Redash 开源中文版
V9.0.0
安装部署培训教程

Redash 中文版维护方

武汉达之云计算有限公司

www.dazdata.com

目录：

日程:

日程	上午	下午
第一天	第一章	第二章
第二天	第三章	第四章
第三天	第五章	第六章
第四天	第七章	第八章
第五天	第九章	第十章

第一章、Redash 技术架构

1、Redash 简介

Redash 宗旨是使任何人，无论技术水平如何，都可以利用数据的力量。熟悉 SQL 语言的用户利用 Redash 来探索数据，组织查询，设计可视化报表；并可通过共享报表，使得来自任何数据源的数据，都能够被组织中的任何人使用这些数据。每天全球成千上万个组织中的数百万用户使用 Redash 来开发探索数据，并制定数据驱动的决策。

Redash 功能：

基于浏览器的：浏览器中的所有内容，都带有可共享的 URL。

易于使用：无需掌握复杂软件即可立即获得数据。

查询编辑器：使用模式浏览器快速组成 SQL 和 NoSQL 查询并自动完成。

可视化和仪表板：通过拖放创建漂亮的可视化文件，并将它们组合成一个仪表板。

共享：通过共享可视化及其相关查询轻松进行协作，从而实现对报告和查询的同行审阅。

计划刷新：根据您定义的固定时间间隔自动更新图表和仪表盘。

警报：定义条件并在数据更改时立即得到警报。

REST API：可以通过 REST API 使用 UI 进行的所有操作。

对数据源的广泛支持：可扩展的数据源 API，具有对一长串常见数据库和平台的本机支持。

2、Redash 技术架构

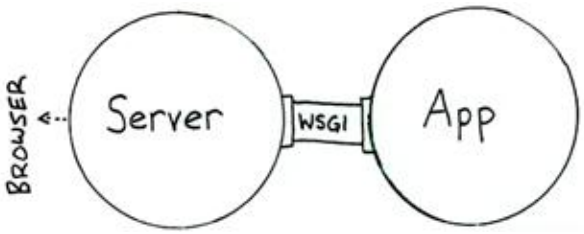
技术架构：Redash 的后台主要分为三部分，为 Server 服务器、Worker 任务执行程序和 Scheduler 任务调度程序。Server 服务器是一个具有 WSGI 接口的 Web 应用，Server 服务器是基于 Python 的 Flask 框架开发的，调式环境下 Flask 框架自带简易应用服务器，生产环境需要配备 uWSGI 应用服务器。Worker 任务执行程序和 Scheduler 任务调度程序是命令行终端程序。

Server 服务器、Worker 任务执行程序和 Scheduler 任务调度程序三者之间没有任何任何依赖，不存在调用关系，完全依靠居中的 Redis 来进行消息发布订阅(Pub/Sub)服务。这套架构模式是通过 Python 的 RQ 组件来完成的。以下简单介绍技术架构主要涉及的概念：

Flask：

Redash 后端采用了 Flask 框架，Flask 是一个微型的 Python 开发的 Web 框架，基于 Werkzeug WSGI 工具箱和 Jinja2 模板引擎。 Flask 也被称为“microframework”，因为它使用简单的核心，用 extension 增加其他功能。Flask 没有默认使用的数据库、窗体验证工具。然而，Flask 保留了扩增的弹性，可以用 Flask-extension 加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。

WSGI：



全称 Python Web Server Gateway Interface，指定了 web 服务器和 Python web 应用或 web 框架之间的标准接口，以提高 web 应用在一系列 web 服务器间的移植性。 从以上介绍我们可以看出：

WSGI 是一套接口标准协议/规范；通信（作用）区间是 Web 应用服务器和 Python Web 应用程序之间；目的是制定标准，以保证不同 Web 服务器可以和不同的 Python 程序之间相互通信。

Postgresql：

PostgreSQL 是一种特性非常齐全的自由软件的对象-关系型数据库管理系统，是以加州大学计算机系开发的 POSTGRES 为基础的对象关系型数据库管理系统。POSTGRES 的许多领先概念只是在比较迟的时候才出现在商业网站数据库中。PostgreSQL 支持大部分的 SQL 标准并且提供了很多其他现代特性，如复

杂查询、外键、触发器、视图、事务完整性、多版本并发控制等。同样，PostgreSQL 也可以用许多方法扩展，例如通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语言等。另外，因为许可证的灵活，任何人都可以以任何目的免费使用、修改和分发 PostgreSQL。

RQ：

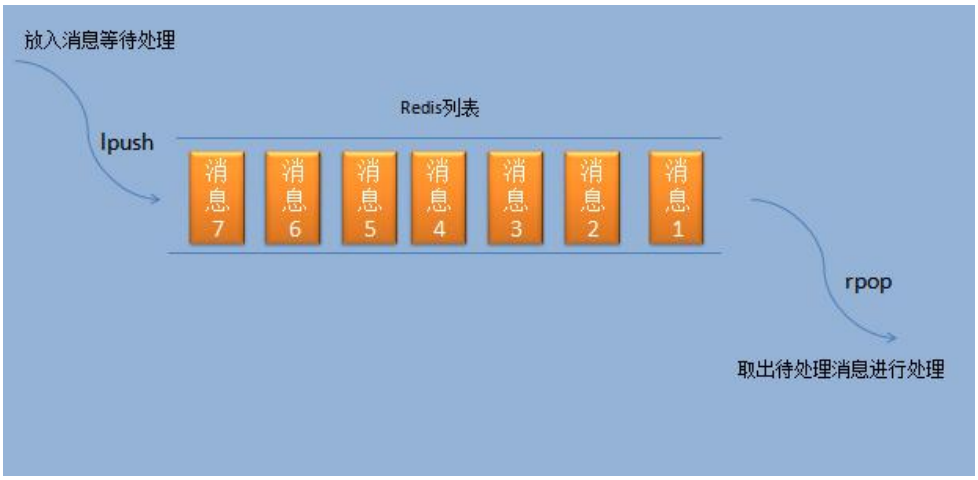
RQ (Redis Queue) 是一个简单的 Python 库用于将作业放到队列中并在后台统一执行，使用 Redis 做后端，可方便的跟 Web 前端集成。

Redash 所有外部数据源的链接测试或执行取数，均在队列里完成。

Redis：

Redis（Remote Dictionary Server)是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。

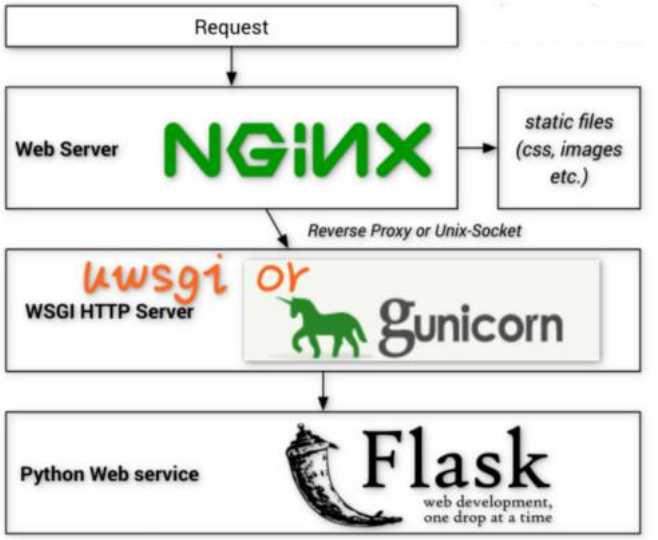
Redis 不仅可作为缓存服务器，还可用作消息队列。它的列表类型天生支持用作消息队列。如下图所示：



Nginx：

Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在 BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上 nginx 的并发能力在同类型的网页服务器中表现较好，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。中国大陆使用 nginx 网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

生产环境：



基于 Python Flask 框架的典型生产环境配置，外加 Supervisor 作为 uWSGI 的进程守护程序。

第二章、Ubuntu 本地部署

Redash 中文版本地安装方式比较繁琐，需要具备一定的 linux 基础知识，主要适用于试用或用于开发的基础环境；同时由于安装过程需要访问国外服务器，极难一次安装成功。在准备好 Ubuntu20.04 中文版环境后，就可以开始以本地方式安装 Redash 中文版。

本安装过程是基于 Ubuntu20.04 的内置 Python3.8 的，如果安装了多版本 Python,一些底层库可能需要下载源码重新编译，比较繁琐，因此不建议在多版本 Python 环境下安装 Redash 中文版。

另外：Ubuntu20.04 安装后，最好还要设置 root 密码：sudo passwd root 一定要更换本地源并注意执行 sudo apt-get update && sudo apt-get upgrade 和当前用户加入 root 组首先 su- 然后 usermod -g root 当前 ubuntu 用户名。Ubuntu20.04 内置 Python3.8.2。

1、基础环境安装

1）、安装 git：sudo apt install git

2）、安装 Python3 的 Pip：这是 python 官方的包管理工具属于是 python 的一部分，负责下载安装 python 的包文件

sudo apt install python3-pip

3）、安装 nodejs 和 npm：nodejs 是 js 文件的一种运行环境，npm 是 nodejs 的包管理器。

sudo apt install nodejs npm

4）、更换 npm 国内源:

sudo npm config set registry https://registry.npm.taobao.org

验证:sudo npm config get registry

5）、安装 nodejs 版本管理器：负责 nodejs 的版本问题方便后续更新

sudo npm install n -g

6）、升级 nodejs 最新版本:

sudo n stable

7）、安装 Redis：处理 redash 中文版缓存消息队列,以及处理发布订阅

sudo apt install redis-server

2、Postgresql9.6 安装配置：

Redash 使用 postgresql 作为查询结果缓存和定义信息、运行数据等的存储数据库。

1）、安装基础包：sudo apt install wget ca-certificates

2）、加载源:

sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

3）、下载证书:

wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

4）、刷新源：sudo apt-get update

5）、安装 Postgre9.6:

sudo apt install postgresql-9.6 postgresql-contrib-9.6 postgresql-client-9.6 libpq-dev pgadmin4 -y

6）、登录 Postgresql：sudo -u postgres psql

7）、在 postgresql 终端执行：alter user postgres with password ‘密码’;create role 当前 ubuntu 用户名;alter role 当前 ubuntu 用户 login; (注意 postgresql 命令语句以分号结束)

- 8）、退出 postgresql 终端：\q

3、安装和初始化

1）、下载代码：git clone https://github.com/dazdata/redash.git

2）、进入工作目录：cd redash

3）、安装 npm 依赖包：npm install

4）、前端打包：npm run build

5）、配置 pip 国内源：sudo cp pip.conf /etc/pip.conf

6）、安装 Python 虚拟环境：sudo pip3 install virtualenv

7）、创建 Python 虚拟环境：virtualenv -p python3 venv

8）、激活 Python 虚拟环境：source venv/bin/activate

9）、安装 pip 包：pip3 install -r requirements.txt -r requirements_dev.txt

10）、Python3.8 安装 pip 包：pip3 install importlib_resources==1.5

11）、若需要支持 Oracle 数据源，先安装 Oracle 客户端程序，后执行：pip install -r requirements_oracle_ds.txt (试用建议不安装)

12）、若需要支持 mysql，请先启动一个新的命令行终端，执行：

sudo apt install default-libmysqlclient-dev (试用建议不安装)

再回到当前命令行终端执行：

pip install mysqlclient (试用建议不安装)

14）、若需要其它数据源：pip install -r requirements_all_ds.txt (试用建议不安装)

15）、初化数据库表结构：./manage.py database create_tables

16）、退出虚拟环境，安装完成：deactivate

4、启动和配置

1）、启动服务：

分别打开三个终端，都执行 cd redash 进入目录后分别执行下列三命令之一：

```
source venv/bin/activate
./manage.py runserver --debugger --reload
和
source venv/bin/activate
./manage.py rq worker
和
source venv/bin/activate
./manage.py rq scheduler
```

2）、每次终端启动运行一个服务。若需要停止服务，直接按 Ctrl+C 键，执行 deactivate 退出 python 虚拟环境。

进入浏览器：http://localhost:5000

5、Redash 环境变量

Redash 设计非常灵活，许多功能都可以通过设置进行更改，设置是通过 redash.settings 环境变量读取的，可以在/opt/redash/current/.env 文件中进行设

