如何利用云计算（AWS）进行百万用户压力测试

本文是记录利用云计算（AWS）对一个开源软件进行百万用户压力测试的过程，你可以认为是一篇云计算使用的基本入门文章。文章将一步步的展示作者一个人是如何利用云计算，构建，部署，测试，验证百万用户压测试这一过程的。希望能给云计算的使用者一点启发。文中所使用的所有代码 脚本，都可以在https://github.com/xiaojiaqi/fakewechat 找到，本文的wiki地址在 https://github.com/xiaojiaqi/fakewechat/wiki/Stress-Testing-in-the-Cloud。 。因为个人能力有限，文中难免有各种错误，请谅解。如果有错误 建议 欢迎邮件ppmsn2005#gmail.com

背景及需求介绍

我们为什么需要云计算?

背景

几个月以前 我开发了一个分布式的软件:fakewechat. 这是一个开源通信系统的原型，它实现了服务器的后端功能。比如它可以为用户提供可靠的通信,可以水平扩展。（它还是只是一个原型，和成熟的商业产品对比 没有意义）

现在我模拟测试了1千,1万用户时候的场景，我认为符合我设计的要求。现在我想验证我的设计在100万用户的情况下，会怎么样？没有实践是没有发言权的，所以我需要模拟一个环境测试一下。 假设我是一位网站的工程师，现在网站每天服务10万人。我想测试当每天有100万用户的情况下，网站是否还能正常服务呢？

核心需求

核心的需求 我需要很多的资源，服务器，网络，存储。我需要把程序放上去测试，搭建好环境，模拟出状态，并得出结果。

可能的解决方案：

1. 买很多电脑和网络设备，搭一套平台，搭起来，测试完再卖掉。不过简单算算买40台服务器的硬件价格，我就只能放弃了。
2. 暂时租用一批电脑和网络设备，租一段时间，为租用时间花钱，就像小时候 租录像带和VCD盘那样，我不需要买很多片子，我只要每天花几块钱，租了看就可以。

方案2 就是云计算

云计算的概念很广，这里的云计算可以特指Iaas, 你可以简单的认为，远处有一个机房，你可以在这个机房暂时租用一些机器，几个小时或者几天以后，你就可以把这些机器还给机房，然后出一点租金。

云计算商家选择

国外：AWS.

国内：腾讯云 UC ...

我选择：AWS

理由：

1. 入门门槛最低
2. 文档和资料最齐全
3. 有中文版
4. 可以全程网页操作
5. 东京区可以直连了
6. 此外在AWS 上访问各种网络资源速度非常快，不会被墙，相对国内云是个非常大的优势

缺点：

1. 必须用信用卡
2. 连接网络质量相对较差
3. 技术支持和客服交互需要英语

这里的例子是测试fakewechat, 当然如果你是网站，AWS做测试也是完全没有问题的。只是复杂度会有所提高，但是原理都一样。

目标

操作目标: 1位操作人员，在10分钟内完成项目，启动，部署和启动测试过程（不包括数据准备的过程），实现fakewechat项目百万用户压力测试。

涉及AWS的模块

在这个测试中我们主要涉及弹性云和私有网络这2项服务

AWS的基本概念：

弹性云

简称为EC2，Amazon Elastic Compute Cloud (Amazon EC2) 是一种 Web 服务，可在云中提供大小可调的计算容量。该服务旨在降低开发人员进行网络规模级云计算的难度。

安全私有云

简称为VPC，Amazon Virtual Private Cloud (Amazon VPC) 允许您在 Amazon Web Services (AWS) 云中预置一个逻辑隔离分区，让您在自己定义的虚拟网络中启动 AWS 资源。您可以完全掌控您的虚拟联网环境，包括选择自有的 IP 地址范围、创建子网，以及配置路由表和网关。

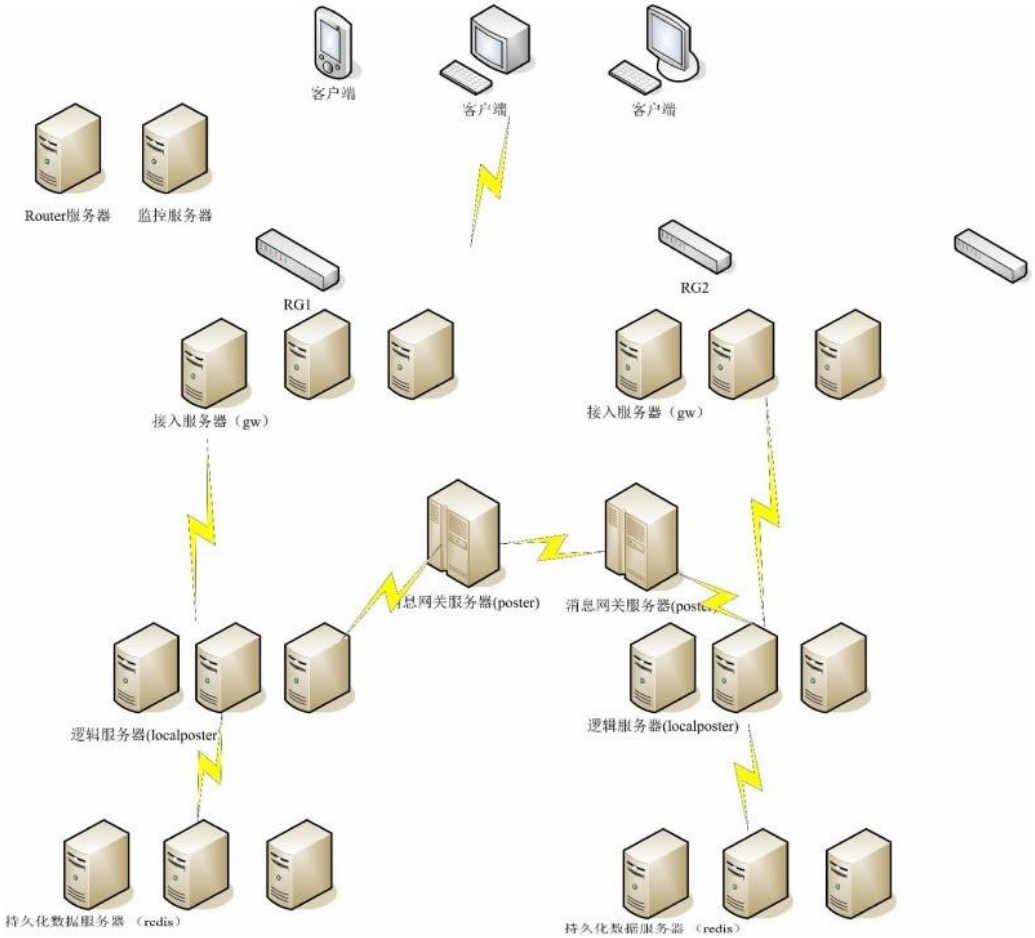
你可以这样简单理解 Ec2 就是虚拟机，它是你的计算资源。你可以启动很多虚拟机，然后在上面安装各种软件，提供服务。而VPC 就是对你网络拓扑的描述，它描述了 你的网络是怎么构成的，外部用什么DNS，IP来访问,有那些子网，网关，路由。这样Ec2的主机拥有了私有或者公有的IP，就可以为用户服务了。

AWS的概念非常复杂，要全部理解不是这样一篇小小的文章可以解决。

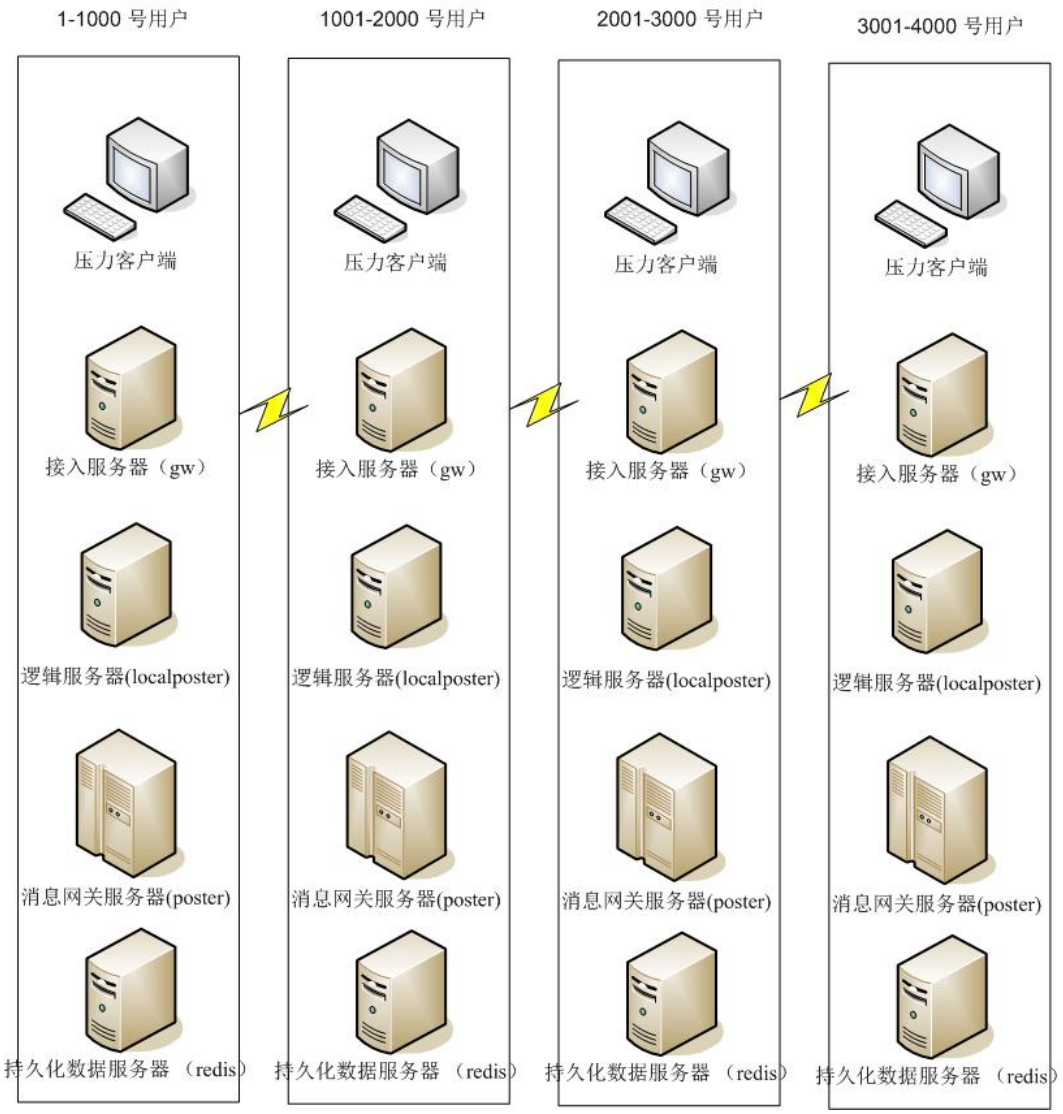
系统架构

首先我们需要考虑如何设计一个水平扩展的架构。

下图是系统的架构图

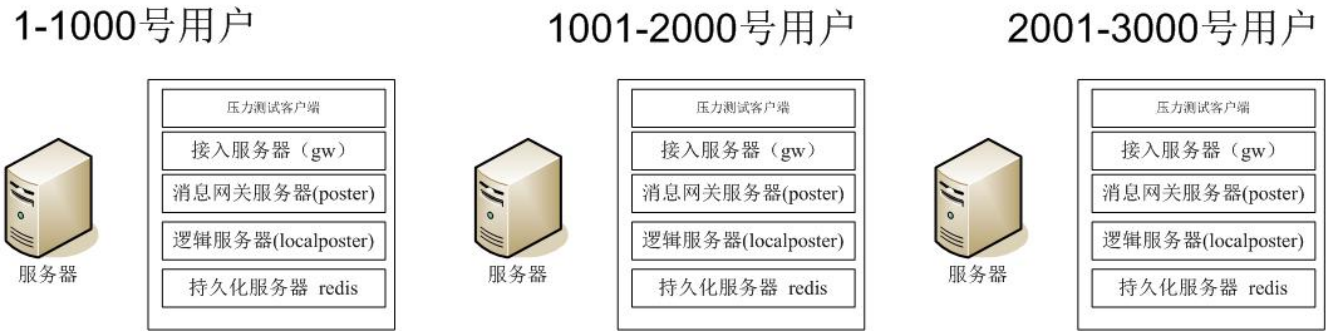


我们可以看到架构在设计的时候已经考虑到了水平扩展的问题。所以，我们可以将系统Cell化。具体设计思想可见文档，简单的说我们可以把所有需要的服务划分为一个Cell,一个Cell是可以独立的为用户提供服务的。



