|  |
| --- |
| Pactera |
| 可持续集成的微服务框架 |
| 基于SpringCloud,Docker,Jenkins实现 |

|  |
| --- |
| 李哲 林松 高强  2017-10-1 |

# 前言

“微服务”一词是在过去几年里涌现出来的，它是一种独立部署的软件应用设计方式。

在以往的传统单体应用开发中我们经常会碰到如下问题：

1. 系统复杂，内部多个模块之间紧密耦合，对象（应用）之间互相依赖，在开发，部署以及测试期间牵一发而动全身。
2. 扩展困难，传统的单体应用在需要动态扩容时往往需要对整个系统进行复制和集群，无法针对系统瓶颈处进行单独的优化处理，变更和升级都十分困难。
3. 不利于后期维护，由于单一应用组件之间的耦合度过高，维护期间的任何改动，升级和系统迁移都可能对整个系统造成影响。

微服务的核心意义在于每个组件运行在独立的进程（或服务器）上，并通过轻量级协议（如restful service）进行沟通。这样的好处在于：

1. 每个服务只专注于一个业务功能
2. 服务可以由不同的团队开发
3. 所有服务之间是低耦合的
4. 服务可以跨语言开发
5. 在云环境蓬勃发展的今天，服务可以被很容易的切割，分散部署在不同的云环境中

本文将基于SpringCloud（应用层），Docker（工作容器）和Jenkins（自动持续集成工具）来讨论如何构建一套完整的，可持续集成的，易于维护并适用于敏捷开发模型的基本开发框架。

目录

[前言 1](#_Toc495306712)

[1. 结构 5](#_Toc495306713)

[2. Springcloud实现 6](#_Toc495306714)

[2.1 注册中心 6](#_Toc495306715)

[2.1.1 pom.xml 6](#_Toc495306716)

[2.1.2 application.properties 6](#_Toc495306717)

[2.1.3 Application.java 7](#_Toc495306718)

[2.2 配置中心 8](#_Toc495306719)

[2.2.1 application.properties 8](#_Toc495306720)

[2.2.2 application-dev.properties 8](#_Toc495306721)

[2.2.3 Application.java 8](#_Toc495306722)

[2.3 EmployeeService 10](#_Toc495306723)

[2.3.1 bootstrap.properties 10](#_Toc495306724)

[2.3.2 EmployeeController 10](#_Toc495306725)

[2.3.3 Response 10](#_Toc495306726)

[2.4 EmployeeService的负载均衡 11](#_Toc495306727)

[2.4.1 bootstrap.properties 11](#_Toc495306728)

[2.4.2 EmployeeController 11](#_Toc495306729)

[2.4.3 Response 12](#_Toc495306730)

[3. Docker实现 13](#_Toc495306731)

[3.1 Dockerfile 14](#_Toc495306732)

[4. 持续集成（基于Jenkins） 15](#_Toc495306733)

[5. 快速部署（基于Ansible） 16](#_Toc495306734)

[6. 监控 19](#_Toc495306735)

[7. 总结 20](#_Toc495306736)

[7.1如何组织团队 20](#_Toc495306737)

[7.2如何快速搭建本地开发环境 20](#_Toc495306738)

[7.3如何快速测试 22](#_Toc495306739)

[7.4如何保持持续集成和发布 22](#_Toc495306740)