置，当然也可以在/etc/profile 或~/.bashrc 等系统级或用户级环境变量文件中设置。详见附件：Redash 环境变量清单。

6、非代码方式部署

Redash 前端开发环境为 React，只要不改变前后端 ajax 的服务请求链接信息，经过打包后的~/client/dist/文件夹是可以复制到其它机器运行的。Redash 后端 python 环境代码可以复制运行，但三方包需要在新机器重新安装。

第三章、Centos 本地部署

1.Centos7 本地部署

1)、更改 yum 国内源

(1)进入源文件目录下:
cd /etc/yum.repos.d/
(2)备份原版本 yum 文件:
sudo mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup
(3) 下载阿里 yum 源文件作为默认源文件:
sudo wget -O /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.aliyun.com/repo/Centos-7.repo
(4)清理原缓存:
sudo yum clean all
(5)建立缓存以后方便缓存中搜索:
sudo yum makecache
(5)更新系统包:
sudo yum update -y
(6)安装常用包:
sudo yum -y install gcc gcc-c++ kernel-devel make zlib zlib-devel libffi-devel openssl-devel git python3-devel postgresql-devel*
(7)进入 root 用户下修改当前用户添加到 root 组:
su -
sudo usermod -g root 当前 centos 用户名

2)、安装 PostgreSql9.6

(1)安装 postgresql 源:
sudo yum install -y https://download.postgresql.org/pub/repos/yum/9.6/redhat/rhel-7-x86_64/pgdg-redhat-repo-42.0-11.noarch.rpm
(2)安装 postgresql6 客户端:
sudo yum install -y postgresql96
(3)安装 postgresql6 服务端:
sudo yum install -y postgresql96-server
(4)初始化:
sudo /usr/pgsql-9.6/bin/postgresql96-setup initdb
(5)设置开机自启:
sudo systemctl enable postgresql-9.6
(6)启动 postgresql9.6 服务:
sudo systemctl start postgresql-9.6
(7)进入 postgresql 数据库修改用户名密码:

1.sudo -u postgres psql
2.alter user postgres with password '此处填写登陆密码';
3.create role 当前 centos 用户名;
4.alter role 当前 centos 用户名 login;
5.\q

3)、安装 Redis

(1)下载 fedora 的 epel 仓库:sudo yum install epel-release -y
(2)安装:sudo yum install redis
(3)启动 redis:sudo systemctl start redis
(4)设置开机自启:sudo systemctl enable redis.service

4)、安装 Nodejs

sudo yum install nodejs -y
sudo npm config set registry https://registry.npm.taobao.org
sudo npm install n -g
sudo n stable

5)、安装 python3

(1)sudo yum -y groupinstall "Development tools"
(2)sudo yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel gdbm-devel db4-devel libpcap-devel xz-devel python3-devel
(3)sudo wget https://www.python.org/ftp/python/3.6.2/Python-3.6.2.tar.xz
(4)sudo mkdir /usr/local/python3

(5)tar -xvJf Python-3.6.2.tar.xz
(6)cd Python-3.6.2
(7)./configure --prefix=/usr/local/python3
(8)sudo make && sudo make install
(11)添加环境变量:
sudo vi /etc/profile
    # 将下面内容添加到文件的最下面
    PATH=$PATH:/usr/local/python3/bin
    #是添加的进行生效命令
    source /etc/profile
    # 最后查看是否添加成功
    echo $PATH

6)、安装 Redash 源码

1、下载代码:
    git clone https://github.com/dazdata/redash.git
    cd redash
更换国内源:npm config set registry https://registry.npm.taobao.org
查看当前源:npm config get registry
2、前端安装依赖包：npm install
3、前端打包：npm run build
4、配置 pip 国内源：sudo cp pip.conf /etc/pip.conf
5、安装 Python 虚拟环境：sudo pip3 install virtualenv
6、创建 Python 虚拟环境：virtualenv venv
7、激活 Python 虚拟环境：source venv/bin/activate
8、安装 Pip 包:
sudo pip3 install --upgrade pip
pip3 install psycopg2-binary
  pip3 install -r requirements.txt -r requirements_dev.txt -r requirements_bundles.txt
9、初始化数据库表结构：./manage.py database create_tables
10、退出虚拟环境，安装完成：deactivate
11、分别打开三个终端，都执行 cd redash 进入目录后分别执行下列三命令之一:
    source venv/bin/activate
    ./manage.py runserver --debugger --reload
    和
    source venv/bin/activate
    ./manage.py rq worker
    和
    source venv/bin/activate
    ./manage.py rq scheduler

2. Centos8 本地部署

Centos8 内置的 Python 为 3.6.8 版本，以下是在内置 Python3.6.8 基础上的安装步骤。由于安装多版本 Python 会导致系统底层库需要下载源码重新编译，比较麻烦，不建议在多版本 Python 环境下安装 Redash 中文版。

1)、初始化环境安装

由于 Centos 使用 yum 下载源,需要更改成国内源:

1)、进入设置 yum 源目录下：cd /etc/yum.repos.d/

2)、下载自动下载文件工具 wget：sudo yum install wget

3）、将 yum 源文件备份一下：sudo mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup

4）、下载阿里 yum 源：sudo wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-8.repo

5）、清理原缓存：sudo yum clean all

6）、建立缓存以后方便缓存中搜索：sudo yum makecache

7）、更新系统包：sudo yum update -y

8）、安装常用包：

sudo yum -y install gcc gcc-c++ kernel-devel make zlib zlib-devel libffi-devel openssl-devel python3-devel git

9）、安装 dnf:sudo yum install dnf -y

10）、进入 root 用户下将当前用户添加到 root 组：

su -

usermod -g root 当前 centos 用户名

2)、Postgresql9.6 安装

安装 PostgreSql9.6 初始化并配置为系统启动时自动启动：

sudo dnf install @postgresql:9.6

sudo dnf install postgresql-contrib

sudo yum install postgresql-devel

sudo postgresql-setup initdb

sudo systemctl enable --now postgresql

登录 Postgresql：sudo -u postgres psql

更改 postges 密码：alter user postgres with password '密码';

创建角色：create role 当前 centos 用户名;

赋予登录权限：alter role 当前 centos 用户名 login;

3)、Redis 安装

sudo yum install redis

sudo systemctl start redis

sudo systemctl enable redis.service

4)、安装 nodejs

sudo yum install nodejs

sudo npm config set registry http://registry.npm.taobao.org/

sudo npm config get registry

sudo npm install n -g

sudo /usr/local/bin/n stable

5)、安装源码初始化

1、下载代码：

git clone https://github.com/dazdata/redash.git && cd redash

2、前端安装依赖包：npm install

3、前端打包：npm run build

4、配置 pip 国内源：sudo cp pip.conf /etc/pip.conf

5、安装 Python 虚拟环境：sudo pip3 install virtualenv

6、创建 Python 虚拟环境：virtualenv venv

7、激活 Python 虚拟环境：source venv/bin/activate

8、安装 Pip 包：

pip3 install -r requirements.txt -r requirements_dev.txt -r requirements_bundles.txt

9、初始化数据库表结构：./manage.py database create_tables

10、退出虚拟环境，安装完成：deactivate

6)、启动程序

分别打开三个终端，都执行 cd redash 进入目录后分别执行下列三个命令之一

source venv/bin/activate

./manage.py runserver --debugger --reload

和

source venv/bin/activate

./manage.py rq worker

和

source venv/bin/activate

./manage.py rq scheduler

打开浏览器，输入地址：http://localhost:5000

第四章、Docker 部署

相比 Linux 环境本地安装而言，Docker 安装方式更为简便，Docker 脚本化安装过程会自动获取 Redis、postgres、Python3.7 镜像，构造 Redash 最新的后台 server、worker、schedule 镜像。前端 npm 依赖包安装和前端最新代码打包是通过卷映射方式挂载到 server 容器，因此这些工作需要人工一次执行；另外构建初始数据库表结构也需要人工一次执行。这些动作执行完毕，启动 Docker 容器就可以访问了。由于安装过程需要访问国外服务器，极难一次安装成功，需要反复试验几次。

1、基础环境搭建

1)、更换 APT 国内源：

```
sudo sed -i s@/archive.ubuntu.com/@/mirrors.aliyun.com/@g /etc/apt/sources.list
```

2)、更新源:

```
sudo apt update && sudo apt upgrade -y
```

3)、安装 Docker 和 Docker-Compose:

```
sudo apt install docker docker-compose -y
```

4)、更换 Docker 国内源:

```
sudo nano /etc/docker/daemon.json
```

输入:

```
{
"registry-mirrors": ["https://m3dz4myl.mirror.aliyuncs.com"]
}
```

5)、重启 Docker 服务:

```
sudo systemctl restart docker
```

6)、安装 Nodejs 和 npm:

```
sudo apt install nodejs npm -y
```

7)、更换 npm 国内源:

```
sudo npm config set registry https://registry.npm.taobao.org
```

8)、升级最新 Nodejs 版本:

```
sudo npm install n -g
sudo npm install -g npm
sudo npm cache clean -f
sudo n stable
```

2、下载源码初始化：

下载源码:

```
https://github.com/dazdata/redash.git
cd redash
前端 npm 依赖包安装(警告可以忽略，若报错需要重来多试几次)：
npm install
前端打包：
npm run build
初始化数据库表结构：
sudo docker-compose run --rm server create_db
```

3、启动 Docker 自动安装执行：

```
sudo docker-compose up
```

4、启动系统&初始设置：

```
启动 Docker 容器：
sudo docker-compose start
若要停止 Docker 容器：
sudo docker-compose stop
进入浏览器：
http://localhost:5000
初次使用：
请见《初始配置 Redash 中文版》
后续使用：
注意一定先执行 cd redash 后，再执行 sudo docker-compose start 以启动 docker 容器。
```

第五章、本地生产环境

Redash 生产环境安装是基于第二章或第三章本地安装基础上进行，需要额外进行的事项主要工作是部署 WWW 服务器和应用服务器。本编介绍基于 Python 和 Flask 项目生产环境，最常见的 uwsgi 应用服务器、Supervisor 进程守护程序和 nginx 反向代理服务器，以及生产环境压力测试常用工具 jMeter、备份恢复、重安装等一一介绍。

1、uwsgi 应用服务器

1）、安装

```
sudo pip3 install uwsgi
```

2）、配置

uwsgi 执行一般有两种方式：命令行和文件配置，但是命令行可能需要识记很多参数，因此采用文件配置是更通用的做法，文件格式支持很多种比如 ini、xml、yaml 等，笔者建议还是采用比较简单 key-value 形式 ini 模式，下面给出一个简单的 uwsgi ini 配置实例：

```
[uwsgi]

http=:5000
chdir=/home/当前 centos 用户名/redash/
wsgi-file=redash/wsgi.py
callable=app
master=true
virtualenv=/home/当前 centos 用户名/redash/venv/
pythonpath=/home/当前 centos 用户名/redash/
processes=1
threads=2
```

执行：uwsgi --ini uwsgi.ini

参数解释：

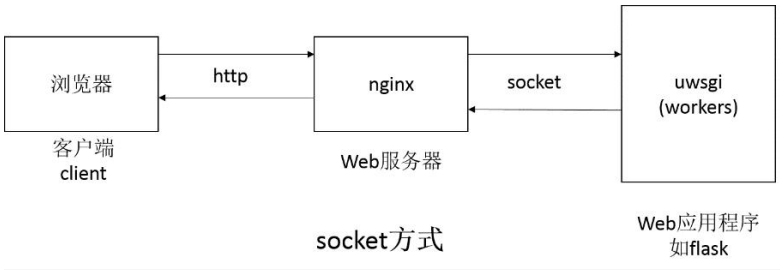
- socket：socket 文件，也可以是地址+端口；
- master：是否启动主进程来管理其他进程；
- chdir：项目的根目录；
- module：wsgi 文件相对路径；
- home：虚拟环境目录；
- workers：开启的进程数量；