注意 简单的按照用户id号划分集群，会有非常大的局限性，实际生产环境中不能简单的使用。目前项目还没有实现实现动态的调度和配置，以后会实现这个要点。

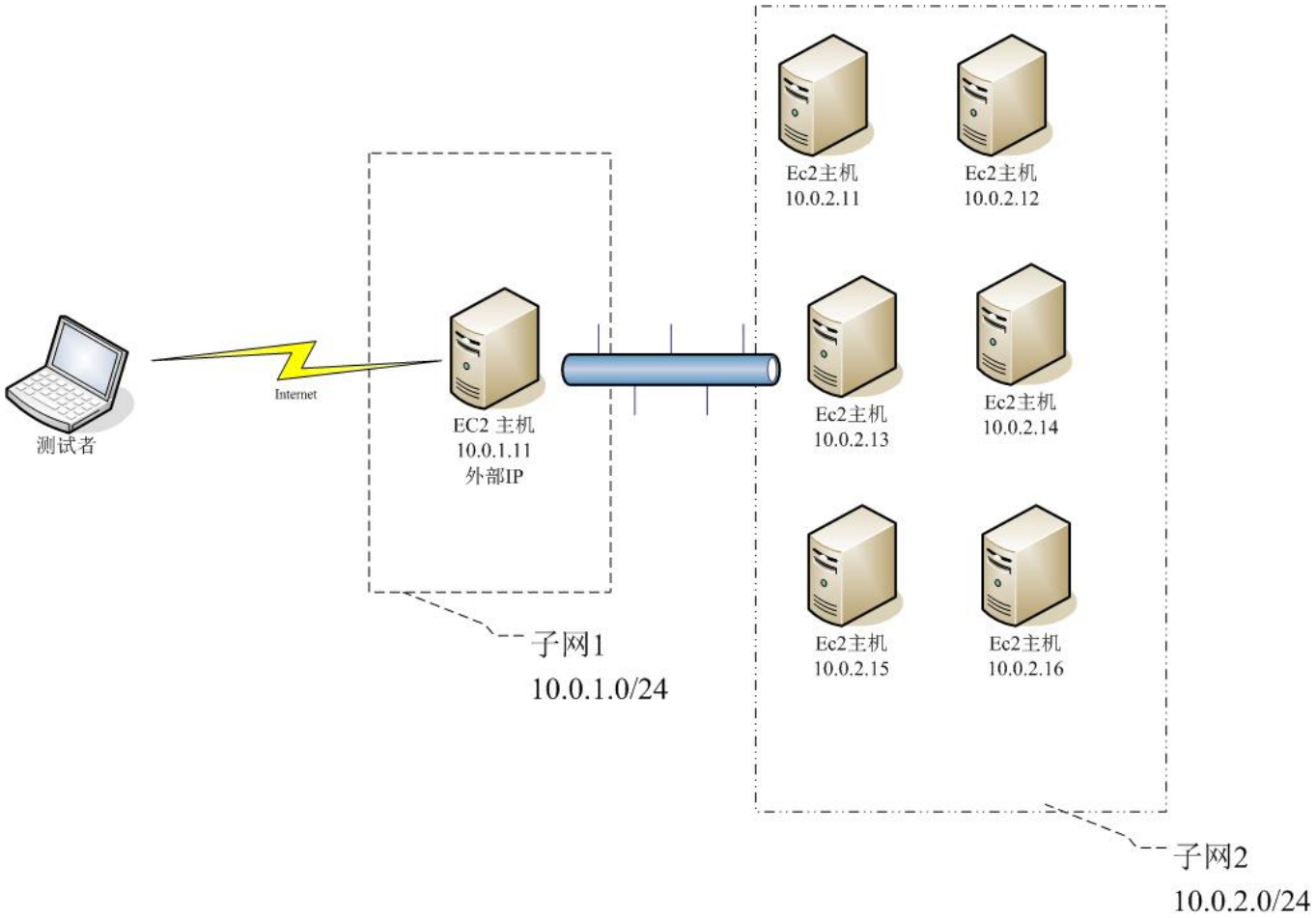
为了进一步节省成本 系统还可以被优化成这个样子



一个虚拟机模拟一个集群，完成集群里所有的业务。（需要注意 这样的测试大部分流量并没有走真正的网络，和真实网络情况有差异。) 这样可以最大程度的节省成本。

如果在真正的生产环境，我们可以把各个服务进程分配到各个独立的服务器上，提高服务的能力。当用户人数降低的时候，我们又可以简单地把服务进程收缩到少数的服务器上，节省支出。真正的做到 收发自如。（本文离真正的收发自如还有10万多公里，剩下的路还需读者自行钻研）

画一下我们的测试需要用到网络拓扑图



这个拓扑比较简单，我简单的描述一下里面各个节点和作用

测试者： 测试者就是作者，作者利用internet 连接进AWS集群,完成百万用户的测试 子网： 在这里我将测试划分为2个子网 10.0.1/24 和10.0.2.0/24 .其中10.0.1.0/24 这个子网有互联网的接入能力，也只放置了一台主机10.0.1.11. 因为我需要在这台主机下载各种需要的软件包。 而10.0.2.0/24 里面可以启动非常多的主机，比如从10.0.2.11-10.0.2.200. 它们可以和主机10.0.1.11互通。它们可以每台服务器模拟一个集群，可以几台服务器模拟一个集群。

注意，一般来说我们会把网段做更好的规划。比如10.0.2.0/24 这个网段作为接入服务器的网段。放置100台计算服务器。10.0.4.0/24 这个网段作为数据库的网段，放置数据库，网络之间连接使用专门的网络设备。我这个测试比较简单，所有的东西都在同一个子网内，服务器之间的流量也不是非常大。

这一切和真正的实际环境，业务模型都有很大关系。 总之这里是个简单版本

初探AWS

准备

开始AWS之旅 如果网络不好的，也许需要翻墙。

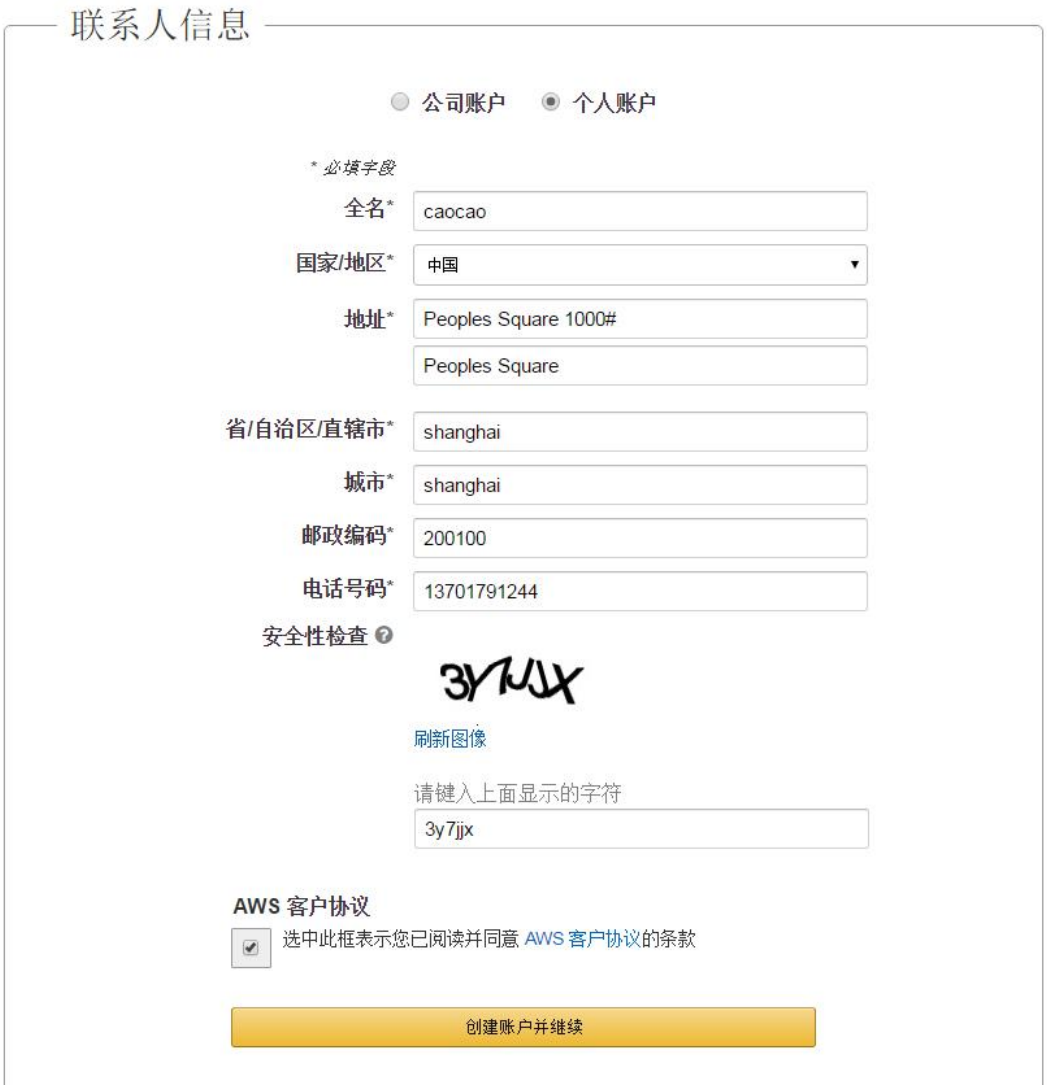
1. 首先准备账号
2. 准备信用卡

访问 http://aws.amazon.com/cn/

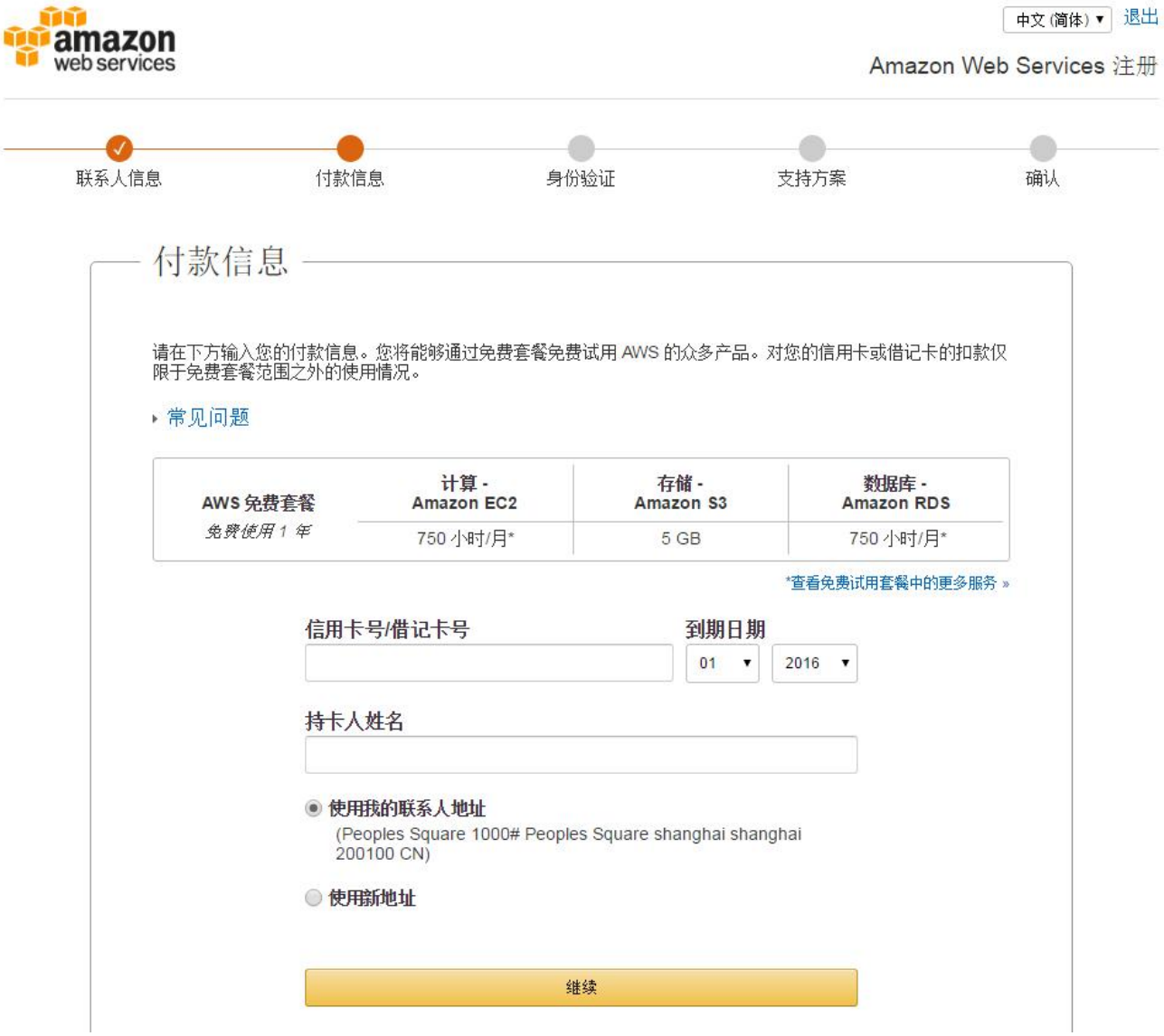
注册帐号

创建用户

输入用户信息


联系人信息
公司账户
个人账户
* 必填字段
全名*
caocao
国家/地区*
中国
地址*
Peoples Square 1000#
Peoples Square
省/自治区/直辖市*
shanghai
城市*
shanghai
邮政编码*
200100
电话号码*
13701791244
安全性检查
刷新图像
请键入上面显示的字符
3y7jjx
AWS 客户协议
选中此框表示您已阅读并同意 AWS 客户协议的条款
创建账户并继续