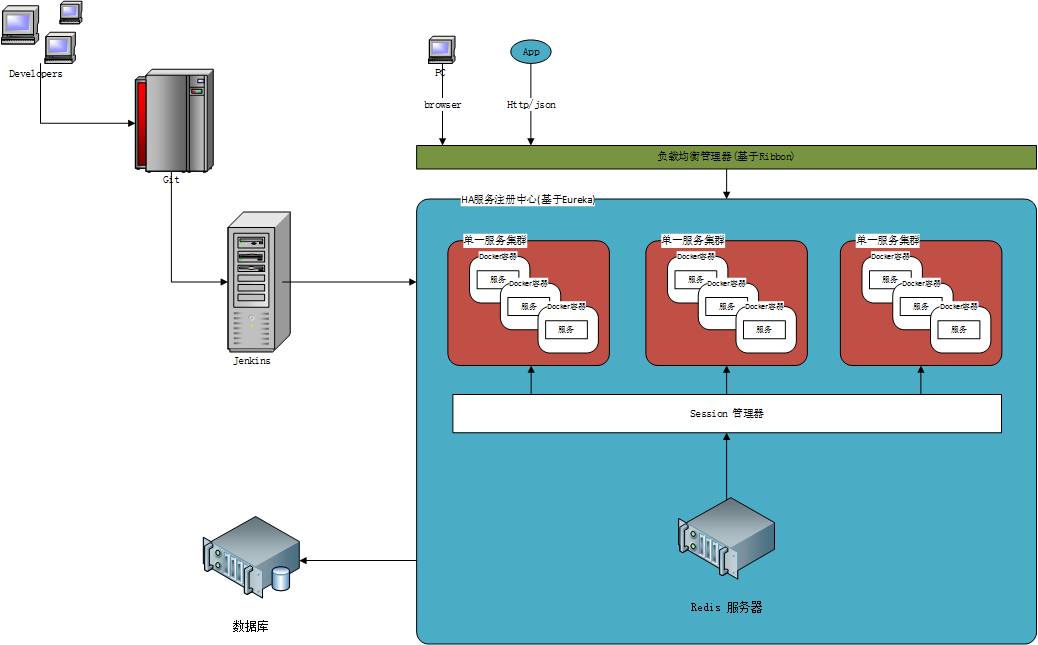
[7.5如何最小化系统差异带来的不稳定性 23](#_Toc495306741)

[7.6如何更有效的尝试、整合其他新技术 24](#_Toc495306742)

[7.7如何快速的应对横向扩展和压力测试 24](#_Toc495306743)

# 结构

* 每个服务运行在独立的Docker容器中
* 每个独立功能由一个相同服务的集群组成
* 所有的服务都在服务注册中心注册
* 调度器会基于特定算法进行负载均衡调度
* 各个服务之间使用Redis共享session



# Springcloud实现

## 2.1 注册中心

微服务框架中一个重要的概念是“服务的注册和发现”，你需要一个中心管理器来管理项目中的服务，在本文中我们使用Eureka来实现这个功能。这里基于Springcloud来使用Eureka非常简单，只需要下面几个步骤。

### pom.xml

使用Maven添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-eureka-server</artifactId>  </dependency> |