- reload-mercy：设置在平滑的重启（直到接收到的请求处理完才重启）一个工作子进程中，等待这个工作结束的最长秒数；
- vacuum：服务结束后时候删除对应的 socket 和 pid 文件；
- max_requests：每个工作进程设置的请求上限；
- limit_as：限制每个 uwsgi 进程占用的虚拟内存数目；
- buffer_size：设置用于 uwsgi 包解析的内部缓存区大小；
- pid_file：指定 pid 文件；
- harakiri：请求的超时时间；
- daemonize：进程后台执行，并保存日志到特定路径；如果 uwsgi 进程被 supervisor 管理，不能设置该参数；

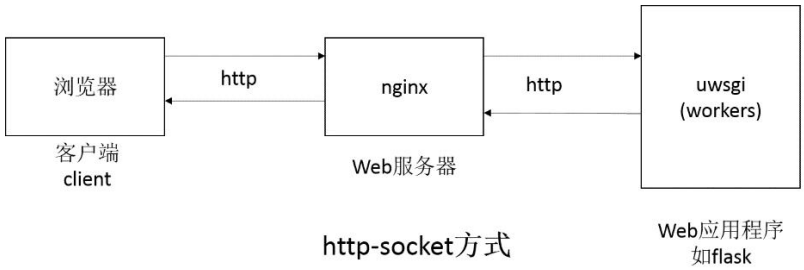
3）、uWSGI 三种通讯方式

配置：

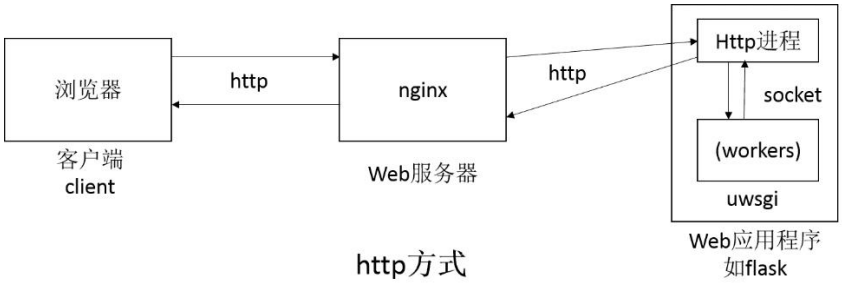
```
socket=127.0.0.1:8888
http-socket=127.0.0.1:8888
http=0.0.0.0:8888
```



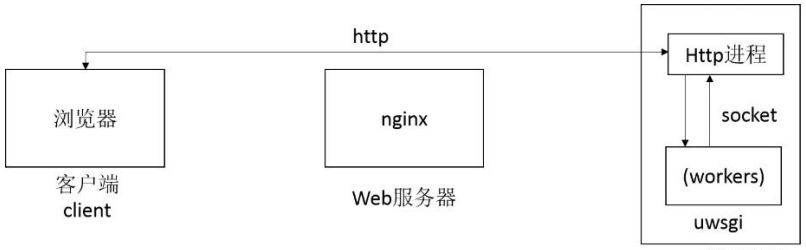
Nginx 与 uWSGI 之间用 socket 通讯。现在大部分 web 服务器（如 nginx）支持 uwsgi，这是这三种方式最高效的一种形式，socket 通信速度会比 http 快。



Nginx 与 uWSGI 之间用 http-socket 通讯，这个适用于 web 服务器不支持 uwsgi 时。



http方式



http方式

后面两个个都是 http 方式，官方推荐的方式为 socket 以及 http-socket 方式，显然使用 http 方式会额外产生一个 http 进程，如果还通过 nginx 转发，那么效率上来说是相对比较低的。

2、Supervisor 的使用

Supervisor 是用 Python 开发的一套通用的进程管理程序，能将一个普通的命令行进程变为后台 daemon，并监控进程状态，异常退出时能自动重启。它是通过 fork/exec 的方式把这些被管理的进程当作 supervisor 的子进程来启动，这样只要在 supervisor 的配置文件中，把要管理的进程的可执行文件的路径写进去即可。也实现当子进程挂掉的时候，父进程可以准确获取子进程挂掉的信息的，可以选择是否自己启动和报警。

1）、supervisor 安装

配置好 yum 源后，可以直接安装，Centos:

```
sudo pip3 install supervisor
```

Debian/Ubuntu 可通过 apt 安装:apt-get install supervisor

2）、Supervisor 生成默认配置文件

进入安装目录: cd ~/redash
echo_supervisord_conf > supervisord.conf
然后增加对 Redash 的启动配置：
方法一：在 supervisord.conf 增加 include 节
[include]
files = conf.d/*.conf
创建 conf.d 目录，其下建立配置文件 redash.conf，增加三个 program 节

```
[program:redash]
command = uwsgi uwsgi.ini

[program:worker]
directory=/root/dazdata/
command=venv/bin/python3 ./manage.py rq worker

[program:scheduler]
directory=/root/dazdata/
command=venv/bin/python3 ./manage.py rq scheduler
```

方法二：在 supervisord.conf 最后直接增加以上三个 program 节

3）、Supervisor 启动停止和查看状态

启动：supervisord -c supervisord.conf
查看：supervisorctl status
停止：supervisorctl stop all
查看开机自启：systemctl is-enabled supervisord
停用开机自启：systemctl disable supervisord

4）、Supervisor 管理后台

若需要 web 查看进程，则去掉[inet_http_server]的注释

```
[inet_http_server]
port=127.0.0.1:9001    ;IP 按需配置
username=user
password=123
```

这个监控端口容易造成 CPU 和内存占用率特别高，导致机器卡死，生产环境慎用。

5）、supervisor 开机自动启动

在目录/usr/lib/systemd/system/ 新建文件 supervisord.service,并添加配置内容：

```
[Unit]
Description=Process Monitoring and Control Daemon
After=rc-local.service nss-user-lookup.target

[Service]
Type=forking
ExecStart=/usr/bin/supervisord -c /usr/supervisor/supervisord.conf ;开机启动时执行
ExecStop=/usr/bin/supervisord shutdown
ExecReload=/usr/bin/supervisord reload
```

killMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target

3、Nginx 安装使用

Nginx (engine x) 是一个高性能的 HTTP 和反向代理 web 服务器，同时也提供了 IMAP/POP3/SMTP 服务。

(1).安装 nginx 所需的环境

sudo yum install gcc-c++ pcre pcre-devel zlib zlib-devel openssl openssl-devel -y

(2).官网下载 nginx 的安装包地址:https://nginx.org/en/download.html

1.使用 wget 命令下载:

sudo wget -c https://nginx.org/download/nginx-1.19.2.tar.gz

2.解压并且进入 nginx 目录下

tar -zxvf nginx-1.19.2.tar.gz

sudo mv nginx-1.19.2 nginx

cd nginx

3.配置(使用默认配置即可)

./configure

4.编译安装

sudo make && sudo make install

5.查找安装路径

whereis nginx

(3) 配置 nginx 的环境变量,启动、停止、开机自启 nginx

1. 配置环境变量:

(1)编辑/etc/profile 文件:

vim /etc/profile

(2) 在最后一行添加配置并且 wq 保存.

```
PATH=$PATH:/usr/local/nginx/sbin
export PATH
```

(3) 让配置生效

source /etc/profile

2.启动: ./nginx

3.停止: ./nginx -s stop

4.等进程结束后退出: ./nginx -s quit

5.重新加载配置:./nginx -s reload

2).修改 nginx 配置文件:

1. 修改 nginx.conf 文件:

vim /usr/local/nginx/conf/nginx.conf

2. 内容如下:

```
server {
        listen          80 default_server;
        server_name     localhost;

        location /static/~(.*)(\.jpg|\.png|\.gif|\.jepg|\.css|\.js) {
            alias /root/redash/client/dist/;
        }

        location / {
            root    html;
            proxy_pass http://127.0.0.1:5000;
            index   index.html index.htm;
        }
}
```

保存退出

3)、启动:

./nginx

进入浏览器输入 http://127.0.0.1:80 能够正常访问即为安装成功。

如果需要在生产环境下 nginx 使用 ssl 协议需要增强安装 ssl 和 http2 等模块

4、Postgresql 的备份恢复

1）、备份：

进入 postgresql 目录

cd /etc/postgresql/9.6/main

执行备份命令.

pg_dump -h 127.0.0.1 -U dbUserName dbName > /home/io/databasename.bak

解释:

127.0.0.1 为 数据库所在计算机 ip;(必须保证数据库外部访问权限)

dbUserName 需要备份的数据库的用户名;

dbName 是需要备份的数据库名; /home/io/databasename.bak 是最后生成的文件的路径和文件名称(可自定义); 执行完成之后,可以去/home/io/路径下查看是否生成.bak 文件.

2）、还原：

要还原,首先需要有一个数据库和用户，最好是空库，否则会覆盖。

进入 postgresql 目录

cd /etc/postgresql/9.5/main

执行还原命令

psql -h 127.0.0.1 newdbUserName -d newdbName < /home/io/ databasename.bak

解释：

127.0.0.1 是数据库的 ip

newdbUserName 是上一步创建的数据库用户 name

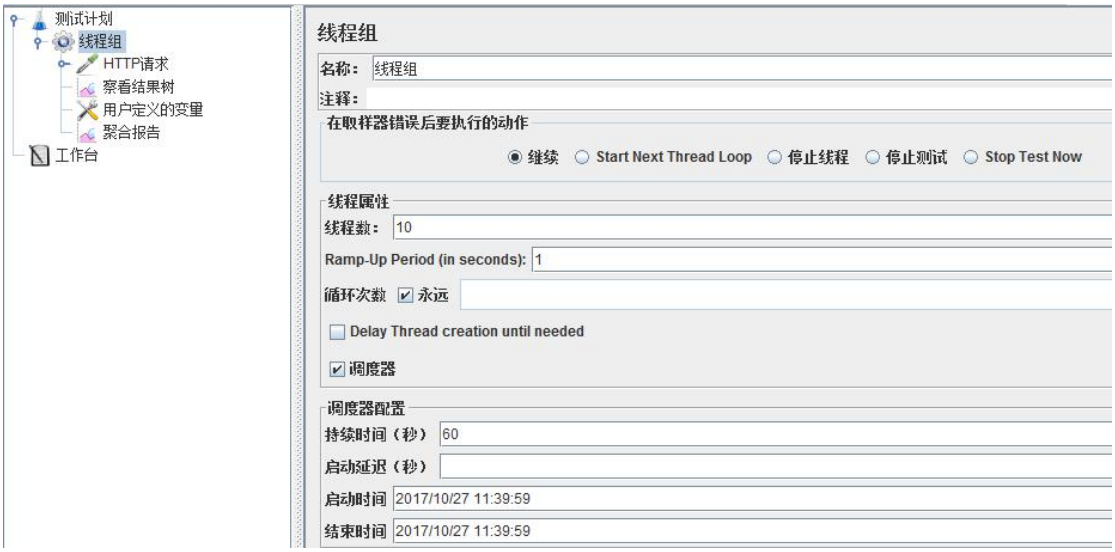
newdbName 是上一步创建的数据库名称

/home/io/ databasename.bak 是之前备份生成的文件.