输入信用卡



我的账号是几年以前注册的，我记得当时会有电话回拨，输入验证码，然后信用卡会被扣除1美金。测试你的信用卡是否正常。这1美金不会真正的扣除，大概是这样，完成整个注册过程。如有有问题 请帮我修正。

软件部分 你需要准备以下的几样东西

考虑到windows的普及性，我windows平台上完成测试。 Mac os,Linux 的用户使用会在使用密钥的时候稍有不同，但问题不大。

1. windows电脑一台
2. linux 电脑一台（主要是转换密钥用)
3. SSH 客户端软件 这里我使用 SecureCRT

简单使用

使用AWS

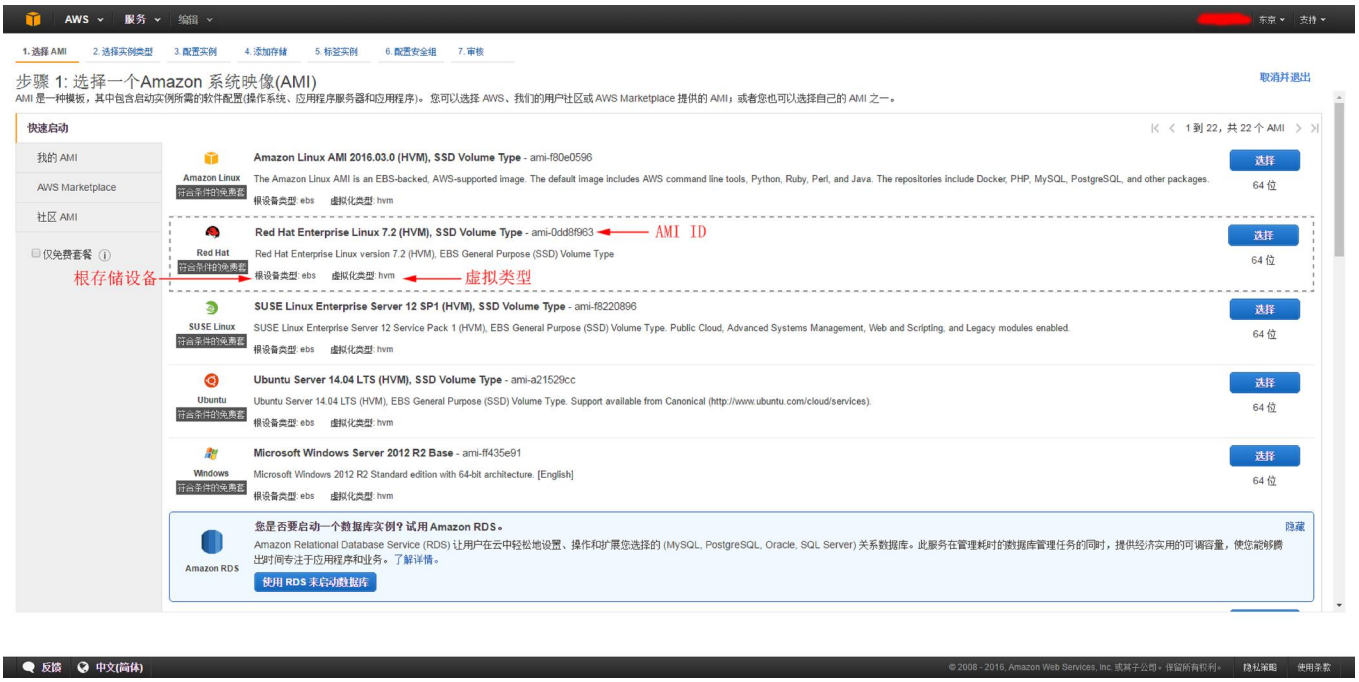
浏览一下首页，右上端显示我正在使用东京区的网页控制台。 主服务区里面罗列了，AWS提供的所有服务。目前我们只需要使用2种 弹性云，安全私有云。

点击右上端的"东京", 会罗列出所有的区域，你可以选择不同的区域跳转

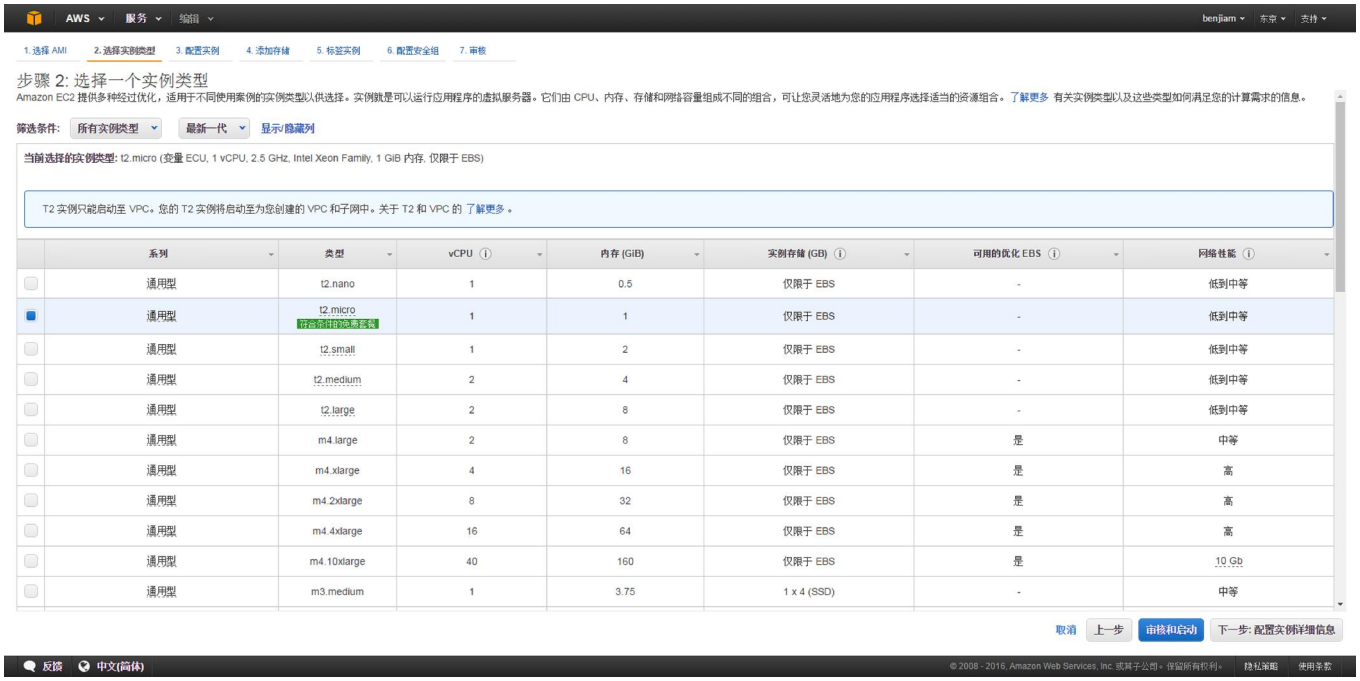
点击首页的弹性云，可以看到弹性云的主页面。

点击启动实例就可以启动开启一个弹性云的主机。

第一步 选择一个AMI，AMI就是磁盘镜像，和Ghost很类似。 AMI分为我的AMI ， AMI MarketPlace, 社区AMI, AMI标签里有 AMI ID， 根存储设备， 虚拟类型这几个参数。 注意 某些AMI是要收钱的。



第二步， 选择一个实例类型，实例的差别主要体现在CPU,内存，网络



第三步，主要是配置网络，如子网，是否拥有外网IP等等

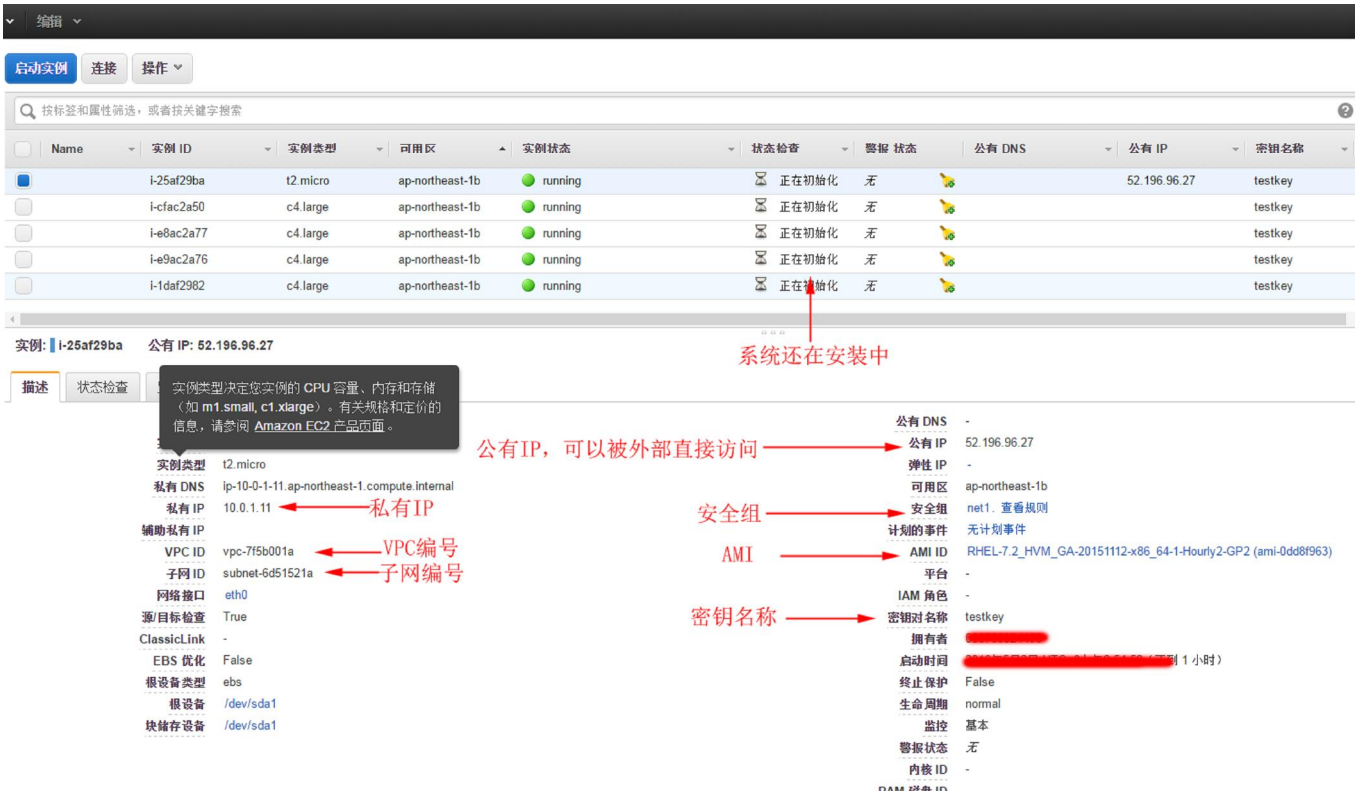
第四步，选择存储，也就是为主机选择一块"硬盘"

第五步 设置标签实例

第六步 设置安全组，也就是设置网络端口

最后运行实例

运行以后大概就是这样



AWS的基本概念

主机部分：

- Region: 分区，简单的说 就是这个机房在那个位置。
- 实例类型: 就是主机的配置，它决定有多少CPU 内存 网络设备
- AMI： 镜像，你可以想象为一个操作系统安装盘。它有一个唯一ID，同时它决定了使用哪种虚拟类型。
- 虚拟类型： hvm, pv. 两种的区别就是全虚拟和半虚拟，在性能上有差别。
- 存储设备： 一般分ESB和SSD.ESB是弹性块存储，SSD 应该就是我们常见的SSD.可以根据业务类型做选择。
- 密钥: 你通过网络连接主机时候的密钥，主要保护你主机的安全。

网络部分： - VPC： VPC就是一套网络配置的总称 - IP: 一台主机拥有的IP，有内部也有外部 - 子网：就是网络子网 - 安全组：可以认为是网络安全配置，决定了那些数据包可以通行，可简单的认为就是防火墙。

注意点：

主机选择：

主机类型的选择： AWS 将主机分为了 通用,计算，内存。

CPU 核数从1 到 32...