### application.properties

因为我们使用Springboot内嵌的tomcat作为容器所以需要指定一下tomcat的相关配置

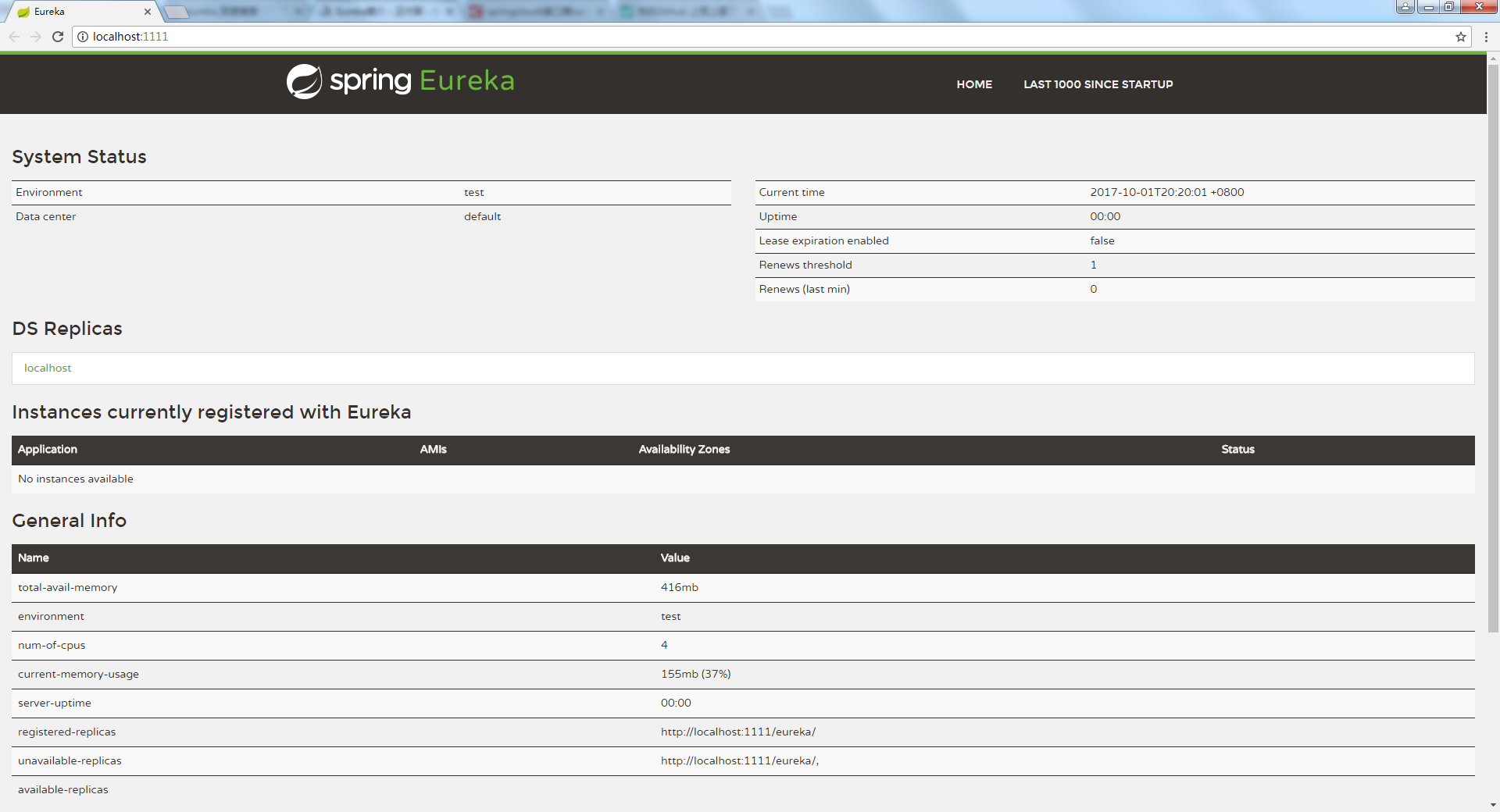
|  |
| --- |
| server.port=1111  eureka.client.register-with-eureka=false  eureka.client.fetch-registry=false  eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/ |

### Application.java

Springboot启动function

|  |
| --- |
| package com.dockerloud.regcenter.app;  import …  @SpringBootApplication  @EnableEurekaServer  public class Application {  public static void main(String[] args) {  SpringApplication.run(Application.class, args);  }  } |

使用浏览器直接访问<http://localhost:1111/>可以看到注册中心的管理界面



## 2.2 配置中心

在实际开发中，我们经常需要在不同的环境中使用不同的配置，例如在开发阶段数据库连接池需要使用dev环境的数据库。而在生产环境中我们需要将原有的配置更换为真实的数据库。对于分布式的系统来说，替换配置文件是一件非常麻烦的事，这里使用一个配置中心来为服务提供不同的配置。

### application.properties

配置中心本身就是一个服务

|  |
| --- |
| server.port=5555  spring.profiles.active=native  spring.application.name=conf |

### application-dev.properties

添加配置

|  |
| --- |
| message1=this is message one  message2=this is message two |

### Application.java

Springboot启动function

|  |
| --- |
| package com.dockerloud.regcenter.app;  import …  @SpringBootApplication  @EnableEurekaServer  public class Application {  public static void main(String[] args) {  SpringApplication.run(Application.class, args);  }  } |

## 2.3 EmployeeService

这里通过一个简单的Employee Service 模块来演示基本功能

### bootstrap.properties

声明配置中心和注册中心地址

|  |
| --- |
| spring.application.name=employee-service  server.port=3333  eureka.client.serviceUrl.defaultZone=http://192.168.1.15:1111/eureka/  spring.cloud.config.profile=dev  spring.cloud.config.uri=http://192.168.1.15:5555/ |

