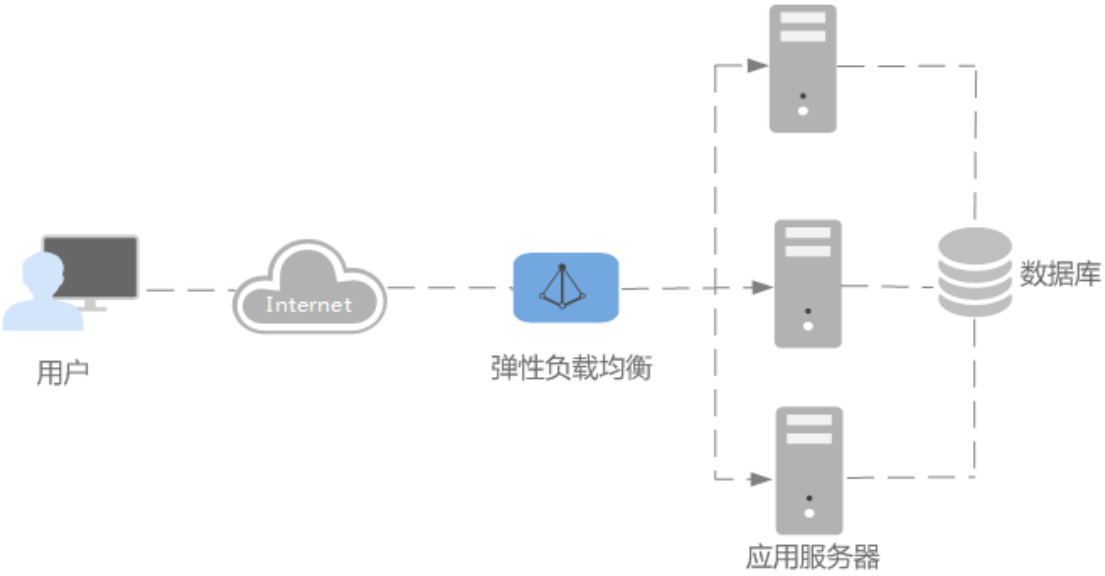


图 4

因此通过弹性负载我们达到如下效果图：



4.2 解决方案

由于目前论坛用户量较少，使用多台轻量应用服务器进行负载均衡即可有效支持用户访问请求。对于弹性服务，本次将仅做思路上的探索。

1. 使用阿里提供的 ECS 弹性可伸缩计算服务。

2. 使用阿里的 ECI 弹性容器实例，支持秒级弹性伸缩，

3. 百度云提供的普通型 BLB

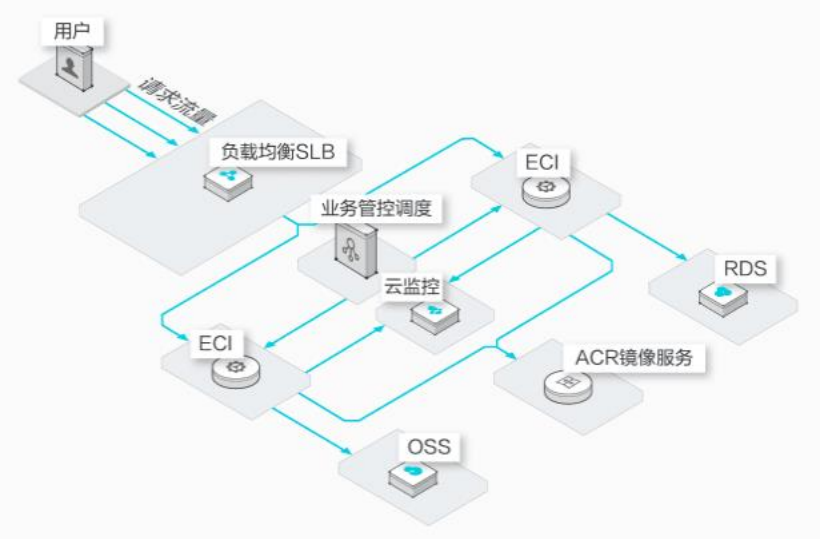
提供了负载均衡常规功能组件，能够满足用户对负载均衡的通用需求，具备轻量化、配置简单、价格便宜等特点。

- 按实例维度组织后端服务器，每个监听器下所有后端服务器监听端口相同
- 流量转发方式：监听器统一调度
- 支持轮询、最小连接数、源 IP 的流量转发

4. 百度云提供的应用型 BLB

针对用户复杂应用部署架构，特别是大型网站架构。使用基于策略的网络管理框架构建，实现业务驱动的流量负载均衡。

- 支持服务器组，可以按监听器维度组织后端服务器，每个监听器下所有后端服务器监听端口可以不同
- 流量转发方式：按策略转发
- 支持根据域名/URL 做流量转发



ECI 弹性容器实例工作流程图

五、 安全

5.1 思考方向

服务器、网站的安全直接影响着我们的系统的稳定性、可用性、用户体验等等方面。当前的网络的服务逐渐增加，由于我们的网站的的公开化，导致网站经常遭到破坏，如果不及时维护将会影响整个网站乃至服务器的使用。网络病毒等恶意攻击行为对网站服务器造成了不良的影响，影响网络服务器的正常使用。这不仅仅不利于网站的维护同时也造成了网络信息的破坏。