5、jMeter 性能测试

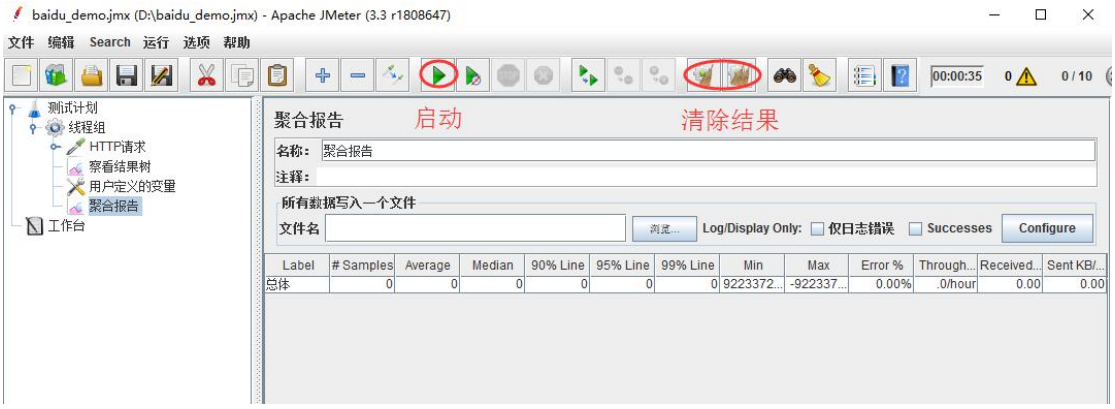
1)、配置线程组

点击线程组，配置本次性能测试相关参数：线程数，循环次数，持续时间等，这里我们配置并发用户数为 10，持续时间为 60s



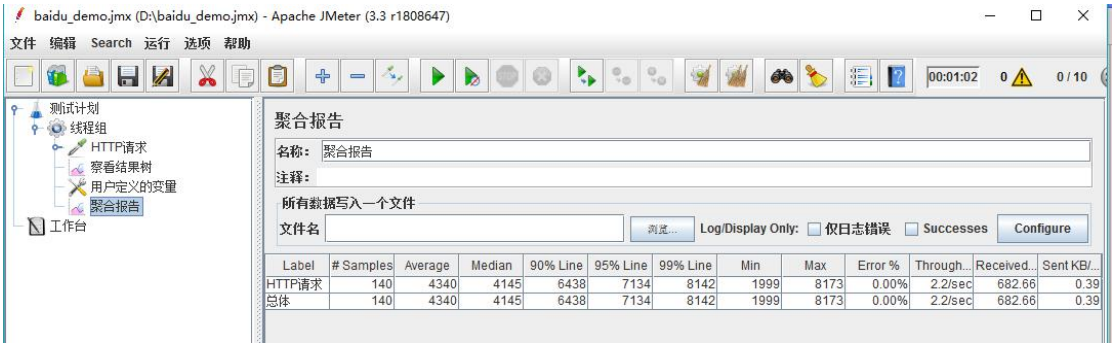
2）、执行测试

点击绿色小箭头按钮即可启动测试，测试之前需要点击小扫把按钮清除之前的调试结果。



3）、分析测试报告

待性能测试执行完成后，打开聚合报告可以看到：



聚合报告参数详解:

1. Label：每个 JMeter 的 element（例如 HTTP Request）都有一个 Name 属性，这里显示的就是 Name 属性的值
2. #Samples：请求数——表示这次测试中一共发出了多少个请求，如果模拟 10 个用户，每个用户迭代 10 次，那么这里显示 100
3. Average：平均响应时间——默认情况下是单个 Request 的平均响应时间，当使用了 Transaction Controller 时，以 Transaction 为单位显示平均响应时间
4. Median：中位数，也就是 50% 用户的响应时间
5. 90% Line：90% 用户的响应时间
6. Min：最小响应时间
7. Max：最大响应时间
8. Error%：错误率——错误请求数/请求总数
9. Throughput：吞吐量——默认情况下表示每秒完成的请求数（Request per Second），当使用了 Transaction Controller 时，也可以表示类似 LoadRunner

的 Transaction per Second 数。
10. KB/Sec：每秒从服务器端接收到的数据量，相当于 LoadRunner 中的 Throughput/Sec

一般而言，性能测试中我们需要重点关注的数据有： #Samples 请求数，Average 平均响应时间，Min 最小响应时间，Max 最大响应时间，Error% 错误率及 Throughput 吞吐量。

6、备份上传数据文件

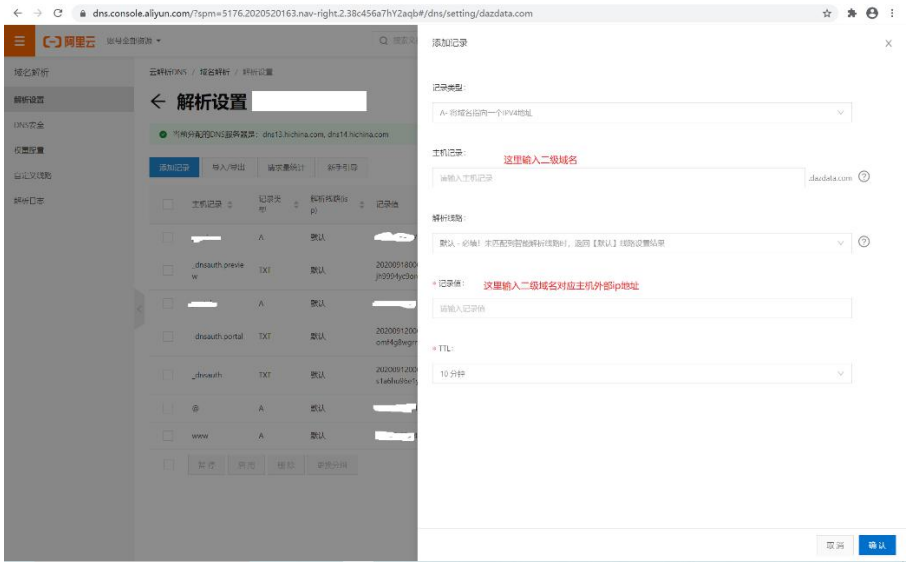
如果使用了 Excel 插件，且没有更改附件 5 所说的环境变量，则本地上传的文件位于默认的前端资源文件夹~/redash/client/dist/files/文件夹下，如果进行重新安装、升级安装，执行 npm run build 命令（该命令会清除~/redash/client/dist/下的所有资源文件）前，请务必备份该文件夹的所有上传数据文件，在重安装后再往回复制该文件夹即可。

第六章：阿里云部署

1、准备：

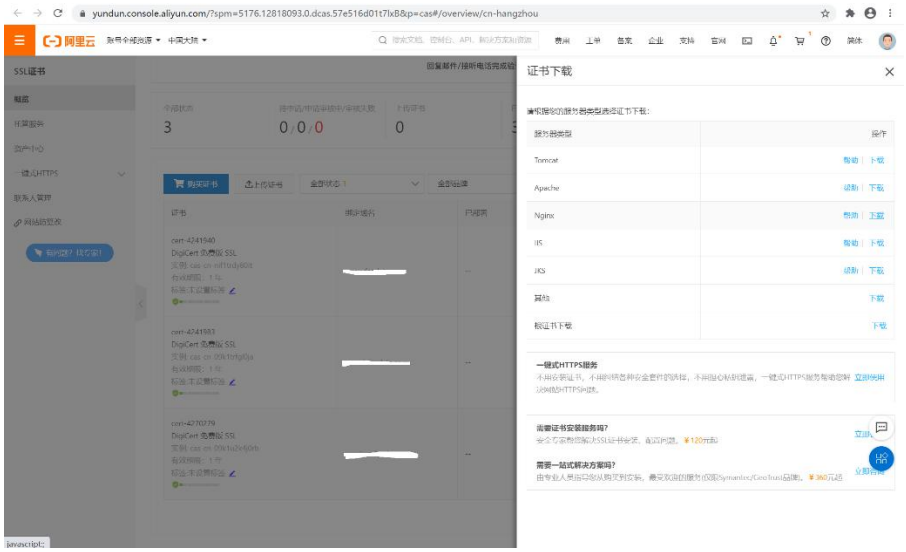
阿里云环境部署和 Ubuntu 或 Centos 部署没有什么不同，如果部署后只通过 IP 地址访问，基本等同于本地部署或本地生产环境部署。但如果部署后要进行外网域名绑定和配置 https 协议等，那还涉及很多外围准备工作：二级域名解析以及 IP 地址绑定、SSL 证书申请以及 Nginx 配置等。

2、二级域名解析：



配置好二级域名解析至新的服务器。

3、SSL 证书：



下载证书上传至目标服务器/etc/nginx/ssl 下待用。

4、阿里云服务器 Nginx 配置示例：

1）、配置 SSL 协议：

```
server {
    listen          443 ssl http2 default_server;
    server_name    domain.com portal.dazdata.com;
    root /usr/share/nginx/html;

    ssl_certificate /etc/nginx/ssl/4489877_portal.dazdata.com.pem;
    ssl_certificate_key /etc/nginx/ssl/4489877_portal.dazdata.com.key;

    location /static/ {
        alias /root/redash/client/dist/;
    }

    location / {
        root html;
        proxy_pass http://127.0.0.1:5000;
    }
}
```

其中涉及 hpps 协议默认端口 443 和 ssl 和 http2,以及证书上传位置。如果是 centos，还需要将 nginx.conf 里的 user=nginx 改为 user=root

2）、配置压缩传输

```
##
# Gzip Settings
##

gzip on;

gzip_vary on;
gzip_proxied any;
gzip_comp_level 6;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript;
```

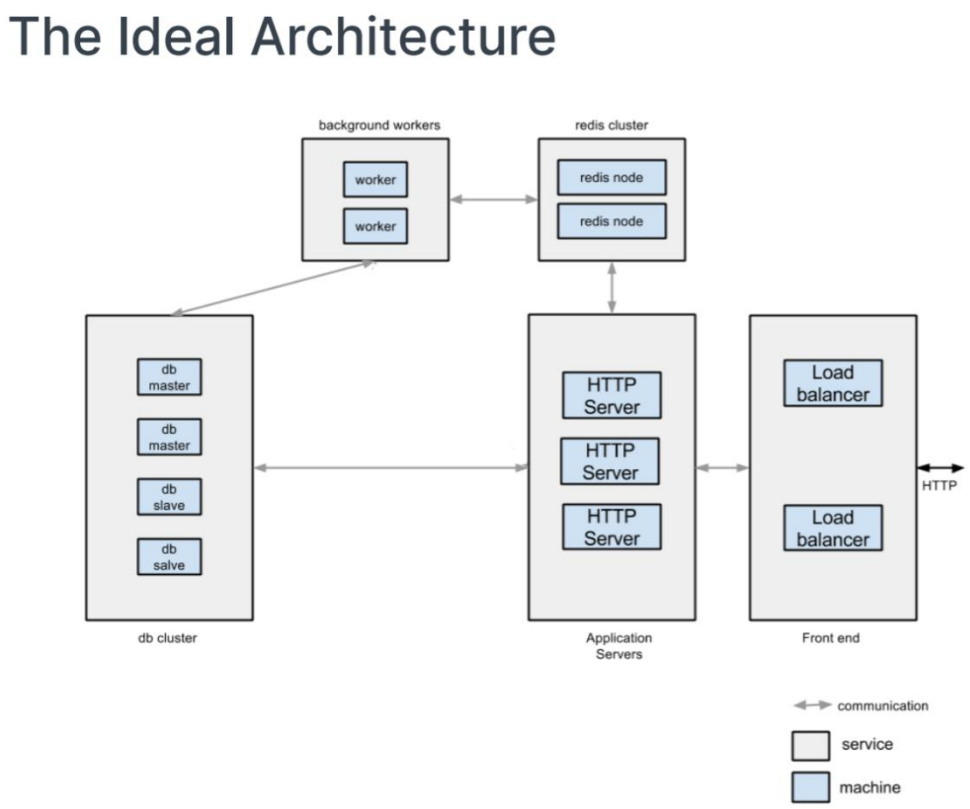
5、注意事项：

阿里云 ECS 服务器默认用户为 root，其主文件夹~为/root/；不同于其它系统的非 root 默认用户，主文件夹~为/home/用户名/；所以路径配置信息略有不同。另外，阿里云 postgresql 创建角色也必须为 root，才能正常登录。

```
[uwsgi]
http=:5000
chdir=/root/dazdata/
wsgi-file=redash/wsgi.py
callable=app
master=true
virtualenv=/root/dazdata/venv/
pythonpath=/root/dazdata/
processes=3
threads=6
```

第七章：PC 集群部署

1、Redash 理想部署方案



Redash 高可用理想部署架构包括：

Nginx 负载均衡集群：Nginx 负载均衡集群部署。

Web 服务器集群：Redash server 的平行集群。

postgres 的数据库集群：包括多种方案可选择，常用的是 Citus 方式。

redis 的数据库集群：包括多种方案可选择，常用的是主从方式。Redash 只是使用 Redis 的发布订阅队列服务，没有业务数据存储于 redis，队列任务如果没有由于 redash 当机等原因没有成功执行，系统会重试发送任务。故此，Redash 不要求 redis 高可用架构。