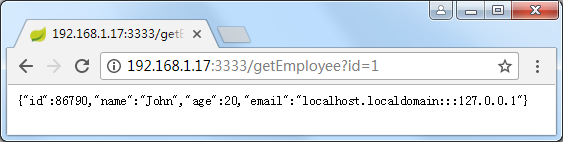
### EmployeeController

具体的功能实现

|  |
| --- |
| @RequestMapping(value = "/getEmployee" ,method = RequestMethod.GET)  public Employee getEmployee(@RequestParam int id) {  System.out.println(message1);  Employee emp = service.getEmployee(id);    try {  InetAddress addr = InetAddress.getLocalHost();  emp.setEmail(addr.getHostName()+":::"+addr.getHostAddress());  } catch (UnknownHostException e) {  e.printStackTrace();  }    return emp;  } |

### Response

通过访问绑定的端口会得到一个我们生成的json数据对象



## 2.4 EmployeeService的负载均衡

之前我们说过每个独立功能都可以由一个集群来实现,不同于以往的J2EE Server 集群的工作方式（通过完整的全部功能的拷贝来实现），微服务仅仅在需要集群的功能点上进行横向扩展。

### bootstrap.properties

负载均衡器client本身也需要注册成为一个服务（端口为部署服务器的2222）

|  |
| --- |
| spring.application.name=employee-balancer  server.port=2222  eureka.client.serviceUrl.defaultZone=http://192.168.1.15:1111/eureka/  spring.cloud.config.profile=dev  spring.cloud.config.uri=http://192.168.1.15:5555/ |

### EmployeeController

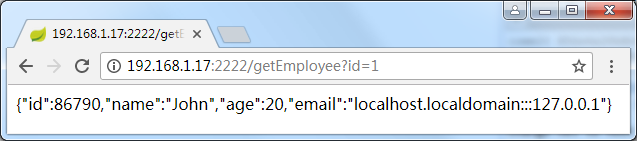
和上面的EmployeeService本身不同，负载均衡client不需要也不应该自己实现业务逻辑，而是直接访问EMPLOYEE-SERVICE来获得相应的数据

|  |
| --- |
| @RequestMapping(value = "/getEmployee", method = RequestMethod.GET)  public String getEmployee(@RequestParam int id) {  return rt.getForEntity("http://EMPLOYEE-SERVICE/getEmployee?id=" + id, String.class).getBody();  } |

### Response

通过访问负载客户端也会得到一个相同的json数据对象。

这里实际上如果存在多个EmployeeService的实例，负载均衡器会自动根据算法选择一个合适的service。



# Docker实现

Docker是一种开源的应用容器，一般用来提供轻量级的PaaS。这里基于容器来构建service的意义在于：

1. 容器可以帮助构建标准化的运行环境。

不同于以往使用puppet或者chef的情况，容器可以在不同的OS（Redhat、CentOS、Fedora、Ubuntu、Debian…）构建出完全一样的环境，将不同服务器上的系统差异降到最低。