我们一下的解决方案主要根据小组服务器遭到攻击、破坏时想到的相应解决措施以及可能遇到的问题的预防措施:

5.2 解决方案

5.2.1 问题：服务器数据库非法登陆

问题由来：SQL 被攻击，所有的表以及表内数据被清空。

问题关键：SQL 放置在服务器上被频繁攻击，是因为服务器本身的安全性不强，而且 SQL 默认对外网开放，如果密码设置的简单的话，很容易就被攻击者破解，导致服务器数据外泄。

解决方法：

1） 首先最重要的是更改连接 SQL 的登陆密码，原先设置的密码是 123456，被破解的概率非常高，在被攻击了两次之后，我们将登陆密码改的更复杂，由数字、英文字母

和特殊字符组成，使得破解密码的复杂度提升了几个次方。

深入思考：

数据库被攻击最让人心烦的是数据丢失问题，因此，在维护数据库的时候，不能只关注数据库的安全性，也要考虑数据丢失后，如何恢复止损。因此，数据库备份是非常重要的步骤，将数据库备份文件存放在本地，即便数据库有被攻击的风险，在被攻击后也可以做到快速回复，不影响前后端开发的进度。

此外，之前也提到数据库被攻击也是服务器被攻击的表现之一，提升服务器的安全性才能根本解决数据库在服务器内的安全风险。从这个角度来看，设置服务器防火墙，以关闭任何外部访问的可能性是可行的方法，将在下一个问题中被提到。

一般攻击者默认攻击服务器的 3306 端口，因为这是 mysql 的默认端口。因此我们可以改变 mysql 的端口配置，让攻击者难以高效率的找到服务器的具体端口。此外，mysql 也最好和服务器一样，禁止远端链接，需修改 mysql 的配置文件。由于我们在修改了数据库登录密码后就再也没有出现过被攻击的情况，因此也没有做后续的工作。

5.2.2 问题：服务器防火墙被攻击

问题由来：Redis 数据库进程被攻击，CPU 被占用高达 90%

问题关键：和上一个问题相同，Redis 数据库的默认配置也是支持外部连接的，在没有设置服务器防火墙的时候，很容易遭到外部攻击。

解决方法：

1）首先需要配置服务器的防火墙：`iptables -A INPUT -s 132.97.0.0/16 -p tcp -m tcp --dport 21 -j ACCEPT` 输入以上指令配置允许远程连接服务器 ip 地址，以上例子表明 132.97.0.0 这一域下的地址可以访问服务器的 21 端口。

2）在配置了防火墙后，Redis 依然有被攻击，此时改变 Redis 的默认 6379 端口，转移到其他端口，目的和 Mysql 相同,并禁用服务器的 6379 端口，做到禁止向公网开放 redis。

深入思考：

从网上的方法来看，如果不需要远程控制服务器，可以将服务器 tcp 端口绑定在 127.0.0.1(本地端口)，这一方法对于我们来说是可行的。如果需要远程控制服务器的话，需要实行加密传输的方式，在 docker 的宿主机上，生成 CA 的私有和公钥，然后创建服务器的注册证书请求（CSR）以及生成一个 Server key，用 CA 签署公钥，只有拥有注册证书的用户端才能正常访问服务器。

要进一步提升 Redis 的安全性，可以使用非 root 权限访问 redis 的方法，具体做法是在用 root 权限安装 Redis 后，用非 root 权限登录，并复制 redis 的配置文件到当前用

户的根目录，修改配置文件中 redis 运行使用到的相关文件和目录的路径把 pidfile /var/run/redis_6379.pid 修改成为 pidfile /home/‘用户名’/run/redis_6379.pid；之后，在当前用户的根目录下创建一个 run 文件夹，并在根目录下启动 redis。

5.2.3 问题：用户被非法登陆

问题由来：可能出现黑客使用密码爆破，偷取数据库密码等方式来进行登陆

问题关键：防止密码流失和黑客进行密码爆破

解决方法：

（1）密文存储：数据库中使用密文储存密码，通过使用密码明文和用户 id 进行单向哈希加密，存储在数据库中。在登陆检验时，进行同样的哈希加密，用于检测密码是否正确。

（2）增加登陆检验：如使用验证码登陆，或增加邮箱检验等方式，来确认用户的信息

深入思考：

增加登陆检验和密文储存，都是增加黑客破译密码的时间成本，但没有解决黑客使用自动化程序暴力破解的问题。因此可以增加 IP 检验，当一个 IP 地址对某个用户申请过多登陆请求，就可以将该 IP 封禁一段时间。

5.2.4 问题：黑客注入攻击

问题由来：黑客可能使用 SQL 等注入攻击的方式进行不法行为

问题关键：在用户输入数据时，黑客可能使用一些不合法的输入，实现其不法目的

解决方法：

（1）后端数据库查询：使用规范化参数匹配数据库函数，防止一些非法输入，能够防止 SQL 注入等数据库攻击

深入思考

同时还可以考虑对 xml 注入攻击的防御。较为直接的处理，可以直接过滤掉部分非法符号以及其转义。同时，要考虑在编写前后端代码时，不能直接存储和显示，否则可能遭受攻击，前端代码也应该尽可能减少重要接口的暴露。而对于一些大量旧的，难为维护的代码，可以统一使用一层过滤，先进行过滤后，再进行调用