Worker 集群：Redash 的 worker 集群，含一台 scheduler。

2、四机分布部署

四台服务器做小型的分布式部署，推荐如下：

服务器	内容	配置
服务器 1：nginx	1. Nginx、 2. 前端打包内容	1、反向代理 2、配置 redash 动静分离， 3、负载均衡
服务器 2：redash 主服务器	1. Redash server 程序， 2. woker 程序， 3. scheduler 任务调度程序 4. Uwsgi 服务器 5. supervisor 进程调度程序	1. 按照生产部署步骤安装配置 uwsgi 服务器和 supervisor 进程调度. 2. 配置 redis 和 postgresql 的连接字符串.
服务器 3：redash 副服务器	1. Redash server 程序， 2. Woker 程序 3. Uwsgi 服务器 4. supervisor 进程调度程序	1、按照步骤 uswgi 和 supervisor. 不同是这回不配置 scheduler. 2、配置 redis 和 postgres 的连接字符串。
服务器 4: postgresql 数据库 Redis 数据库	1. postgresql 数据库 2. Redis 数据库	1. 开放相应端口和 ip 2. 关闭防火墙

四机分布部署是将来升级集群部署的基础。

(1) Redis 连接字符串：**默认值 redis://localhost:6379/0，含义：本机、6379 端口、使用 0 库(Redis 公用 0-15 共 16 个库，一般用 0 库)**

分布部署只需要更改 localhost 为上述服务器 2 的内网 ip 地址。例：在/etc/profile 或~/.bashrc 结尾增加一行：

export `REDASH_REDIS_URL="redis://192.168.0.2:6379/0"`

(2) Postgresql 连接字符串：**默认值 postgresql://postgres，含义本机、postgres 实例，使用 linux 当前用户作为数据库角色登录，本机登录无需密码。该角色必须具有登录权限，前面的安装里有执行这部分 SQL 脚本。这个是本机连接的简化连接字符串。分布连接字符串格式：**

postgresql://username:password@hostname/database

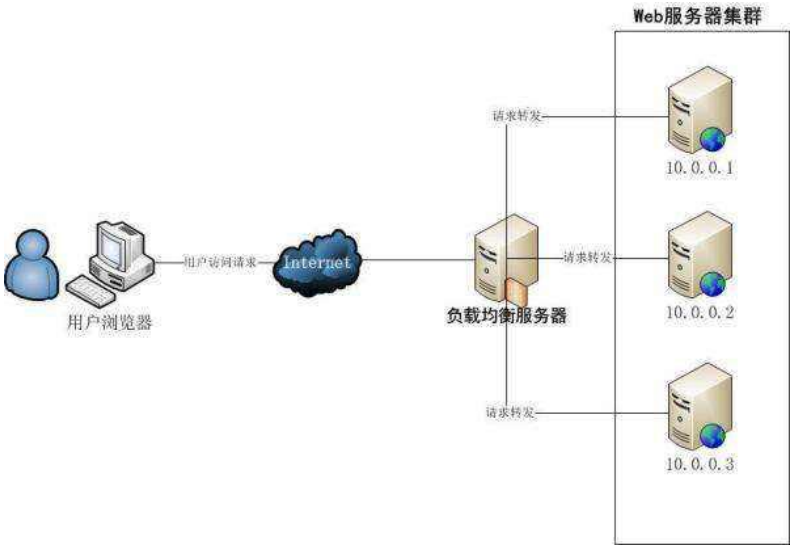
这里的用户名密码指的是数据库服务器的 postgresql 里设置的角色和密码，按教程安装一般有两个角色，一个是 postgres 密码 redash 能远程登录，第二个角色为服务器 2 的登录用户名，没有密码。Postgres 要求远程连接必须要有密码，故角色 2 只能本地登录，当然完全可以自行增加 postgres 的角色，或者设置角色 2 的密码使其那能远程登录。这里选择第一个角色和密码，

分布部署要更改连接字符串，例：在/etc/profile 或~/.bashrc 结尾增加一行：

export `REDASH_DATABASE_URL="postgresql://postgres:redash@192.168.0.2/postgres"`

或 export `DATABASE_URL="postgresql://postgres:redash@192.168.0.2/postgres"`**均可**

3、Nginx 集群



nginx 的负载均衡用于 upstream 模板定义的后端服务器列表中选取一台服务器接收用户的请求。一个基本的 upstream 模块如下：

```
upstream [服务器组名称]{
    server [IP 地址]:[端口号];
    server [IP 地址]:[端口号];
    ....
}
```

在 upstream 模块配置完成后，要让指定的访问反向代理到服务器列表，格式如下：

```
location ~ .*$ {
    index index.jsp index.html;
    proxy_pass http://[服务器组名称];
}
```

扩展：nginx 的 location 配置规则：
http://outofmemory.cn/code-snippet/742/nginx-location-configuration-xiangxi-explain

这样就完成了最基本的负载均衡，但是这并不能满足实际需求。目前 Nginx 的 upstream 模块支持 6 种方式的负载均衡策略（算法）：轮询（默认方式）、weight（权重方式）、ip_hash（依据 ip 分配方式）、least_conn（最少连接方式）、fair（第三方提供的响应时间方式）、url_hash（第三方通过的依据 URL 分配方式）。

4、uWSGI 配置 Server 进程线程

在保证其他条件不变的情况下(cpu、内存、访问接口的进程、相同的带宽等)，改变 processes 和 thread 参数，然后访问量相比：

processes	thread	访问量(每分钟)
8	100	2800-3000
8	40	2000-2600
30	40	13000-14000
40	40	17000-18000

结论：processes 和 thread 都会影响 uwsgi 的性能，其中 processes 的影响因素比较大，建议根据服务器的性能负载来增加(主要是 cpu 和内存，然后带宽也会影响)。Processes 推荐配置为 CPU 核心数*2-1。

5、Supervisor 配置 Worker 多进程

Redash 的 Woker 任务执行进程负责执行数据源取数，其集群部署是通过 Supervisor 进程守护来实现的。

worker.conf 文件

```
[supervisord]
logfile=/dev/null
pidfile=/tmp/supervisord.pid
nodaemon=true


[unix_http_server]
file = /tmp/supervisor.sock


[rpcinterface:supervisor]
supervisor.rpcinterface_factory =
supervisor.rpcinterface:make_main_rpcinterfac
e


[program:worker]
command=./manage.py rq worker %(ENV_QUEUES)s
process_name=%(program_name)s-%(process_num)s
numprocs=%(ENV_WORKERS_COUNT)s
directory=/app
```

```
stopsignal=TERM
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[eventlistener:worker_healthcheck]
serverurl=AUTO
command=./manage.py rq healthcheck
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
events=TICK_60
```

6、Postgres 主从备份

1）、分别在 master 节点和 standby 节点安装 Postgresql

2)、Master 上的配置：

（1）创建备份用户：

create role rep1 login replication encrypted password ‘passwd’;

（2）修改 pg_hba.conf

host　　replication　　all　　127.0.0.1/32　　trust

host　　replication　　rep1　　192.168.108.76/32　　md5

（3）修改 postgresql.conf

Listen_adresses = ‘*’

wal_level = replica　　模式选择

max_wal_senders=2　　流复制允许连接进程

wal_keep_segments =64

max_connections = 100　　数据库的连接数

（4）重启 master 数据库服务

pg_ctl start -D $PGDATA

3)、Standby 上的配置：

（1）基础备份：若从库已经初始化，则删除其 data 目录下文件：

pg_basebackup -h 192.168.108.75 -U rep1 -D $PGDATA -X stream -P

（2）修改 postgresql.conf 文件

关闭：wal_level、max_wal_senders、wal_keep_segments 等参数

打开如下参数：

hot_standby = on　　在备份的同时允许查询

max_standby_streaming_delay = 30s 可选，流复制最大延迟

wal_receiver_status_interval = 10s　　可选，从向主报告状态的最大间隔时间

hot_standby_feedback = on 可选，查询冲突时向主反馈

（3）修改 recovery.conf 文件

cp share/recovery.conf.sample data/recovery.conf

vi recovery.conf

recovery_target_timeline = 'latest'　　同步到最新数据

standby_mode = on　　指明从库身份

primary_conninfo = 'host=192.168.108.75 port=5432 user=rep1 password=passwd'　连接主库信息

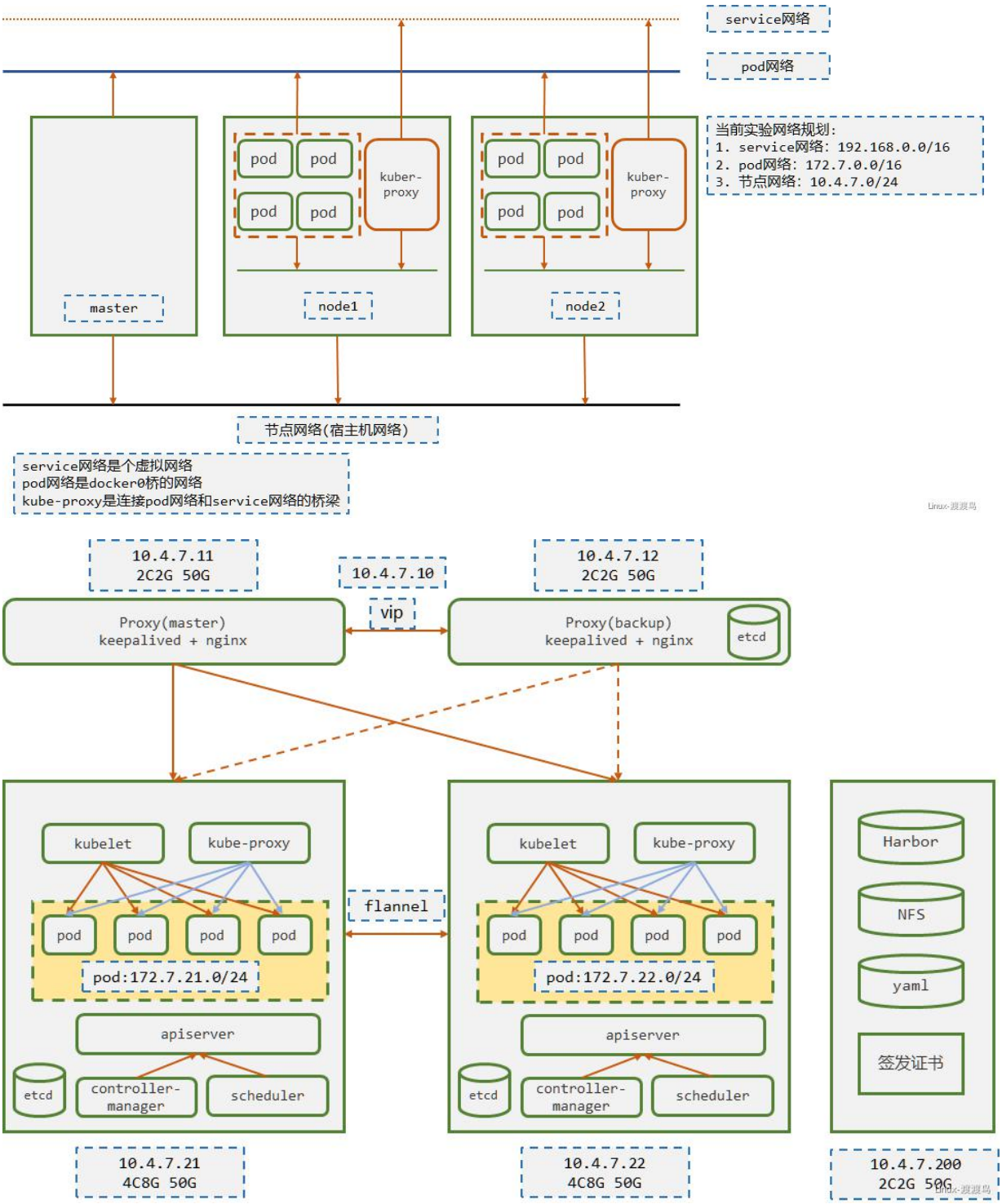
（4）启动从库

pg_ctl -D $PGDATA start

第八章：K8S 集群部署

1. 二进制方式安装

安装示例:



Role	HOSTNAME	IP	CPU	MEM	OS	DISK
LB,DNS	hdss7-11.host.com	10.4.7.11	2C	2G	Centos7.5	/data/ 50G
LB,ETCD	hdss7-12.host.com	10.4.7.12	2C	2G	Centos7.5	/data/ 50G
K8S Master,K8S Node,ETCD	hdss7-21.host.com	10.4.7.21	4C	8G	Centos7.5	/data/ 50G
K8S Master,K8S Node,ETCD	hdss7-22.host.com	10.4.7.21	4C	8G	Centos7.5	/data/ 50G
Harbor,NFS	hdss7-200.host.com	10.4.7.200	2C	2G	Centos7.5	/data/ 50G

2. 安装前准备

2.1、环境准备

所有机器都需要执行

```
1. systemctl stop firewalld
2. systemctl disable firewalld
3. setenforce 0
4. sed -ir '/^SELINUX=/s/=.+/=disabled/' /etc/selinux/config
5. yum install -y epel-release
6. yum install -y wget net-tools telnet tree nmap sysstat lrzsz dos2unix bind-utils vim less
```

2.2、bind 安装

1).hdss7-11 安装 bind

```
1. yum install -y bind
```

2).hdss7-11 配置 bind

* 主配置文件

```
[root@hdss7-11 ~]# vim /etc/named.conf  # 确保以下配置正确
  listen-on port 53 { 10.4.7.11; };
  directory   "/var/named";
  allow-query     { any; };
  forwarders      { 10.4.7.254; };
  recursion yes;
  dnssec-enable no;
  dnssec-validation no;
```

* 在 hdss7-11.host.com 配置区域文件

```
# 增加两个zone配置，od.com为业务域，host.com.zone为主机域
[root@hdss7-11 ~]# vim /etc/named.rfc1912.zones
zone "host.com" IN {
        type  master;
        file  "host.com.zone";
        allow-update { 10.4.7.11; };
};

zone "od.com" IN {
        type  master;
        file  "od.com.zone";
        allow-update { 10.4.7.11; };
};
```

* 在 hdss7-11.host.com 配置主机域文件

```
# line6中时间需要修改
[root@hdss7-11 ~]# vim /var/named/host.com.zone
$ORIGIN host.com.
$TTL 600  ; 10 minutes
@       IN SOA  dns.host.com. dnsadmin.host.com. (
        2020010501 ; serial
        10800      ; refresh (3 hours)
        900        ; retry (15 minutes)
        604800     ; expire (1 week)
        86400      ; minimum (1 day)
        )
      NS   dns.host.com.
$TTL 60 ; 1 minute
dns                A    10.4.7.11
HDSS7-11           A    10.4.7.11
HDSS7-12           A    10.4.7.12
HDSS7-21           A    10.4.7.21
HDSS7-22           A    10.4.7.22
HDSS7-200          A    10.4.7.200
```

* 在 hdss7-11.host.com 配置业务域文件

```
[root@hdss7-11 ~]# vim /var/named/od.com.zone
$ORIGIN od.com.
$TTL 600  ; 10 minutes
@       IN SOA  dns.od.com. dnsadmin.od.com. (
        2020010501 ; serial
        10800      ; refresh (3 hours)
        900        ; retry (15 minutes)
        604800     ; expire (1 week)
        86400      ; minimum (1 day)
        )
        NS   dns.od.com.
$TTL 60 ; 1 minute
dns                A    10.4.7.11
```

* 在 hdss7-11.host.com 启动 bind 服务，并测试

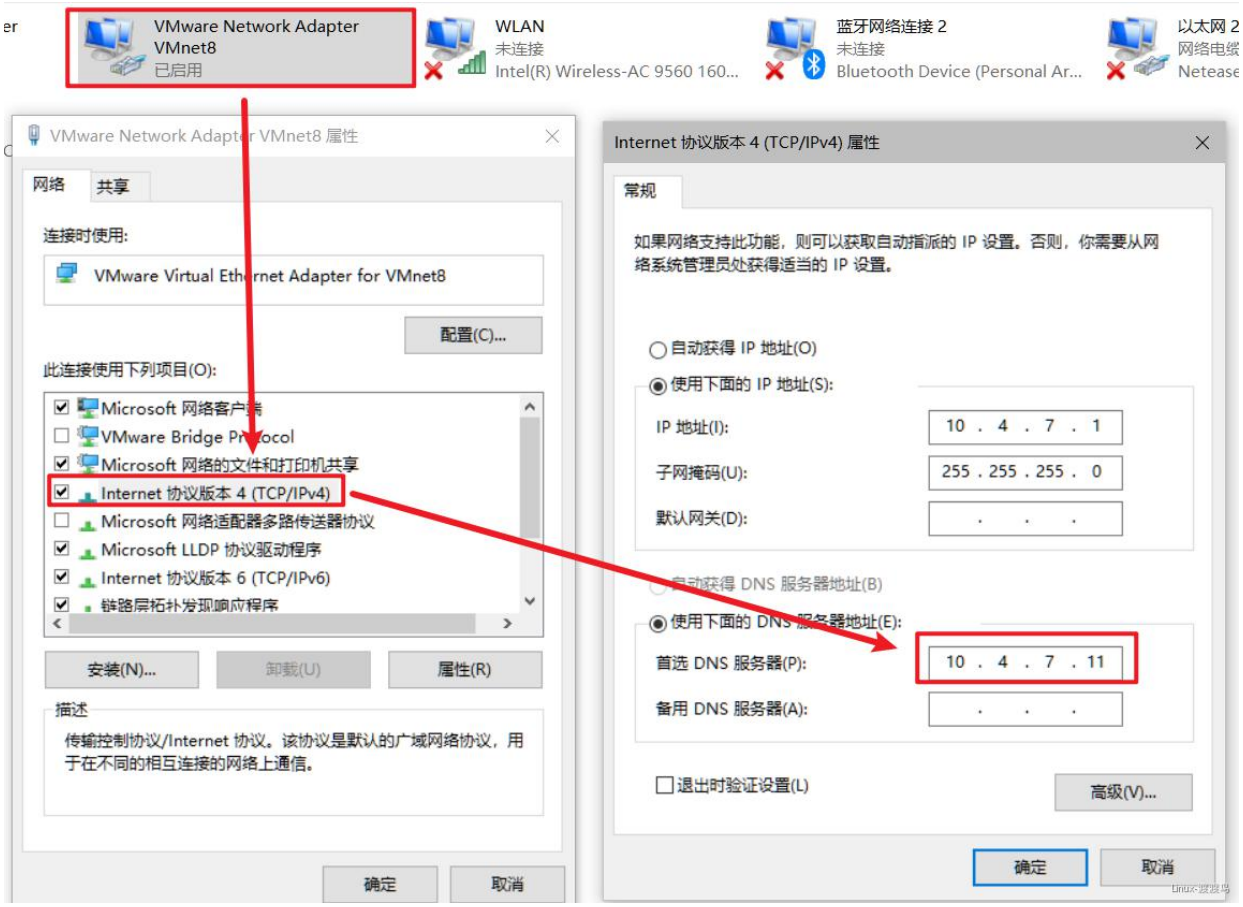
```
[root@hdss7-11 ~]# named-checkconf  # 检查配置文件
[root@hdss7-11 ~]# systemctl start named ; systemctl enable named
[root@hdss7-11 ~]# host HDSS7-200 10.4.7.11
Using domain server:
Name: 10.4.7.11
Address: 10.4.7.11#53
Aliases:

HDSS7-200.host.com has address 10.4.7.200
```

3). 修改主机 DNS

* 修改所有主机的 dns 服务器地址

```
[root@hdss7-11 ~]# sed -i '/DNS1/s/10.4.7.254/10.4.7.11/' /etc/sysconfig/network-scripts/ifcfg-ens32
[root@hdss7-11 ~]# systemctl restart network
（centos8 使用 service NetworkManager start 重启网络服务）
[root@hdss7-11 ~]# cat /etc/resolv.conf
# Generated by NetworkManager
search host.com
nameserver 10.4.7.11
```

* 本次实验环境使用的是虚拟机，因此也要对 windows 宿主机 NAT 网卡 DNS 进行修改




```
C:\Users\msi->ping hdss7-200.host.com

正在 Ping HDSS7-200.host.com [10.4.7.200] 具有 32 字节的数据:
来自 10.4.7.200 的回复: 字节=32 时间<1ms TTL=64
来自 10.4.7.200 的回复: 字节=32 时间<1ms TTL=64
```

2.3、根证书准备

* 在 hdss7-200 下载工具

1. wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64 -O /usr/local/bin/cfssl
2. wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64 -O /usr/local/bin/cfssl-json
3. wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64 -O /usr/local/bin/cfssl-certinfo
4. chmod u+x /usr/local/bin/cfssl*

* 在 hdss7-200 签发根证书

```
[root@hdss7-200 ~]# mkdir /opt/certs/ ; cd /opt/certs/
# 根证书配置：
# CN 一般写域名，浏览器会校验
# names 为地区和公司信息
# expiry 为过期时间
[root@hdss7-200 certs]# vim /opt/certs/ca-csr.json
{
    "CN": "OldboyEdu",
    "hosts": [
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "beijing",
            "L": "beijing",
            "O": "od",
            "OU": "ops"
        }
    ],
    "ca": {
        "expiry": "175200h"
    }
}
[root@hdss7-200 certs]# cfssl gencert -initca ca-csr.json | cfssl-json -bare ca
2020/01/05 10:42:07 [INFO] generating a new CA key and certificate from CSR
2020/01/05 10:42:07 [INFO] generate received request
2020/01/05 10:42:07 [INFO] received CSR
2020/01/05 10:42:07 [INFO] generating key: rsa-2048
2020/01/05 10:42:08 [INFO] encoded CSR
2020/01/05 10:42:08 [INFO] signed certificate with serial number 451005524427475354617025360
[root@hdss7-200 certs]# ls -l ca*
-rw-r--r-- 1 root root  993 Jan  5 10:42 ca.csr
-rw-r--r-- 1 root root  328 Jan  5 10:39 ca-csr.json
-rw------- 1 root root 1675 Jan  5 10:42 ca-key.pem
-rw-r--r-- 1 root root 1346 Jan  5 10:42 ca.pem
```

2.4、docker 环境准备

需要安装 docker 的机器：hdss7-21 hdss7-22 hdss7-200，以 hdss7-21 为例

1. wget -O /etc/yum.repos.d/docker-ce.repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
2. yum install https://download.docker.com/linux/fedora/30/x86_64/stable/Packages/containerd.io-1.2.6-3.3.fc30.x86_64.rpm
3. yum install -y yum-utils device-mapper-persistent-data lvm2
4. yum install -y docker-ce
5. mkdir /etc/docker/

不安全的 registry 中增加了 harbor 地址

各个机器上 bip 网段不一致，bip 中间两段与宿主机最后两段相同，目的是方便定位问题

6. vim /etc/docker/daemon.json

```
{
  "graph": "/data/docker",
  "storage-driver": "overlay2",
  "insecure-registries": ["registry.access.redhat.com","quay.io","harbor.od.com"],
  "registry-mirrors": ["https://registry.docker-cn.com"],
  "bip": "172.7.21.1/24",
  "exec-opts": ["native.cgroupdriver=systemd"],
  "live-restore": true
}
```

7. mkdir /root/data/docker
8. systemctl start docker ; systemctl enable docker

2.5、harbor 安装

参考地址：https://www.yuque.com/duduniao/trp3ic/ohrxds#9Zpxx
官方地址：https://goharbor.io/
下载地址：https://github.com/goharbor/harbor/releases