1. 容器可以帮助构建更安全、更私密的环境。

Docker容器运行在各自的沙箱内，即使容器内的服务出现漏洞也不会波及到其他容器，在一定程度上降低服务器的风险。

1. 更轻量，比KVM的启动速度更快，资源占用更小

Docker容器本身不会在硬件级别上进行虚拟化，所以比传统的“虚拟机”技术带来的系统开销小很多。

1. 可以帮助系统在不同服务器上的持续集成和快速部署

每个容器都可以被当做无差异的，可以快速部署的工作单元，在分布式集群系统中，容器更可以非常便利的获得多个拷贝，从而对系统的瓶颈处进行横向扩展

## 3.1 Dockerfile

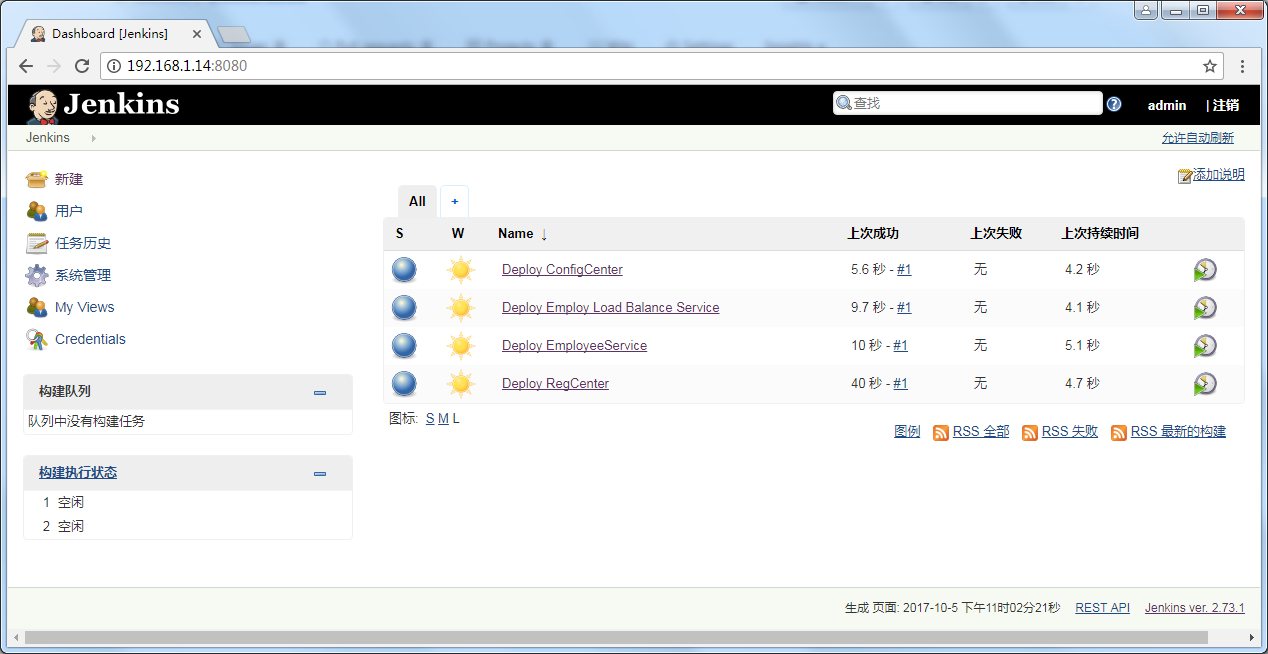
本例中我们只需要一个jdk环境，所以基于docker.io/openjdk镜像构建容器

|  |
| --- |
| FROM docker.io/openjdk  EXPOSE 1111  COPY ./release\_jars /release\_jars  ENTRYPOINT java -Djava.ext.dirs=/release\_jars com.dockerloud.regcenter.app.Application |