某些主机CPU有积分, 如果你使用一个CPU很弱的主机，跑一个重CPU的业务，那么它将迅速的把这个主机的CPU主机资源消耗殆尽，然后进入非常低速的状态，这时使用top检测系统，表现的状态就是st非常高。aws很精明，它将计算资源，内存资源，网络资源，存储资源都做了非常精确的划分。所以你需要谨慎的选择你的主机类型，不要让小马拉大车，同时也要避免大马拉小车的浪费。

解决办法: 在软件设计的时候，应该有各种参数控制程序对资源的使用，比如线程并发数，队列长度，是否可以暂停请求等等。你可以通过调节参数让主机的负载能力和cpu 网络 的消耗符合要求。

Region：

各个分区的主机价格是不一样的！但是离你太远的话，网络又是问题。所以你需要做一个折中。

AMI：

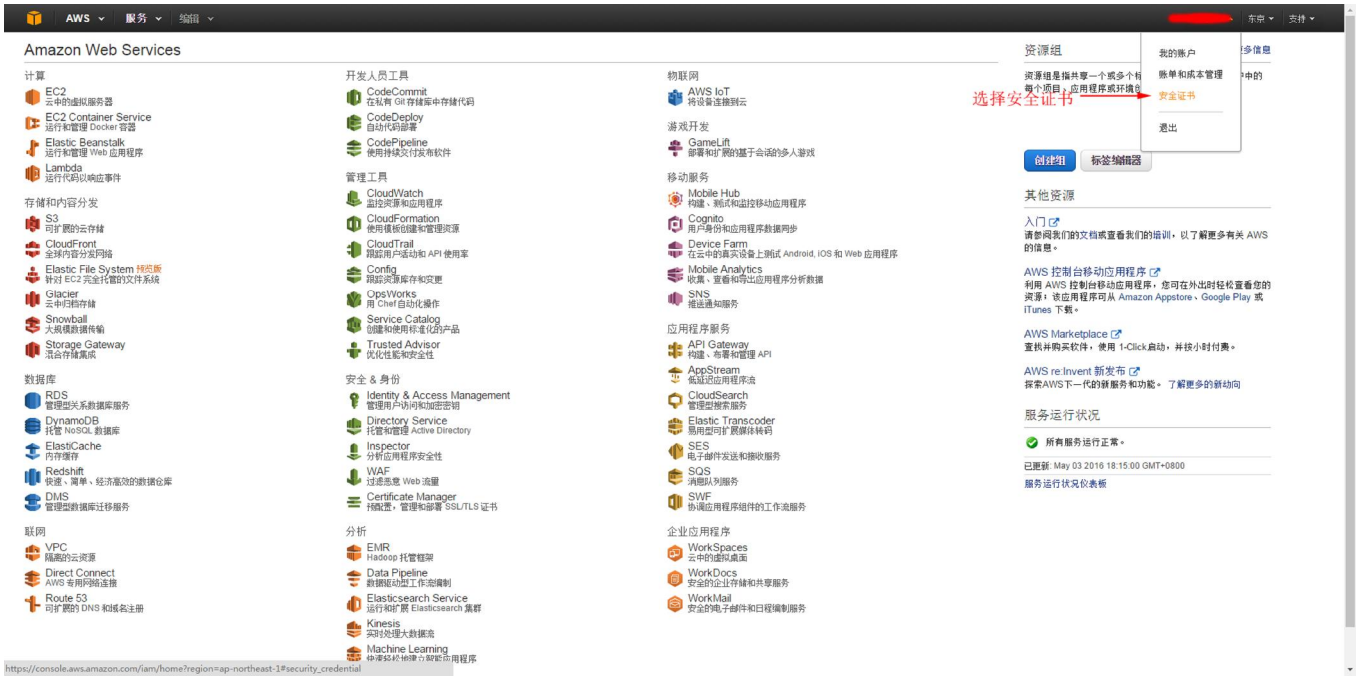
有一些AMI是要花钱的，而且价格不菲。 比如t2.Micro +RHEL 每小时是0.08$ 而纯的T2.Micro + linux是0.02$. 如果你在t2.Micro上使用1小时的RHEL, 就可以使用等额4小时的Centos. 作为工程师必须节约资源！ （当然默认的RHEL的确很好用）

如何搭建AWS系统

刚才我们已经简单使用了一下aws，现在开始一些准备

获取证书

从首页进入安全证书



AWS有很多种安全方法，我们选择第三个访问密钥

然后你可以看到你现有的安全凭证

选择生成一个安全证书，妥善的保管它！

注意！

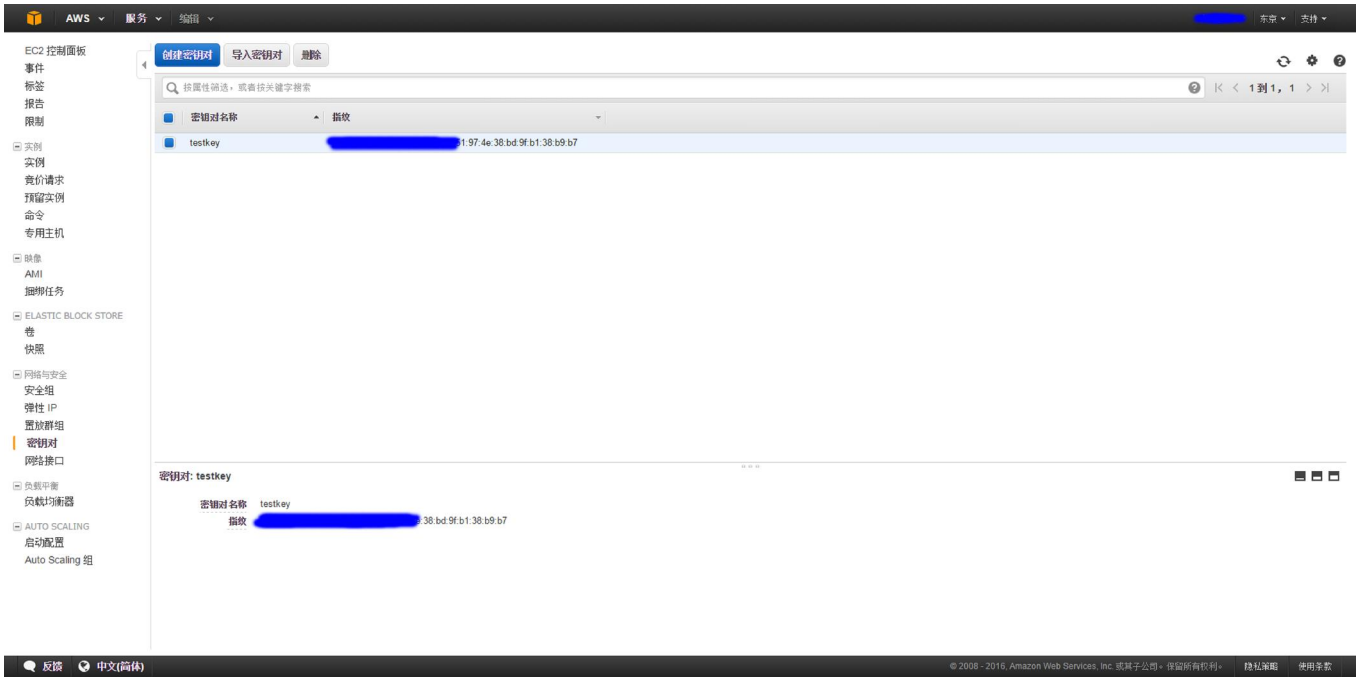
安全证书很重要 很重要。网上有无数人因为证书泄漏，被人恶意的挖矿，爬虫，攻击等等，白白花费了aws的费用，。所以一定要保管好，要保管好,要保管好! 重要的事情说3遍。

如果发现证书丢失，赶紧上网注销证书！

申请一个密钥对

点击EC2面板，选择密钥对

生成一个密钥，给它取一个名字



下载密钥

这个名字很重要，编码的时候会用到

用linux主机转换这个密钥 可以参考 http://blog.csdn.net/iamshaofa/article/details/7878346

用SecureCRT登录, 需要根据xxx.pem生成一个公钥文件xxx.pem.pub。不过生成*.pub还是需要linux下进行：

$ chmod 700 xxx.pem

$ ssh-keygen -y -f xxx.pem >xxx.pem.pub

就是说，先改一下*.pem的权限，然后再用ssh-keygen制作 pub文件

证书和密钥的区别：

证书可以访问你的服务，权利非常大，API开发需要利用证书，对你的请求进行签名。是你用户身份的标识 密钥对是弹性云计算中主机访问使用时候使用的密钥，没有它你无法连接上新创建的主机。

测试过程

一个分布式系统，从0开始构建，我是按照下面的步骤进行

1. 搭建主机资源，为主机安装基本的操作系统，以及基本的网络设置如IP,路由

2. 配置基本的网络资源，划分子网，带宽，网络端口开放
3. 主机上安装基本的软件，打开防火墙端口，修改内核参数
4. 编译所有需要用的软件，准备所需要的软件包，也包括依赖的软件包
5. 为所有的主机准备启动，停止，监控所需要的配置文件，脚本
6. 为所有的主机部署所需要的各种软件，保证安装正确，所有的配置文件和脚本安装正确
7. 将预制数据准备进系统
8. 启动监控系统，
9. 开启压力测试程序，在监控系统对进度做监控
10. 压力测试完成后，停止所有服务
11. 检测压力测试的数据结果，看结果是否符合要求
12. 销毁所有主机和网络资源
13. 检测压力测试中的系统记录数据，画出对应的状态变化曲线。

如果是网站测试的话，我认为可以把软件安装这部分 改为为安装http服务器，tomcat，数据库，Php/Java 程序，准备Memcached/redis, LVS代理 这样的业务。实质只是运行的软件不一样，但是流程上大同小异。

实现的过程

1.搭建主机：

我们可以在网页里一步步的点，如果是这样做，云计算就完全没有存在的必要，所以我们需要使用AWS提供的API接口，创建我们的主机，VPC网络，最后利用API销毁这些主机。

2.编译所有的软件包：

一般公司使用jenkins对软件包进行编译，打包，单元测试这些过程。如果是持续开发的环境，这样的系统是必备的。目前从0开始，我把过程简化成为一个shell脚本，编译完成所有的软件。

3.为所有的主机准备脚本：

写了一个简单的脚本，可以为系统准备随机数据，为每台服务器准备它们需要的启动脚本，停止脚本，redis 启动，清空等一系列脚本。

4.为所有的主机安装软件和脚本：

很明显我们需要一个开源自动化配置管理工具，选择非常多Saltstack, cfeng, puppet 以及ansible. 在不考虑语言本身的差异，这几个软件差异并不大，考虑到目前的实际情况，我认为在这个项目中ansible有一个比较好的优势：使用ssh 无需为每台服务器安装相应的客户端，这非常方便。并且执行的批处理也很简单。所以我选择ansible 作为部署软件。

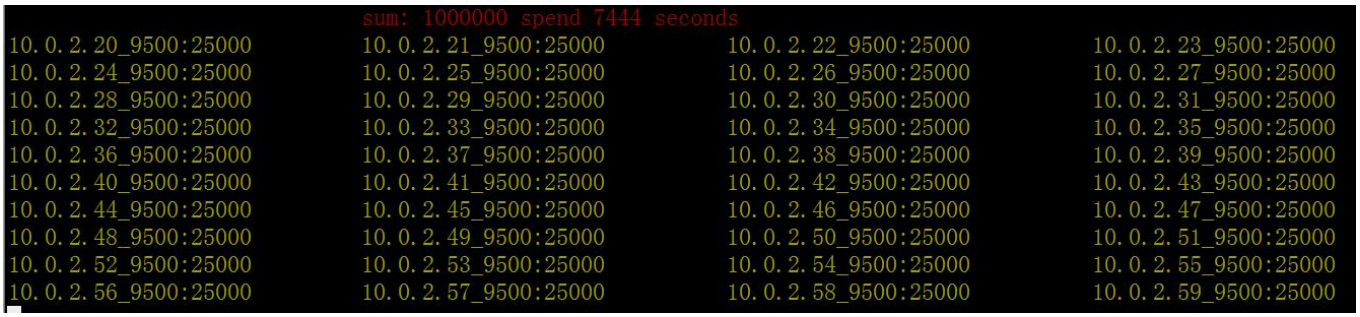
5.启动和停止服务：

用ansible批处理的形式完成

6.监控:

需要一个监控的服务，可以对测试的情况做及时的捕捉，掌握状态。已经提供了部分zabbix系统的脚

本，但是完整实现还需一些代码，所以目前写了直接写屏软件，用它记录各个服务器压力测试的状态，类似股票市场的监控图。因为服务器的各个时间点状态，都有记录所以可以在后期做测试。 它看上去是这样的



7.事后分析:

已经有专门的脚本支持

整个系统的数据准备，软件编译，分发，部署，测试命令分发，我都在一台主机10.0.1.11上完成，这是唯一可以连接internet的服务器。这样其他所有的主机都无需连接Internet，可以节省网络费用。fakewechat使用是Go语言编写，所以不存在第三方库依赖问题，非常适合快速部署。

题外话：

从发布测试成熟度来看，我认为可以分为几个阶段， 第一个阶段就是茹毛饮血hardcode的脚本时代（就像我目前做的），第二个就是 自动脚本时代， 第三代 就是service时代，第四代则是智能管理时代。我觉得，在云时代，企业产品云化需要这样的服务。

构建虚拟机环境

脚本是一年多以前写的，语言python, 类库boto2， 目前boto版本是3.