如果 yum install -y docker-compose 出现错误请更换 docker 的源参照 https://blog.csdn.net/qq_32828933/article/details/104220570 进行安装

```
# 目录说明：
# /opt/src ： 源码、文件下载目录
# /opt/release ： 各个版本软件存放位置
# /opt/apps ： 各个软件当前版本的软链接
[root@hdss7-200 ~]# cd /opt/src
[root@hdss7-200 src]# wget https://github.com/goharbor/harbor/releases/download/v1.9.4/har
[root@hdss7-200 src]# mv harbor /opt/release/harbor-v1.9.4
[root@hdss7-200 src]# ln -s /opt/release/harbor-v1.9.4 /opt/apps/harbor
[root@hdss7-200 src]# ll /opt/apps/
total 0
lrwxrwxrwx 1 root root 26 Jan  5 11:13 harbor -> /opt/release/harbor-v1.9.4
# 实验环境仅修改以下配置项，生产环境还得修改密码
[root@hdss7-200 src]# vim /opt/apps/harbor/harbor.yml
hostname: harbor.od.com
http:
  port: 180
data_volume: /data/harbor
location: /data/harbor/logs
[root@hdss7-200 src]# yum install -y docker-compose
[root@hdss7-200 src]# cd /opt/apps/harbor/
[root@hdss7-200 harbor]# ./install.sh
```

* 设置 harbor 开机启动

```
[root@hdss7-200 harbor]# vim /etc/rc.d/rc.local  # 增加以下内容
# start harbor
cd /opt/apps/harbor
/usr/bin/docker-compose stop
/usr/bin/docker-compose start
```

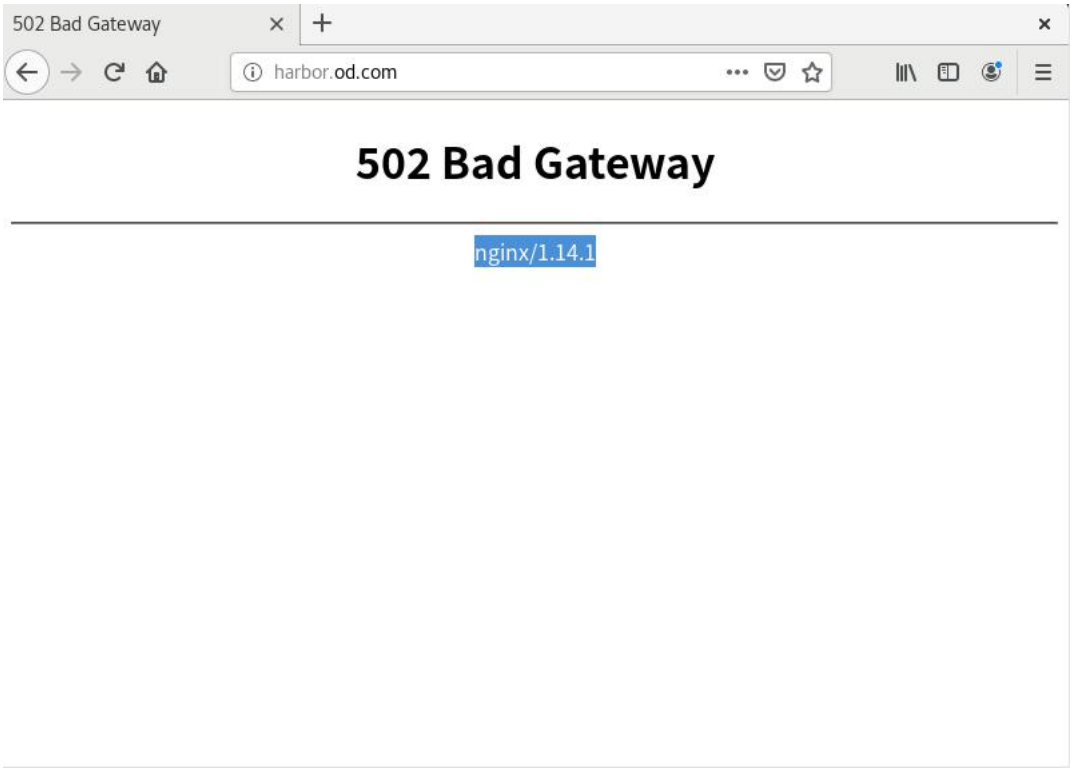
* hdss7-200 安装 nginx
* 安装 nginx 反向代理 harbor

```
# 当前机器中Nginx功能较少，使用yum安装即可。如有多个harbor考虑源码编译且配置健康检查
# nginx配置此处忽略，仅仅使用最简单的配置。
[root@hdss7-200 harbor]# vim /etc/nginx/conf.d/harbor.conf
[root@hdss7-200 harbor]# cat /etc/nginx/conf.d/harbor.conf
server {
    listen       80;
    server_name  harbor.od.com;
    # 避免出现上传失败的情况
    client_max_body_size 1000m;

    location / {
        proxy_pass http://127.0.0.1:180;
    }
}
[root@hdss7-200 harbor]# systemctl start nginx ; systemctl enable nginx
```

* hdss7-11 配置 DNS 解析

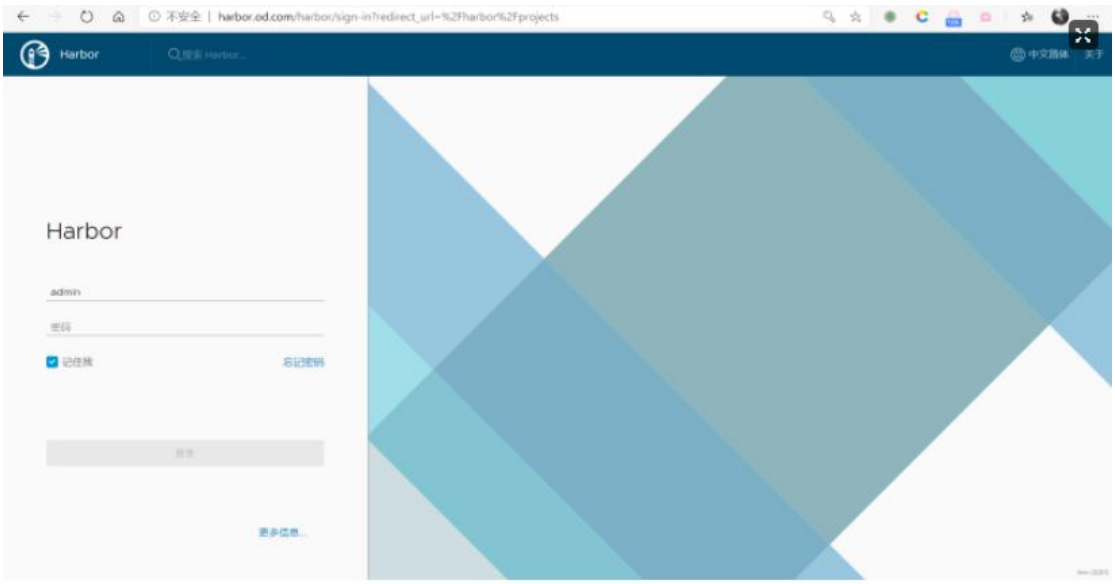
```
[root@hdss7-11 ~]# vim /var/named/od.com.zone  # 序列号需要滚动一个
$ORIGIN od.com.
$TTL 600  ; 10 minutes
@       IN SOA  dns.od.com. dnsadmin.od.com. (
        2020010502 ; serial
        10800      ; refresh (3 hours)
        900        ; retry (15 minutes)
        604800     ; expire (1 week)
        86400      ; minimum (1 day)
        )
        NS   dns.od.com.
$TTL 60 ; 1 minute
dns                A    10.4.7.11
harbor             A    10.4.7.200
[root@hdss7-11 ~]# systemctl restart named.service  # reload 无法使得配置生效
[root@hdss7-11 ~]# host harbor.od.com
harbor.od.com has address 10.4.7.200
```

然后就输入：harbor.od.com 进行访问如果出现以下情况：



请修改/etc/selinux/config 这个文件将 SELINUX=disabled 即可。

```
[root@node05 conf.d]# cat /etc/selinux/config

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```



* 新建项目：public

* 测试 harbor

如果缺少镜像请执行 docker pull nginx:latest 拉取

```
[root@hdss7-21 ~]# docker image tag nginx:latest harbor.od.com/public/nginx:latest
[root@hdss7-21 ~]# docker login -u admin harbor.od.com
[root@hdss7-21 ~]# docker image push harbor.od.com/public/nginx:latest
[root@hdss7-21 ~]# docker logout
```

3. 主控节点安装

3.1、etcd 安装

etcd 的 leader 选举机制，要求至少为 3 台或以上的奇数台。本次安装涉及：hdss7-12，hdss7-21，hdss7-22

1. 签发 etcd 证书

证书签发服务器 hdss7-200:

* 创建 ca 的 json 配置:/opt/certs/ca-config.json

server 表示服务端连接客户端时携带的证书，用于客户端验证服务端身份

client 表示客户端连接服务端时携带的证书，用于服务端验证客户端身份

peer 表示相互之间连接时使用的证书，如 etcd 节点之间验证

```
{
    "signing": {
        "default": {
            "expiry": "175200h"
        },
        "profiles": {
            "server": {
                "expiry": "175200h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth"
                ]
            },
            "client": {
                "expiry": "175200h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "client auth"
                ]
            },
            "peer": {
                "expiry": "175200h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "client auth"
                ]
            }
        }
    }
}
```

* 创建 etcd 证书配置：/opt/certs/etcd-peer-csr.json

重点在 hosts 上，将所有可能的 etcd 服务器添加到 host 列表，不能使用网段，新增 etcd 服务器需要重新签发证书

```
{
    "CN": "k8s-etcd",
    "hosts": [
        "10.4.7.11",
        "10.4.7.12",
        "10.4.7.21",
        "10.4.7.22"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "beijing",
            "L": "beijing",
            "O": "od",
            "OU": "ops"
        }
    ]
}
```

* 签发证书

```
[root@hdss7-200 ~]# cd /opt/certs/
[root@hdss7-200 certs]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json  -profile=peer  etcd-peer-csr.json  |cfssl-json -bare etcd-peer
[root@hdss7-200 certs]# ll etcd-peer*
-rw-r--r-- 1 root root 1062 Jan  5 17:01 etcd-peer.csr
-rw-r--r-- 1 root root  363 Jan  5 16:59 etcd-peer-csr.json
-rw------- 1 root root 1675 Jan  5 17:01 etcd-peer-key.pem
-rw-r--r-- 1 root root 1428 Jan  5 17:01 etcd-peer.pem
```

2. 安装 etcd

etcd 地址：https://github.com/etcd-io/etcd/
实验使用版本：etcd-v3.1.20-linux-amd64.tar.gz
本次安装涉及：hdss7-12，hdss7-21，hdss7-22

* 下载 etcd

```
[root@hdss7-12 ~]# useradd -s /sbin/nologin -M etcd
[root@hdss7-12 ~]# cd /opt/src/
[root@hdss7-12 src]# wget https://github.com/etcd-io/etcd/releases/download/v3.1.20/etcd-v3.1.20-linux-amd64.tar.gz
```

```
[root@hdss7-12 src]# tar -xf etcd-v3.1.20-linux-amd64.tar.gz
[root@hdss7-12 src]# mv etcd-v3.1.20-linux-amd64
/opt/release/etcd-v3.1.20
[root@hdss7-12 src]# ln -s /opt/release/etcd-v3.1.20
/opt/apps/etcd
[root@hdss7-12 src]# ll /opt/apps/etcd
lrwxrwxrwx 1 root root 25 Jan  5 17:56 /opt/apps/etcd ->
/opt/release/etcd-v3.1.20
[root@hdss7-12 src]# mkdir -p /opt/apps/etcd/certs /data/etcd
/data/logs/etcd-server
```

* 下发证书到各个 etcd 上

```
[root@hdss7-200 ~]# cd /opt/certs/
[root@hdss7-200 certs]# for i in 12 21 22;do scp ca.pem etcd-peer.pem
etcd-peer-key.pem hdss7-${i}:/opt/apps/etcd/certs/ ;done
[root@hdss7-12 src]# md5sum
/opt/apps/etcd/certs/*8778d0c3411891af61a287e49a70c89a
/opt/apps/etcd/certs/ca.pem7918783c2f6bf69e96edf03e67d04983
/opt/apps/etcd/certs/etcd-peer-key.pemd4d849751a834c7727d42324fdedf9
2d  /opt/apps/etcd/certs/etcd-peer.pem
```

* 创建启动脚本(部分参数每台机器不同)