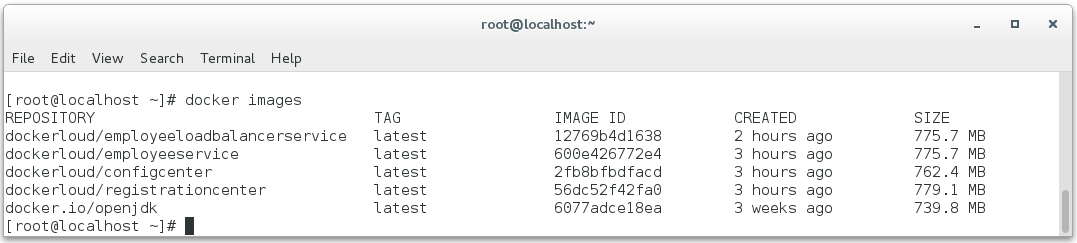
# 持续集成（基于Jenkins）

快速、方便、准确的持续集成是敏捷开发的必要条件。

本例中使用Jenkins来自动构建每个service的容器



Jenkins通过对应脚本将每个服务的应用容器化



# 快速部署（基于Ansible）

每个service（模块）在构建之初就需要提供一个用于部署自己的Playbook描述文件

例如 注册中心的描述文件

|  |
| --- |
| - name: install registration center  hosts: registrationcenter  sudo: True  tasks:  - name: install docker  shell: sudo yum -y install docker-io  - name: delete image  shell: rm -f /root/registrationcenter.tar  - name: copy image  copy: src=/root/registrationcenter.tar dest=/root/registrationcenter.tar owner=root group=root mode=777  - name: start docker service  shell: systemctl start docker  - name: stop container  shell: docker stop RegistrationCenter  ignore\_errors: yes  - name: remove container  shell: docker rm RegistrationCenter  ignore\_errors: yes  - name: remove image  shell: docker rmi dockerloud/registrationcenter  ignore\_errors: yes  - name: load image  shell: docker load < /root/registrationcenter.tar  - name: build and start container  shell: docker run --name RegistrationCenter --net=host dockerloud/registrationcenter &  - name: sleep 5s  shell: sleep 5s  ignore\_errors: yes  - name: start container  shell: docker start RegistrationCenter  ignore\_errors: yes |

Ansible会根据hosts文件中的定义将各个服务（容器）按照需要分发给不同的服务器

|  |
| --- |
| [registrationcenter]  192.168.1.15 ansible\_ssh\_user=root ansible\_ssh\_pass=root  [configcenter]  192.168.1.15 ansible\_ssh\_user=root ansible\_ssh\_pass=root  [employeeservice]  192.168.1.16 ansible\_ssh\_user=root ansible\_ssh\_pass=root  192.168.1.17 ansible\_ssh\_user=root ansible\_ssh\_pass=root  [employeeloadbalancerservice]  192.168.1.17 ansible\_ssh\_user=root ansible\_ssh\_pass=root |