位置：https://github.com/xiaojiaqi/fakewechat/blob/master/aws/create_vpc.py

只需要用修改几个参数就可以完成

脚本很简单，它大概做了一下几件事情

1. 创建一个VPC
2. 创建一个Internet的网关
3. 将创建的网关加入VPC
4. 创建为网关创建一个路由
5. 创建4个子网分别是10.0.1.0/24,10.0.2.0/24,10.0.3.0/24,10.0.4.0/24
6. 创建安全组，让所有的主机可以通信
7. 创建一台主机10.0.1.11
8. 创建多台主机 IP从10.0.2.20开始

运行文件即可，大概1，2分钟。几十台服务器就已经安装，配置完成。IP 网关也配置完成。 步骤 1，2 完成

```
E:\fakewechat\src\github.com\fakewechat\aws>python create_vpc.py
list route table
begin create vpc
vpc  vpc-57ecb632  was created
gw  igw-e4de5281  was created
attach gateway success
list route table
rtb-fd090498
new route table id is  rtb-fd090498
create route
begin create subnet
subnet  subnet-2552087c  was created
subnet  subnet-2452087d  was created
subnet  subnet-2752087e  was created
subnet  subnet-2652087f  was created
begin create security group
begin add authorize to security
 security group  sg-0cc14768  was created
 security group  sg-0dc14769  was created
 security group  sg-0ec1476a  was created
 security group  sg-0fc1476b  was created
begin create ec2 instance
Reservation:r-1cf5d4be 10.0.1.11
Reservation:r-97f4d535 10.0.2.20
Reservation:r-1df5d4bf 10.0.2.21
```