在上面的例子中我们会在

服务器192.168.1.15上安装注册中心

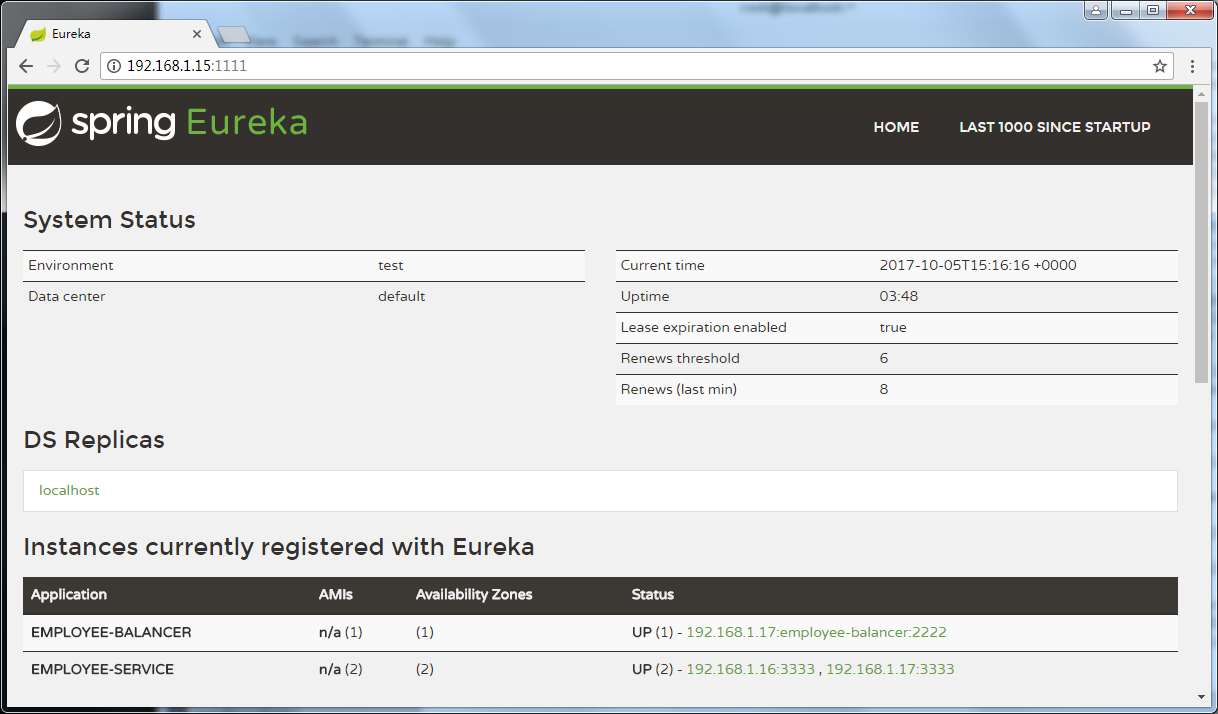
服务器192.168.1.15上安装配置中心

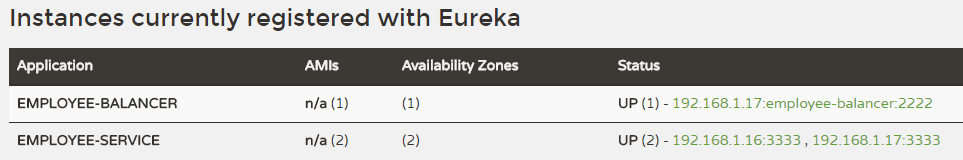
服务器192.168.1.16和192.168.1.17上安装EmployeeService

服务器192.168.1.17上安装一个EmployeeService的负载均衡客户端

可以看到注册中心上已经包含的3个活动服务

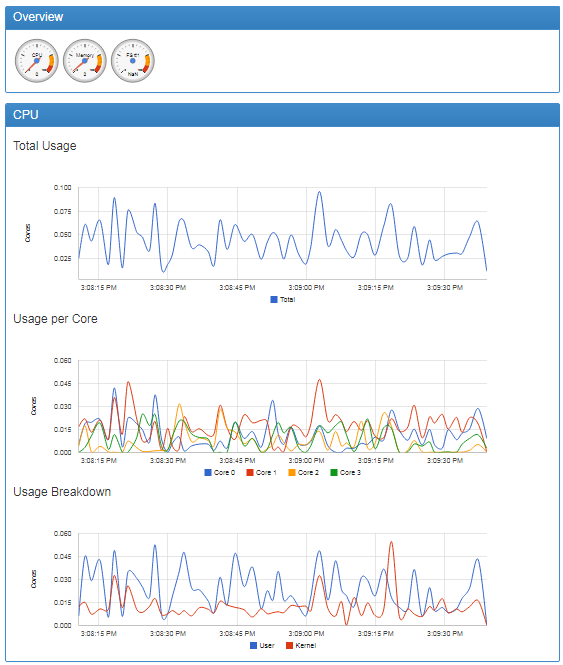
一个负载均衡客户端和两个EmployeeService





# 监控

通过cAdvisor我们还可以方便的对容器内部的运行情况和宿主机的运行情况进行监控。



# 总结

在本文讨论的范例中,我们给出了如下问题的解决方案。

## 7.1如何组织团队

因为每个服务都是一个独立的个体。可以按照一个Team一个或多个服务进行分割，只需要定义好输入输出接口。

## 7.2如何快速搭建本地开发环境

在任何分布式的大集群系统开发中，如何快速搭建本地开发环境都是一个不小的挑战。笔者曾经经历过编译一次需要一个小时的Flex项目，启动一次本地开发服务器（环境）需要45分钟的J2EE项目，在敏捷开发中这都会给持续集成造成不小的影响。下面将讨论如何避免这些问题。

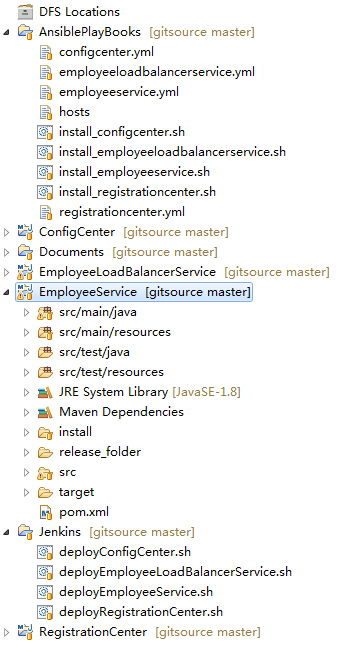
本例中我们借助于微服务方式和虚拟化技术，当一个developer/team对应一个服务时，他（们）甚至不需要再本地搭建其他的服务，一切需要的非当前开发的服务都可以由可配置的远程API提供，假设有三个独立服务A、B和C，分别由团队1、2和3来构建。