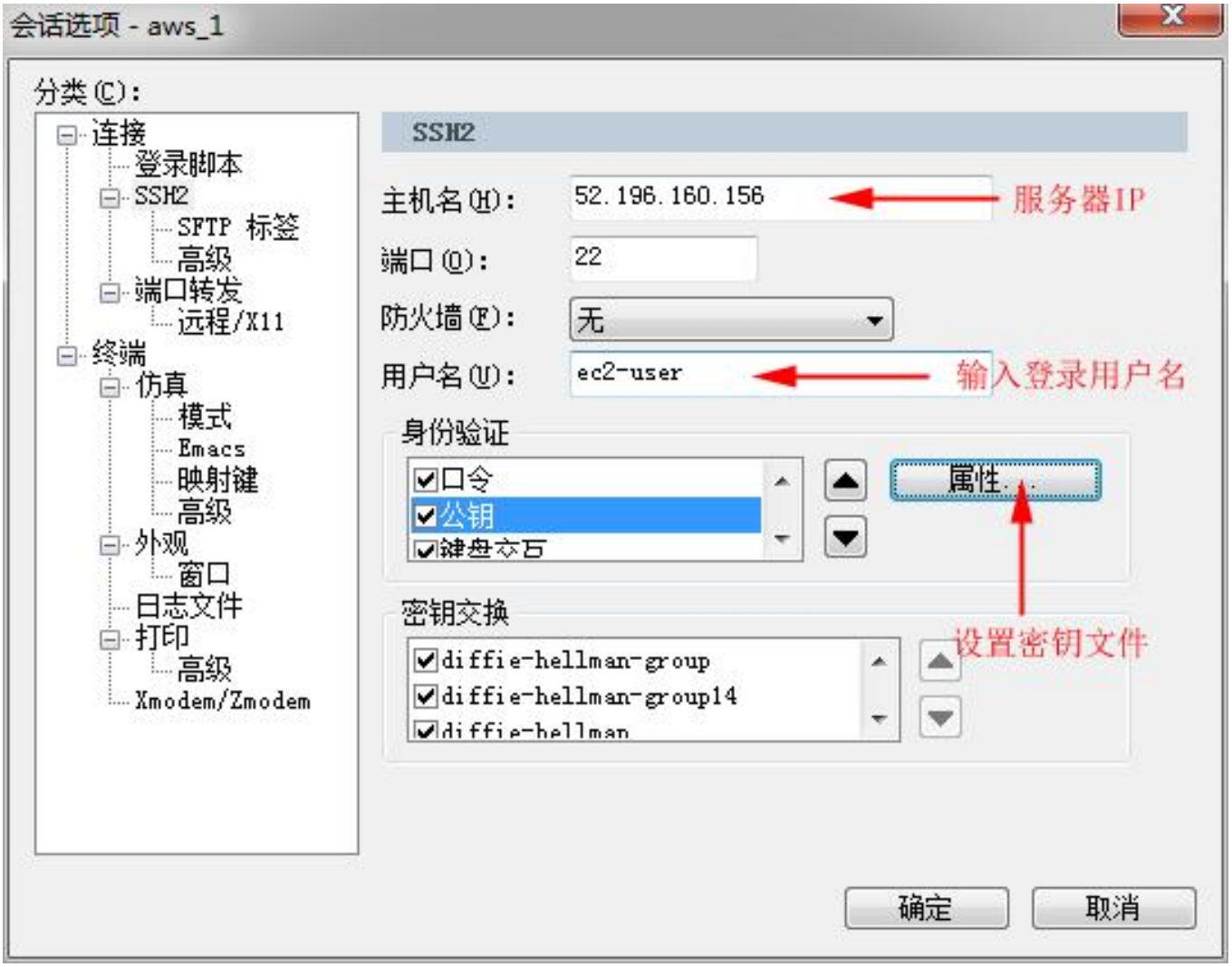
构建软件 安装

现在使用 SecureCRT 连接服务器

SecureCRT的配置过程是这样的

创建一个连接

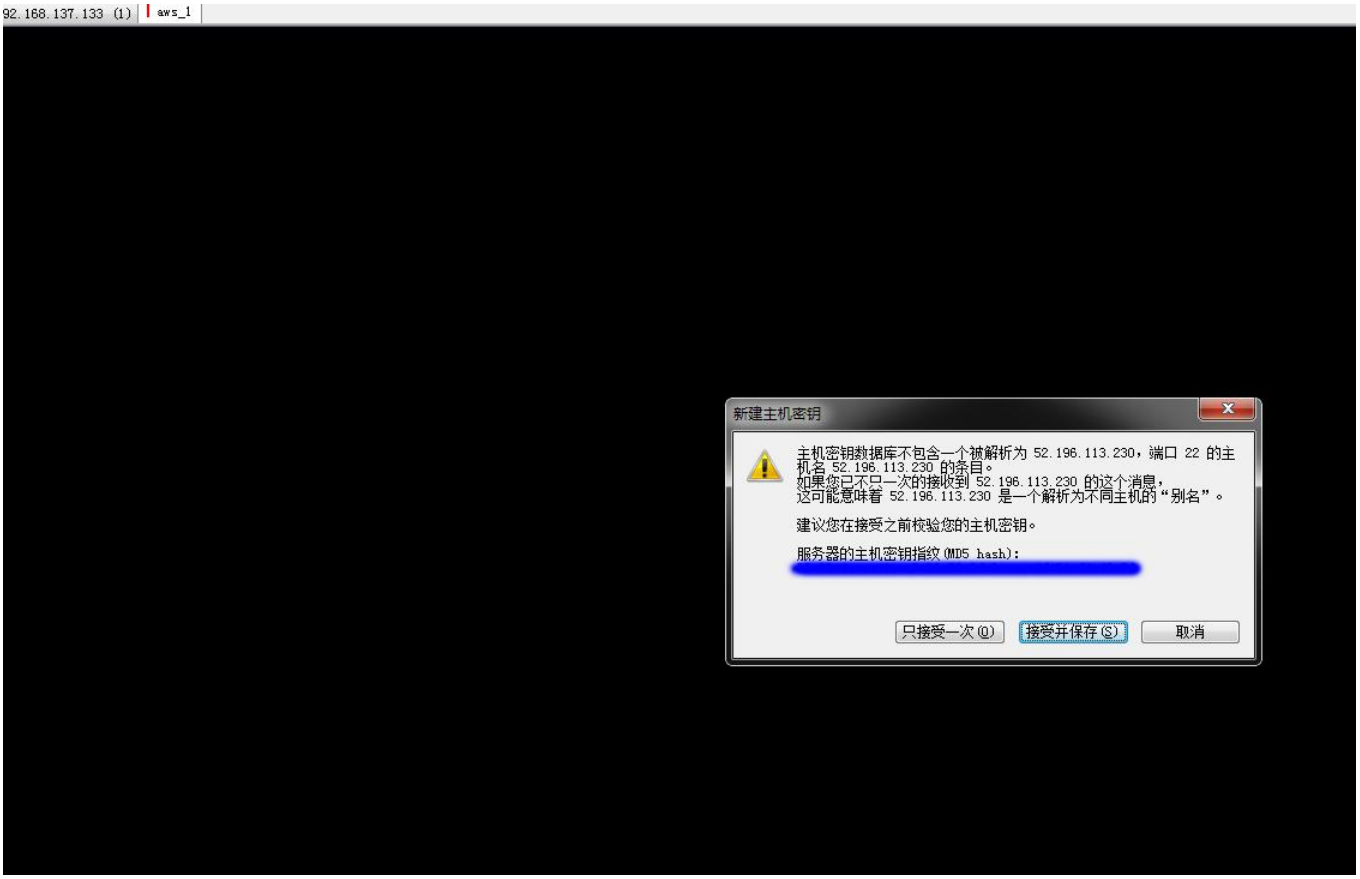
将主机的IP 设置进去，（服务器IP可以从控制台获得，也可以用DNS管理）



如果是RHEL 则默认登录用户是ec2-user, 大部分Linux是root

设置公钥文件，选择 刚才转换好的文件

接受密钥，完成连接

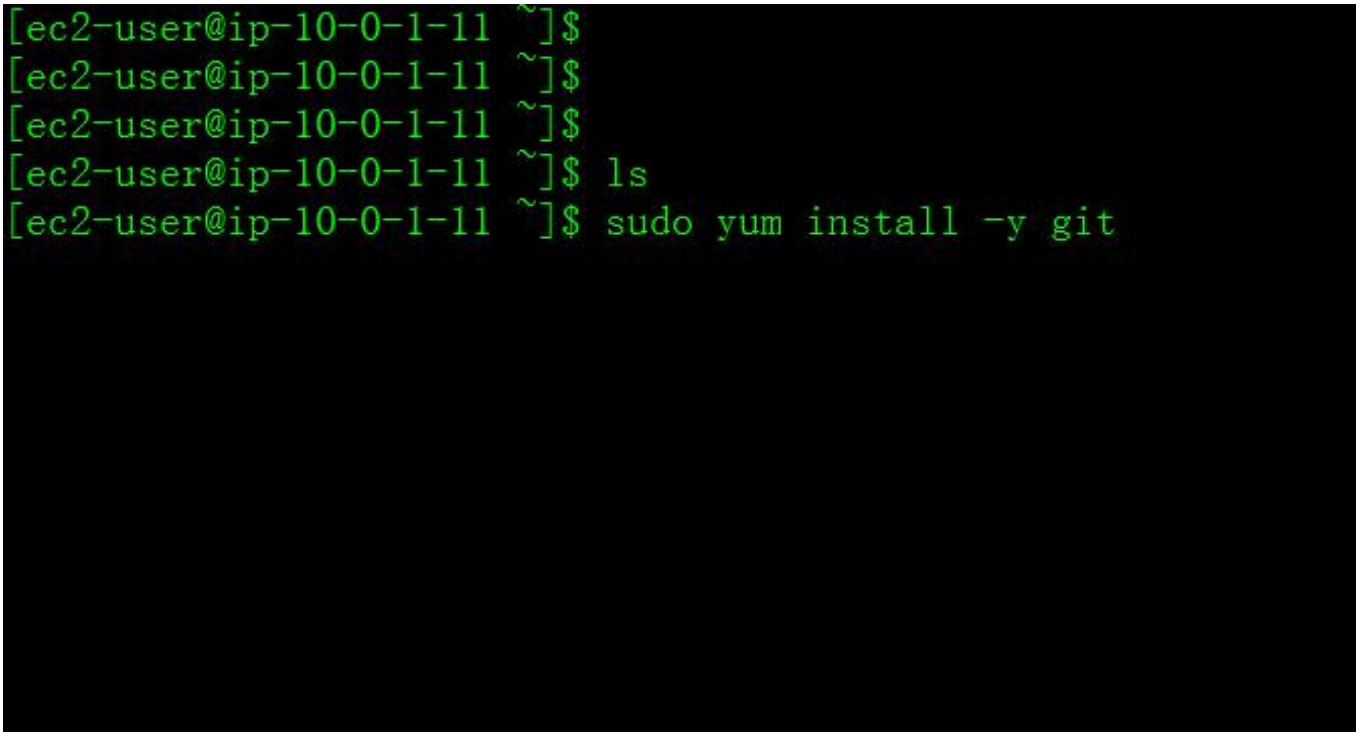


实践过程

我们假设最先登录的主机为10.0.1.11,它有外部IP. 但是我们简称它为10.0.1.11主机，下面的操作都是在这台主机上完成的

系统部署

1. 首先安装git工具



1. 在当前目录创建 gopath/src/github.com, 进入$HOME/gopath/src/github.com/ 克隆整个项目

```
  Verifying  : perl-File-Temp-0.23.01-3.el7.noarch                              13/32
  Verifying  : 1:perl-Pod-Simple-3.28-4.el7.noarch                              14/32
  Verifying  : perl-Time-Local-1.2300-2.el7.noarch                              15/32
  Verifying  : perl-Pod-Perldoc-3.20-4.el7.noarch                               16/32
  Verifying  : perl-Git-1.8.3.1-6.el7_2.1.noarch                                17/32
  Verifying  : perl-Carp-1.26-244.el7.noarch                                    18/32
  Verifying  : 1:perl-Error-0.17020-2.el7.noarch                                19/32
  Verifying  : 4:perl-Time-HiRes-1.9725-3.el7.x86_64                            20/32
  Verifying  : perl-Scalar-List-Utils-1.27-248.el7.x86_64                       21/32
  Verifying  : libgnome-keyring-3.8.0-3.el7.x86_64                              22/32
  Verifying  : perl-Pod-Usage-1.63-3.el7.noarch                                 23/32
  Verifying  : perl-Encode-2.51-7.el7.x86_64                                    24/32
  Verifying  : perl-podlators-2.5.1-3.el7.noarch                                25/32
  Verifying  : perl-Getopt-Long-2.40-2.el7.noarch                               26/32
  Verifying  : perl-File-Path-2.09-2.el7.noarch                                 27/32
  Verifying  : perl-threads-1.87-4.el7.x86_64                                   28/32
  Verifying  : perl-Socket-2.010-3.el7.x86_64                                   29/32
  Verifying  : perl-Filter-1.49-3.el7.x86_64                                    30/32
  Verifying  : perl-Text-ParseWords-3.29-4.el7.noarch                           31/32
  Verifying  : git-1.8.3.1-6.el7_2.1.x86_64                                     32/32

Installed:
  git.x86_64 0:1.8.3.1-6.el7_2.1

Dependency Installed:
  libgnome-keyring.x86_64 0:3.8.0-3.el7      perl.x86_64 4:5.16.3-286.el7           perl-Carp.noarch 0:1.26-244.el7
  perl-Encode.x86_64 0:2.51-7.el7            perl-Error.noarch 1:0.17020-2.el7      perl-Exporter.noarch 0:5.68-3.el7
  perl-File-Path.noarch 0:2.09-2.el7         perl-File-Temp.noarch 0:0.23.01-3.el7  perl-Filter.x86_64 0:1.49-3.el7
  perl-Getopt-Long.noarch 0:2.40-2.el7       perl-Git.noarch 0:1.8.3.1-6.el7_2.1    perl-HTTP-Tiny.noarch 0:0.033-3.el7
  perl-PathTools.x86_64 0:3.40-5.el7         perl-Pod-Escapes.noarch 1:1.04-286.el7 perl-Pod-Perldoc.noarch 0:3.20-4.el7
  perl-Pod-Simple.noarch 1:3.28-4.el7        perl-Pod-Usage.noarch 0:1.63-3.el7     perl-Scalar-List-Utils.x86_64 0:1.27-248.el7
  perl-Socket.x86_64 0:2.010-3.el7           perl-Storable.x86_64 0:2.45-3.el7      perl-TermReadKey.x86_64 0:2.30-20.el7
  perl-Text-ParseWords.noarch 0:3.29-4.el7   perl-Time-HiRes.x86_64 4:1.9725-3.el7  perl-Time-Local.noarch 0:1.2300-2.el7
  perl-constant.noarch 0:1.27-2.el7          perl-libs.x86_64 4:5.16.3-286.el7      perl-macros.x86_64 4:5.16.3-286.el7
  perl-parent.noarch 1:0.225-244.el7         perl-podlators.noarch 0:2.5.1-3.el7    perl-threads.x86_64 0:1.87-4.el7
  perl-threads-shared.x86_64 0:1.43-6.el7

Complete!
[ec2-user@ip-10-0-1-11 ~]$ mkdir -p gopath/src/github.com
[ec2-user@ip-10-0-1-11 ~]$ cd gopath/src/github.com/
[ec2-user@ip-10-0-1-11 github.com]$ git clone https://github.com/xiaojiaqi/fakewechat.git
```

1. 进入$HOME/gopath/src/github.com/fakewechat/package 运行install.sh 脚本,脚本主要做了几件事情
   - 利用yum 安装了所需要的gcc 等工具
   - 下载golang 的安装包，解压，并设置环境
   - 编译项目，生成所有的可执行文件
   - 编译redis服务器，生成需要使用的redis-server 和redis-cli可执行文件
   - 导入RPM-GPG-KEY-EPEL-7, 并安装ansible包
2. 修改/etc/ansible/ansible.cfg 文件 大概38行左右，去掉这个注释。 这样ansible连上新的主机，就不需要确认了。

```
28 #
29 # smart - gather by default, but don't regather if already gathered
30 # implicit - gather by default, turn off with gather_facts: False
31 # explicit - do not gather by default, must say gather_facts: True
32 #gathering = implicit
33
34 # additional paths to search for roles in, colon separated
35 #roles_path    = /etc/ansible/roles
36
37 # uncommon this to disable SSH key host checking
38 host_key_checking = False      <-- 去掉这行的注释
39
40 # change the default callback
41 #stdout_callback = skippy
42 # enable additional callbacks
43 #callback_whitelist = timer, mail
44
```

1. 进入进入$HOME/gopath/src/github.com/fakewechat/bin 运行deploy.sh 脚本主要完成了下面的事情
   - 运行listcm.py 脚本，为每一台主机生成它们需要的配置文件和启动脚本，关闭脚本，数据载

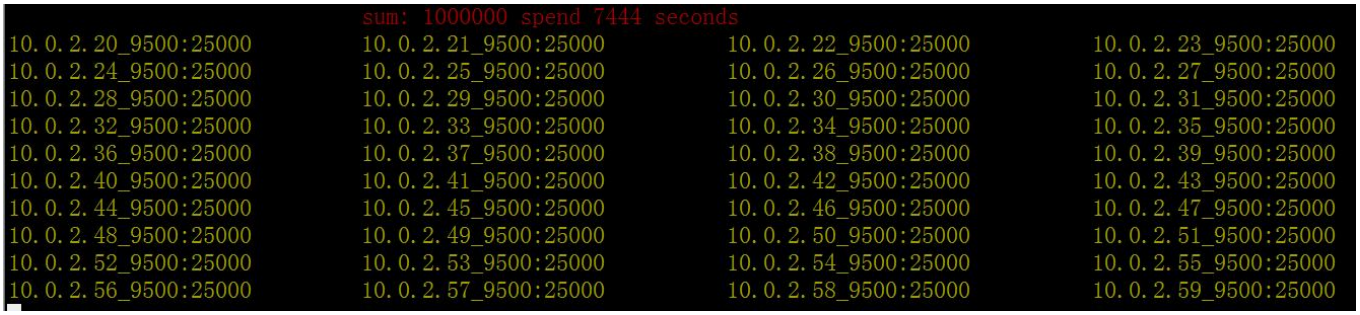
入脚本,结果校验脚本，收集脚本这样一些列的脚本。

- 自动生成部署需要的ansible脚本
- 在本地生成初始数据
- 生成主机IP列表，并将主机IP列表配置到ansible的hosts文件中，这样ansible可以对所有主机进行控制
- 运行部署的ansible脚本，将所有的初始数据和程序，系统配置文件都部署到主机上，并设置权限
- 为每台主机重新设置内核参数，并关闭防火墙
- 在每台主机上，启动redis服务
- 在每台主机上，将初始数据载入redis服务器
- 启动服务器的监控服务器
- 启动服务注册发现服务
- 在每台服务器上，启动服务程序，让服务程序自动注册

做到这一步就完成了大部分的准备工作 （默认的配置是4个rg, 每个rg是1000位用户，在运行前，修改脚本 num*rg 代表rg的数目 rg*size 代表这个rg里的用户数目 如果是100万用户，可以采用40台主机，每台主机2.5万个用户的配置)

系统测试

1. 重新建立一个连接，通过10.0.1.11 主机，跳转进入10.0.2.20主机，运行monitorclient . 这样我们就可以
2. 在10.0.1.11 主机上利用 screen 启动所有的压力测试客户端 运行一下命令启动测试客户端 screen ansible all -f 40 -a "$HOME/bin/client.sh" 使用screen 的原因是，我们可以将它切入后台。当我们和10.0.1.11 的网络断开的时候，不会意外打断我们的测试。 因为测试可能要运行几个小时。
3. 观察状态变化。当所有的客户端显示完成的时候，表示成功。



1. 在10.0.1.11 运行 命令，这个命令会让每个主机，对完成的结果校验，并将结果写入文件 运行以下命令启动校验结果 screen ansible all -a "$HOME/bin/checkdb.sh"
2. 利用收集脚本将数据收集到10.0.1.11上面，把结果数据拖回本地
3. 利用脚本，关闭所有的主机！这个每分钟都是钱 运行以下命令关闭所有的ec2

```
E:\fakewechat\src\github.com\fakewechat\aws>python close_ec2.py
[Instance:i-a1c9982e]
i-a1c9982e Instance:i-a1c9982e
[Instance:i-e8c69767]
i-e8c69767 Instance:i-e8c69767
[Instance:i-68c697e7]
i-68c697e7 Instance:i-68c697e7
[Instance:i-1dc99892]
i-1dc99892 Instance:i-1dc99892
[Instance:i-edc69762]
i-edc69762 Instance:i-edc69762
[Instance:i-6cc697e3]
i-6cc697e3 Instance:i-6cc697e3
[Instance:i-e1c6976e]
i-e1c6976e Instance:i-e1c6976e
[Instance:i-12c9989d]
i-12c9989d Instance:i-12c9989d
```

1. 慢慢的整理结果 完毕

结果对比

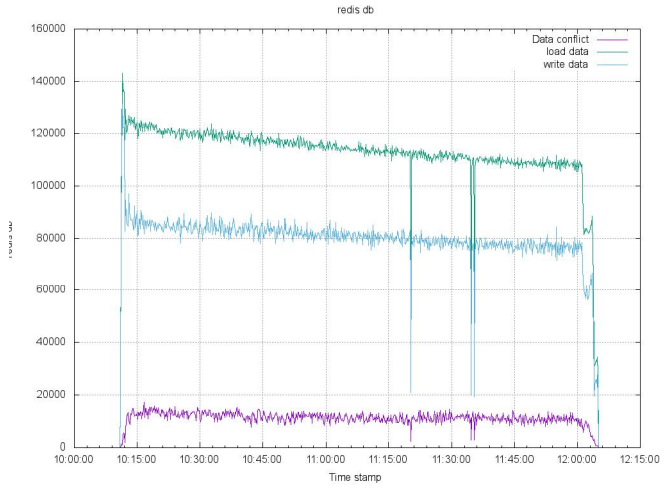
数据分布

脚本总共生成了100万位用户，用户之间的相互关系29741136条，平均每位用户接近30条。其中每位用户在同一个rg的关系为20条，跨rg的关系为10条。每位用户向自己的好友发送5条消息，测试共产生了1.4 亿条(148705680)条消息，并完全正确接受。经过验证数据的没有问题，每位用户都成功发送了消息，也成功接受了消息。 因为消息需要跨服务器进行交互，从侧面证明，系统不是单独在单机上运作，是全集群有消息交互的。

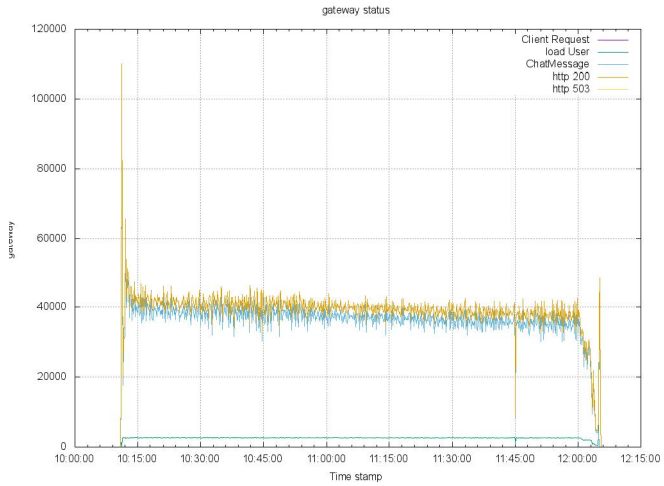
性能比较

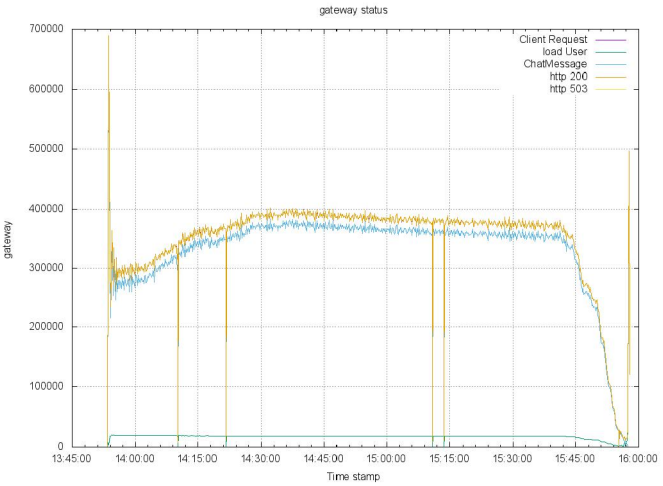
1台服务器 和2台服务器的数据可以见

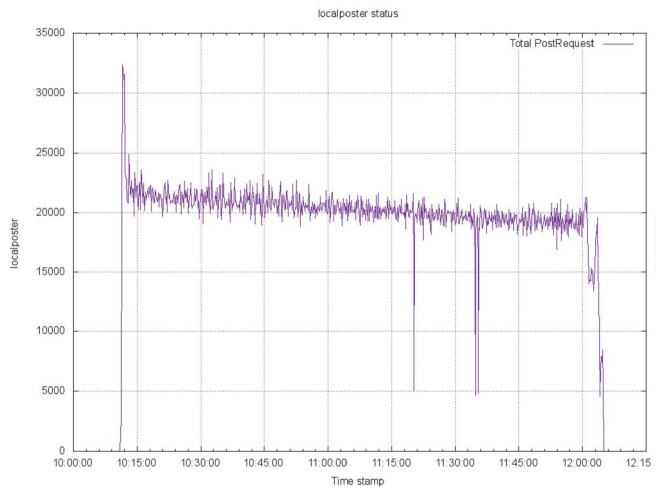
4台服务器处理10万用户，40台服务器处理100万用户 对比

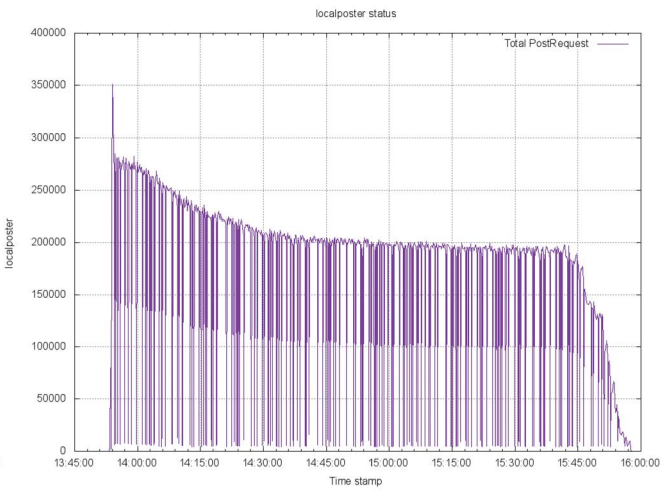

redis db
Data conflict
load data
write data
redis db
Time stamp

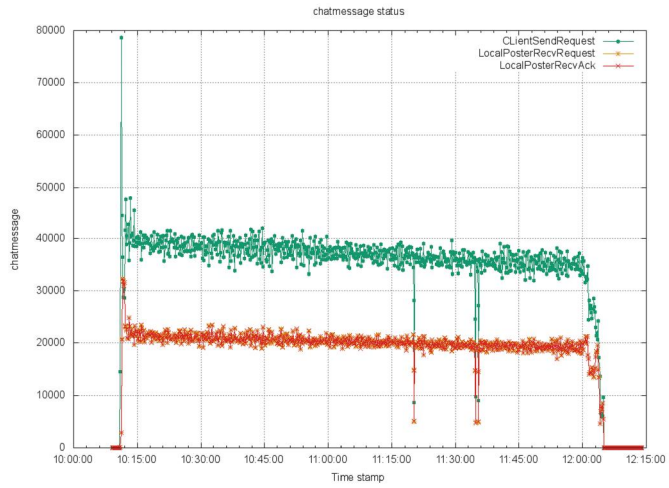

redis db
Data conflict
load data
write data
redis db
Time stamp

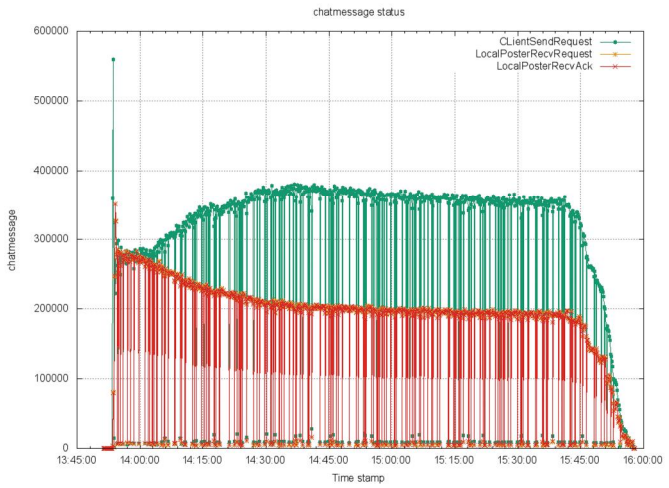

gateway status
Client Request
load User
ChatMessage
http 200
http 503
gateway
Time stamp

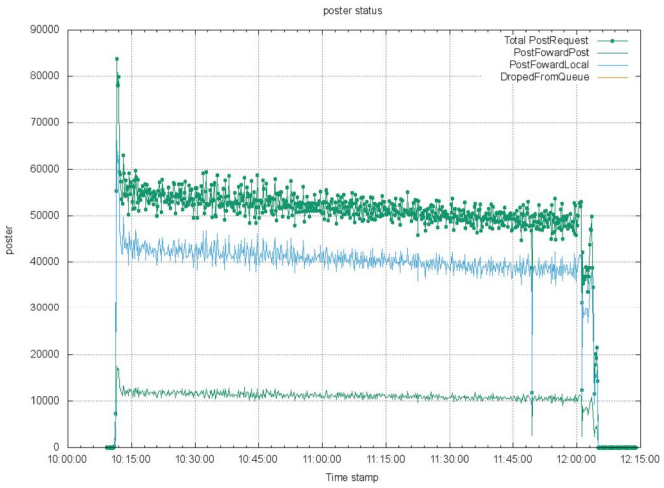

gateway status
Client Request
load User
ChatMessage
http 200
http 503
gateway
Time stamp

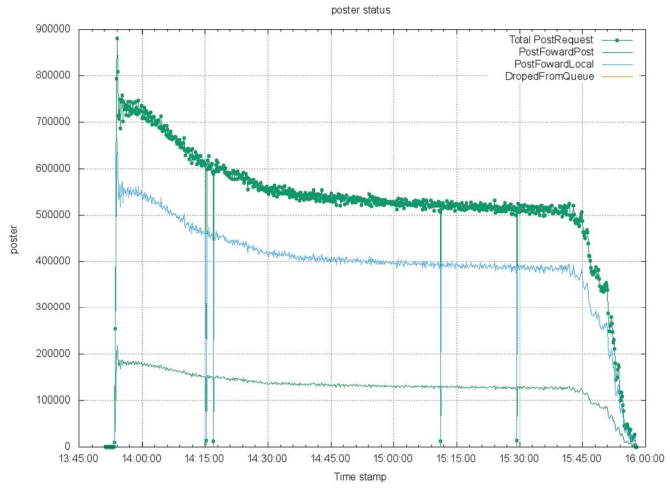

localposter status
Total PostRequest
localposter
Time stamp


localposter status
Total PostRequest
localposter
Time stamp









可以看到 计数器显示，系统的吞吐数据有接近线性的增长，这表明软件原型 在100万级别有水平扩展的能力。

教训

仔细回想了一下，经验收获不多，但是教训有一些。列下来引以为戒吧！

1.数据注入成为瓶颈

最开始的设计，在10.0.1.11 上将所有的数据准备好，然后导入redis 服务器。开始一切都很好，当开始百万用户级别的时候，导入这些用户的时间就让人无法接受，每台主机需要接近10分钟才能导入完成。40台主机就是400分钟。这是极大的浪费。所以 流程改进为改完所有的数据文件，复制到每一台主机上，用golang的程序载入，基本做到了几十秒内完成。

教训: 大规模并发服务，任何地方都不能单点运行,当规模扩大了，缺陷也立刻被放大。

2.T2.Mircro 导致CPU 挂起

一开始希望用低端的主机，实现"蚂蚁吃大象"的办法完成百万级用户测试，但是到最后的时候，因为

CPU分数用尽，导致系统基本停止。最后只有换比较高端的主机。

教训 对系统的CPU使用情况没有仔细分析，对AWS主机分类理解不够

3.配置bug导致端口冲突

生成配置的脚本bug，导致某台主机的端口冲突，引起某个服务自动下线。最后在监控上发现服务数目不够，才查到了原因。

教训 监控需要更加智能，更细致。

4.网络不畅导致断线

连接AWS的网络连接并不稳定，结果某次断线,导致测试中断，最后重新测试

教训： 需要长期运行的程序 应该用screen这样的程序放到后台, 更好的办法是service化。 此外如果程序不是无状态的，就无法重试继续，需要对流程进行改进。

5.小细节导致流量暴增

本来监控程序的刷新频率，靠业务数据驱动，每次有更新就自动刷新，用户少的时候，没有问题。当40台服务器一起更新的时候，刷新速度就没办法看了，导致流量涨得太高。 只能改成每3秒刷新一下