方案1，

开发用中央服务器会同时部署当前的三个服务A、B、C，各个模块的开发团队直接在本地启动自己负责的模块。这样本地开发和测试都不再需要启动所有模块，不需要改动的模块都可以由一个中央服务器提供。

方案2，

所有的服务模块都由本地启动，可以提供对应的Ansible脚本来快速部署启动这些模块，每次修改后只需要对需要修改的模块进行单独的热部署。



## 7.3如何快速测试

因为每个独立服务都是高度自治的，所以配合Junit可以方便的进行单元测试。

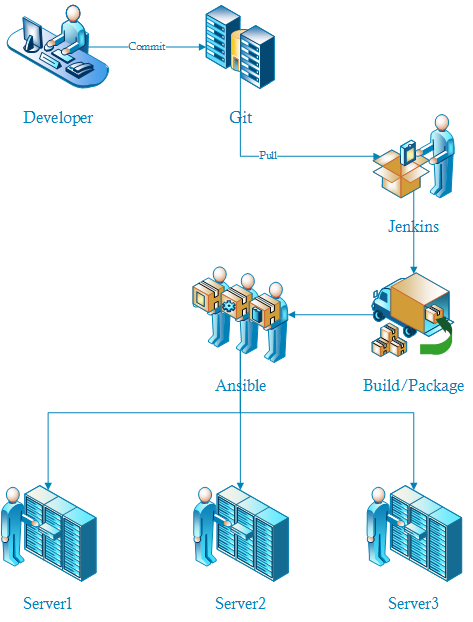
## 7.4如何保持持续集成和发布

分布式应用的一个最大的好处在于可以快速的横向扩展。

在本例中，我们使用了3个主要工具（Jenkins，Docker，Ansible）来实现系统的自动编译、自动打包、自动部署的功能。

你只需要在一台主控制机上将Git代码管理器上的代码进行编译，然后打包成Docker容器文件，分发给其它服务器。当系统进行横向扩展时，仅仅需要修改对应的服务器列表即可。当一个服务集群的工作节点增加时，它们会将自己注册给Eureka。

|  |
| --- |
| [registrationcenter]  **192.168.1.15** ansible\_ssh\_user=root ansible\_ssh\_pass=root  [configcenter]  **192.168.1.15** ansible\_ssh\_user=root ansible\_ssh\_pass=root  [employeeservice]  **192.168.1.16** ansible\_ssh\_user=root ansible\_ssh\_pass=root  **192.168.1.17** ansible\_ssh\_user=root ansible\_ssh\_pass=root  [employeeloadbalancerservice]  **192.168.1.17** ansible\_ssh\_user=root ansible\_ssh\_pass=root |



## 7.5如何最小化系统差异带来的不稳定性

Docker容器的使用可以很大程度上的降低不同server间的差异，每个模块基于同一个镜像创建的容器工作，对于应用而言，系统环境是一致的。

## 7.6如何更有效的尝试、整合其他新技术

各个服务之间使用Json或者xml格式的数据进行通信，大多数编程语言和技术都可以使用这两种格式的数据，例如Go，Python等，在系统升级或者有新技术加入时，可以对各个模块进行逐一的更新而不会对整个系统的稳定性造成冲击。

## 7.7如何快速的应对横向扩展和压力测试

早在2015年的618促销期间，京东就曾经大批量的使用Docker的容器技术为某些高负载的服务进行动态扩容，容器可以方便的负载上升时横向扩充节点，从而带来更好的性能。在促销期结束之后，这些节点都可以动态的被拆除，以降低成本。