教训 品质细节需要实践

6.进程部署是瓶颈

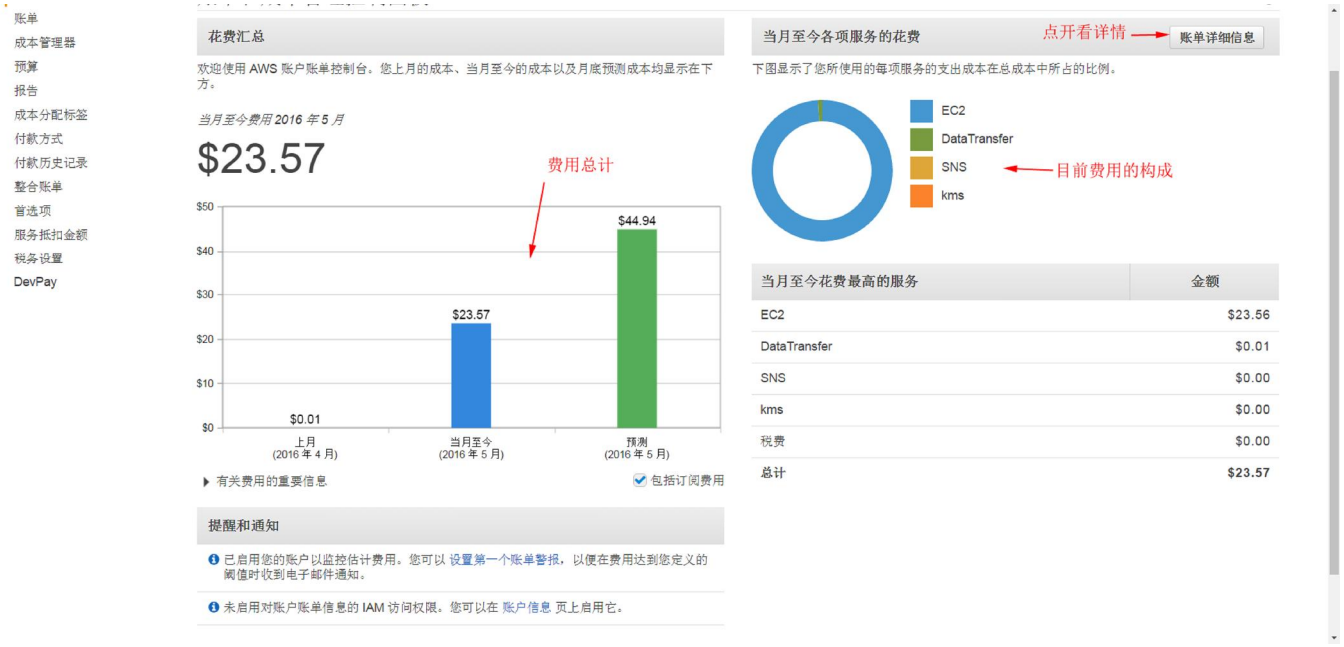
虽然使用了ansible分发程序，但是可以看得出分发过程，仍然是系统的瓶颈，同时也是个单点故障。这套部署架构并不适合更多的主机。

教训 大规模云计算系统里应该利用p2p 进行程序部署，分发。

账单问题

AWS的账单可以从这里看

账单的组成部分：



点击查看详单

流量部分

详细信息		总计
AWS 服务费		**$23.57**
Data Transfer		**$0.01**
Asia Pacific (Tokyo) Region	**用量**	
AWS Data Transfer APN1-USE1-AWS-In-Bytes		
$0.00 per GB - Asia Pacific (Tokyo) data transfer from US East (Northern Virginia)	0.000000070 GB	$0.00
总计:		**$0.00**
AWS Data Transfer APN1-USE1-AWS-Out-Bytes		
$0.09 per GB - Asia Pacific (Tokyo) data transfer to US East (Northern Virginia)	0.000000040 GB	$0.00
总计:		**$0.00**
AWS Data Transfer APN1-USW1-AWS-In-Bytes		
$0.00 per GB - Asia Pacific (Tokyo) data transfer from US West (Northern California)	0.000000180 GB	$0.00
总计:		**$0.00**
AWS Data Transfer APN1-USW1-AWS-Out-Bytes		
$0.09 per GB - Asia Pacific (Tokyo) data transfer to US West (Northern California)	0.000000150 GB	$0.00
总计:		**$0.00**
Bandwidth		
$0.000 per GB - data transfer in per month	0.559 GB	$0.00
$0.000 per GB - first 1 GB of data transferred out per month	0.073 GB	$0.00
$0.010 per GB - regional data transfer - in/out/between EC2 AZs or using IPs or ELB	0.415 GB	$0.01
总计:		**$0.01**
地区总计:		**$0.01**

你需要为你的服务器接收的每个字节付钱

主机部分

地区总计:		**$0.00**
Elastic Compute Cloud		**$23.56**
Asia Pacific (Tokyo) Region	**用量**	
Amazon Elastic Compute Cloud running Red Hat Enterprise Linux		
$0.080 per On Demand RHEL t2.micro Instance Hour	260 Hrs	$20.80
$0.193 per On Demand RHEL c4.large Instance Hour	12 Hrs	$2.32
总计:		**$23.12**
EBS		
$0.12 per GB-month of General Purpose SSD (gp2) provisioned storage - Asia Pacific (Tokyo)	3.656 GB-Mo	$0.44
总计:		**$0.44**
地区总计:		**$23.56**
Key Management Service		**$0.00**
Asia Pacific (Tokyo) Region	**用量**	
AWS Key Management Service ap-northeast-1-KMS-Requests		
$0.00 per request - Monthly Global Free Tier for KMS requests	1 Requests	$0.00
总计:		**$0.00**
地区总计:		**$0.00**
Simple Notification Service		**$0.00**
Asia Pacific (Tokyo) Region	**用量**	
Amazon Simple Notification Service APN1-Requests-Tier1		
First 1,000,000 Amazon SNS API Requests per month are free	1 Requests	$0.00
总计:		**$0.00**

主机存在着就算钱，1小时起

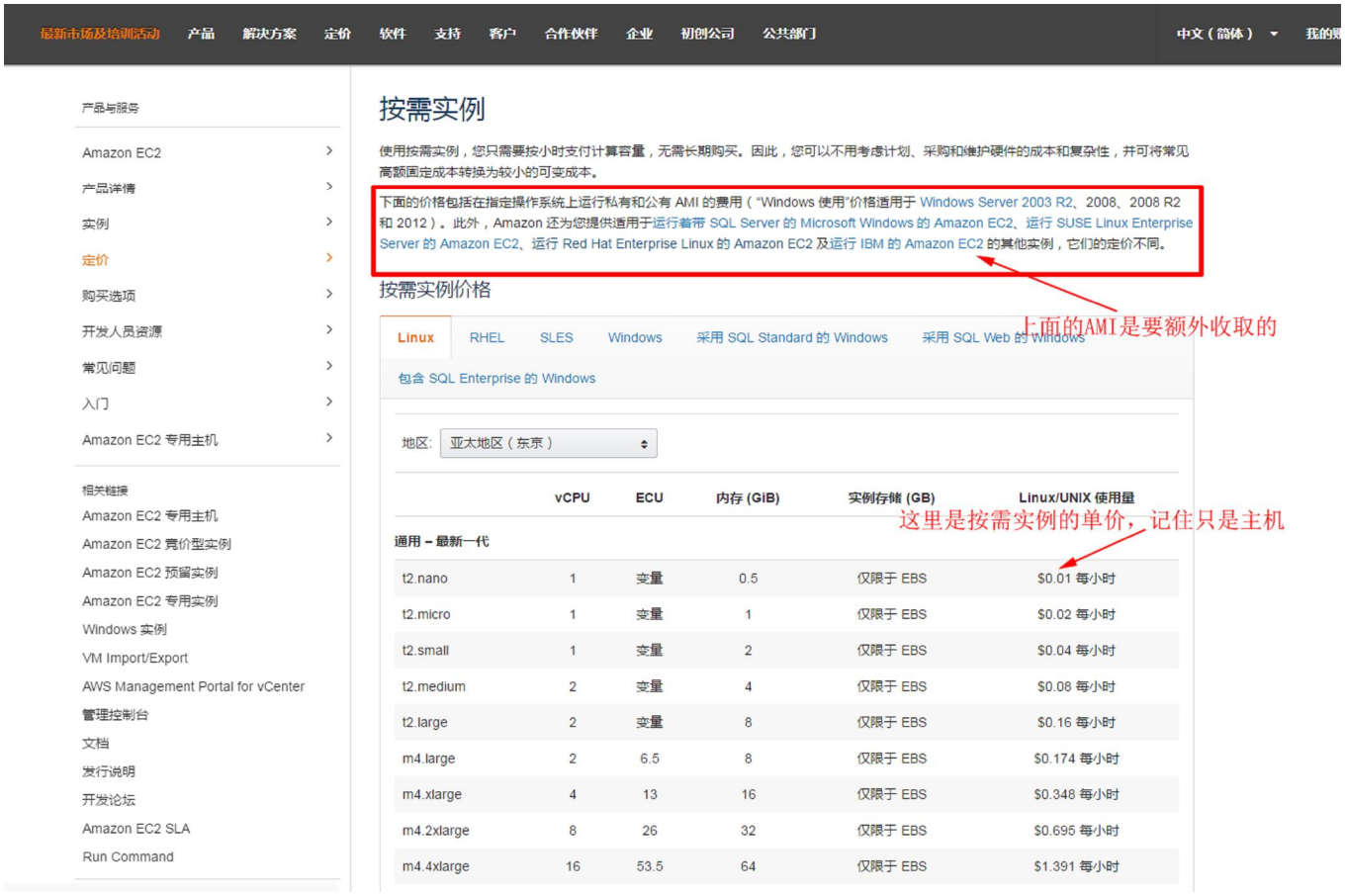
RHEL 需要为软件付费

硬盘也是需要收钱的

密钥管理也要钱

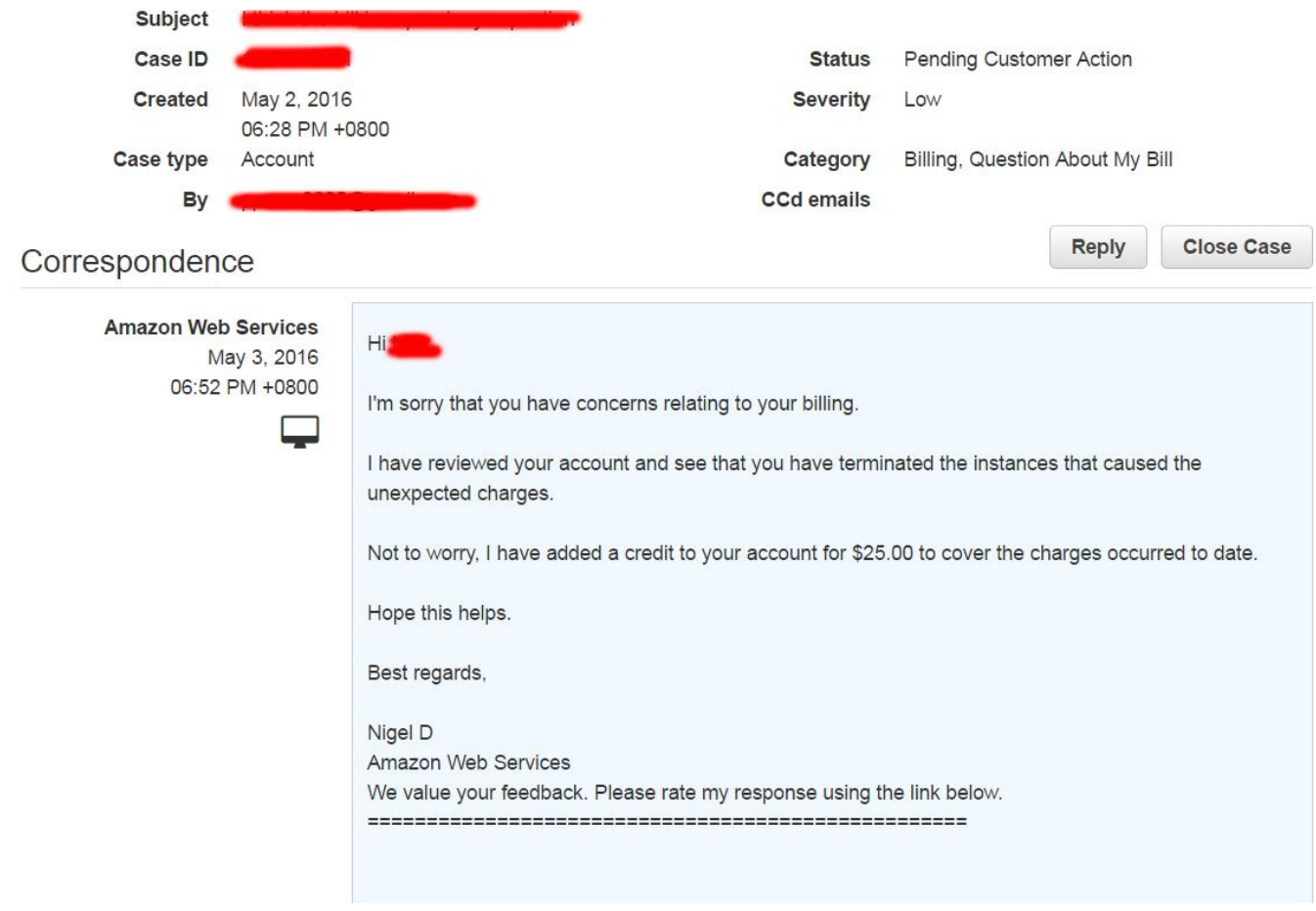
价格可以从这里看

http://aws.amazon.com/cn/ec2/pricing/



总之，用完以后，立刻删除所有的资源！

最后提醒一下，AWS的客服非常好！有任何问题都可以直接和他交流，可以很快得到解决。



如果觉得对项目有兴趣可以访问 https://github.com/xiaojiaqi/fakewechat 本文地址

https://github.com/xiaojiaqi/fakewechat/wiki/Stress-Testing-in-the-Cloud 会持续更新