```
[root@hdss7-12 ~]# vim /opt/apps/etcd/etcd-server-startup.sh
#!/bin/sh
# listen-peer-urls etcd 节点之间通信端口
# listen-client-urls 客户端与 etcd 通信端口
# quota-backend-bytes 配额大小
# 需要修改的参数：
name,listen-peer-urls,listen-client-urls,initial-advertise-peer-urls
WORK_DIR=$(dirname $(readlink -f $0))
[ $? -eq 0 ] && cd $WORK_DIR || exit
/opt/apps/etcd/etcd --name etcd-server-7-12 \
            --data-dir /data/etcd/etcd-server \
            --listen-peer-urls https://10.4.7.12:2380 \
            --listen-client-urls
    https://10.4.7.12:2379,http://127.0.0.1:2379 \
            --quota-backend-bytes 8000000000 \
            --initial-advertise-peer-urls https://10.4.7.12:2380
    \
            --advertise-client-urls
    https://10.4.7.12:2379,http://127.0.0.1:2379 \
            --initial-cluster
    etcd-server-7-12=https://10.4.7.12:2380,etcd-server-7-21=htt
    ps://10.4.7.21:2380,etcd-server-7-22=https://10.4.7.22:2380 \
```

```
                --ca-file ./certs/ca.pem \
                --cert-file ./certs/etcd-peer.pem \
                --key-file ./certs/etcd-peer-key.pem \
                --client-cert-auth  \
                --trusted-ca-file ./certs/ca.pem \
                --peer-ca-file ./certs/ca.pem \
                --peer-cert-file ./certs/etcd-peer.pem \
                --peer-key-file ./certs/etcd-peer-key.pem \
                --peer-client-cert-auth \
                --peer-trusted-ca-file ./certs/ca.pem \
            --log-output stdout
[root@hdss7-12 ~]# chmod u+x
      /opt/apps/etcd/etcd-server-startup.sh
[root@hdss7-12 ~]# chown -R etcd.etcd /opt/apps/etcd/ /data/etcd
       /data/logs/etcd-server
```

3.启动 etcd

因为这些进程都是要启动为后台进程，要么手动启动，要么采用后台进程管理工具，实验中使用后台管理工具

```
[root@hdss7-12 ~]# yum install -y supervisor
[root@hdss7-12 ~]# systemctl start supervisord ; systemctl enable supervisord
[root@hdss7-12 ~]# vim /etc/supervisord.d/etcd-server.ini
[program:etcd-server-7-12]
    command=/opt/apps/etcd/etcd-server-startup.sh          ; the program (relative uses PATH, can take args)
    numprocs=1                                             ; number of processes copies to start (def 1)
    directory=/opt/apps/etcd                               ; directory to cwd to before exec (def no cwd)
    autostart=true                                         ; start at supervisord start (default: true)
    autorestart=true                                       ; retstart at unexpected quit (default: true)
    startsecs=30                                           ; number of secs prog must stay running (def. 1)
    startretries=3                                         ; max # of serial start failures (default 3)
    exitcodes=0,2                                          ; 'expected' exit codes for process (default 0,2)
    stopsignal=QUIT                                        ; signal used to kill process (default TERM)
```

```
    stopwaitsecs=10                                                 ;
max num secs to wait b4 SIGKILL (default 10)
    user=etcd                                                       ;
setuid to this UNIX account to run the program
    redirect_stderr=true                                            ;
redirect proc stderr to stdout (default false)
    stdout_logfile=/data/logs/etcd-server/etcd.stdout.log ;
stdout log path, NONE for none; default AUTO
    stdout_logfile_maxbytes=64MB                                    ;
max # logfile bytes b4 rotation (default 50MB)
    stdout_logfile_backups=5                                        ; #
of stdout logfile backups (default 10)
    stdout_capture_maxbytes=1MB                                     ;
number of bytes in 'capturemode' (default 0)
    stdout_events_enabled=false                                     ;
emit events on stdout writes (default false)
[root@hdss7-12 ~]# supervisorctl update
    etcd-server-7-12: added process group
```

* etcd 进程状态查看

```
[root@hdss7-12 ~]# supervisorctl status  # supervisorctl 状态
    etcd-server-7-12                 RUNNING    pid 22375,
uptime 0:00:39

[root@hdss7-12 ~]# netstat -lntp|grep etcd
    tcp         0       0 10.4.7.12:2379          0.0.0.0:*
LISTEN      22379/etcd
    tcp         0       0 127.0.0.1:2379          0.0.0.0:*
LISTEN      22379/etcd
    tcp         0       0 10.4.7.12:2380          0.0.0.0:*
LISTEN      22379/etcd

[root@hdss7-12 ~]# /opt/apps/etcd/etcdctl member list # 随着 etcd
重启，leader 会变化
    988139385f78284: name=etcd-server-7-22
peerURLs=https://10.4.7.22:2380
clientURLs=http://127.0.0.1:2379,https://10.4.7.22:2379
isLeader=false
    5a0ef2a004fc4349: name=etcd-server-7-21
peerURLs=https://10.4.7.21:2380
clientURLs=http://127.0.0.1:2379,https://10.4.7.21:2379
isLeader=true
    f4a0cb0a765574a8: name=etcd-server-7-12
peerURLs=https://10.4.7.12:2380
```

```
clientURLs=http://127.0.0.1:2379,https://10.4.7.12:2379
isLeader=false

[root@hdss7-12 ~]# /opt/apps/etcd/etcdctl cluster-health
    member 988139385f78284 is healthy: got healthy result from
http://127.0.0.1:2379
    member 5a0ef2a004fc4349 is healthy: got healthy result from
http://127.0.0.1:2379
    member f4a0cb0a765574a8 is healthy: got healthy result from
http://127.0.0.1:2379
    cluster is healthy
```

* etcd 启停方式

```
[root@hdss7-12 ~]# supervisorctl start etcd-server-7-12
[root@hdss7-12 ~]# supervisorctl stop etcd-server-7-12
[root@hdss7-12 ~]# supervisorctl restart etcd-server-7-12
[root@hdss7-12 ~]# supervisorctl status etcd-server-7-12
```

3.2、apiserver 安装

1. 下载 kubernetes 服务端

aipserver 涉及的服务器：hdss7-21，hdss7-22 下载 kubernetes 二进制版本包需要科学上网工具

1. 进入 kubernetes 的 github 页面：
   https://github.com/kubernetes/kubernetes
2. 进入 tags 页签：https://github.com/kubernetes/kubernetes/tags
3. 选择要下载的版本：
   https://github.com/kubernetes/kubernetes/releases/tag/v1.15.2
4. 点击 CHANGELOG-${version}.md 进入说明页面：
   https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.15.md#downloads-for-v1152
5. 下载 Server Binaries：
   https://dl.k8s.io/v1.15.2/kubernetes-server-linux-amd64.tar.gz

```
[root@hdss7-21 ~]# cd /opt/src
[root@hdss7-21 src]# wget https://dl.k8s.io/v1.15.2/kubernetes-server-linux-amd64.tar.gz

[root@hdss7-21 src]# tar -xf kubernetes-server-linux-amd64.tar.gz
[root@hdss7-21 src]# mv kubernetes /opt/release/kubernetes-v1.15.2
[root@hdss7-21 src]# ln -s /opt/release/kubernetes-v1.15.2 /opt/apps/kubernetes
[root@hdss7-21 src]# ll /opt/apps/kubernetes
lrwxrwxrwx 1 root root 31 Jan  6 12:59 /opt/apps/kubernetes -> /opt/release/kubernetes-v1.15.2

[root@hdss7-21 src]# cd /opt/apps/kubernetes
[root@hdss7-21 kubernetes]# rm -f kubernetes-src.tar.gz
[root@hdss7-21 kubernetes]# cd server/bin/
[root@hdss7-21 bin]# rm -f *.tar *_tag  # *.tar *_tag 镜像文件
[root@hdss7-21 bin]# ll
total 884636
-rwxr-xr-x 1 root root  43534816 Aug  5 18:01 apiextensions-apiserver
-rwxr-xr-x 1 root root 100548640 Aug  5 18:01 cloud-controller-manager
-rwxr-xr-x 1 root root 200648416 Aug  5 18:01 hyperkube
-rwxr-xr-x 1 root root  40182208 Aug  5 18:01 kubeadm
-rwxr-xr-x 1 root root 164501920 Aug  5 18:01 kube-apiserver
-rwxr-xr-x 1 root root 116397088 Aug  5 18:01 kube-controller-manager
-rwxr-xr-x 1 root root  42985504 Aug  5 18:01 kubectl
-rwxr-xr-x 1 root root 119616640 Aug  5 18:01 kubelet
-rwxr-xr-x 1 root root  36987488 Aug  5 18:01 kube-proxy
-rwxr-xr-x 1 root root  38786144 Aug  5 18:01 kube-scheduler
-rwxr-xr-x 1 root root   1648224 Aug  5 18:01 mounter
```

2.签发证书

签发证书 涉及的服务器：hdss7-200

* 签发client证书(apiserver和etcd通信证书)

```
[root@hdss7-200 ~]# cd /opt/certs/
[root@hdss7-200 certs]# vim /opt/certs/client-csr.json
{
    "CN": "k8s-node",
    "hosts": [
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "beijing",
            "L": "beijing",
            "O": "od",
            "OU": "ops"
        }
```

```
        ]
    }
    [root@hdss7-200 certs]# cfssl gencert -ca=ca.pem
-ca-key=ca-key.pem -config=ca-config.json -profile=client
client-csr.json |cfssl-json -bare client
    2020/01/06 13:42:47 [INFO] generate received request
    2020/01/06 13:42:47 [INFO] received CSR
    2020/01/06 13:42:47 [INFO] generating key: rsa-2048
    2020/01/06 13:42:47 [INFO] encoded CSR
    2020/01/06 13:42:47 [INFO] signed certificate with serial
number 268276380983442021656020268926931973684313260543
    2020/01/06 13:42:47 [WARNING] This certificate lacks a
"hosts" field. This makes it unsuitable for
    websites. For more information see the Baseline
Requirements for the Issuance and Management
    of Publicly-Trusted Certificates, v.1.1.6, from the
CA/Browser Forum (https://cabforum.org);
    specifically, section 10.2.3 ("Information
Requirements").
    [root@hdss7-200 certs]# ls client* -l
    -rw-r--r-- 1 root root  993 Jan  6 13:42 client.csr
    -rw-r--r-- 1 root root  280 Jan  6 13:42 client-csr.json
    -rw------- 1 root root 1679 Jan  6 13:42 client-key.pem
    -rw-r--r-- 1 root root 1363 Jan  6 13:42 client.pem
```

* 签发 server 证书（apiserver 和其它 k8s 组件通信使用）

hosts 中将所有可能作为 apiserver 的 ip 添加进去，VIP 10.4.7.10 也要加入

```
[root@hdss7-200 certs]# vim /opt/certs/apiserver-csr.json
{
    "CN": "k8s-apiserver",
    "hosts": [
        "127.0.0.1",
        "192.168.0.1",
        "kubernetes.default",
        "kubernetes.default.svc",
        "kubernetes.default.svc.cluster",
        "kubernetes.default.svc.cluster.local",
        "10.4.7.10",
        "10.4.7.21",
        "10.4.7.22",
        "10.4.7.23"
    ],
    "key": {
```

```
            "algo": "rsa",
            "size": 2048
        },
        "names": [
            {
                "C": "CN",
                "ST": "beijing",
                "L": "beijing",
                "O": "od",
                "OU": "ops"
            }
        ]
    }
    [root@hdss7-200    certs]#    cfssl    gencert    -ca=ca.pem
-ca-key=ca-key.pem    -config=ca-config.json    -profile=server
apiserver-csr.json |cfssl-json -bare apiserver
    2020/01/06 13:46:56 [INFO] generate received request
    2020/01/06 13:46:56 [INFO] received CSR
    2020/01/06 13:46:56 [INFO] generating key: rsa-2048
    2020/01/06 13:46:56 [INFO] encoded CSR
    2020/01/06 13:46:56 [INFO] signed certificate with serial number
573076691386375893093727554861295529219004473872
    2020/01/06 13:46:56 [WARNING] This certificate lacks a "hosts"
field. This makes it unsuitable for
    websites. For more information see the Baseline Requirements for
the Issuance and Management
    of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser
Forum (https://cabforum.org);
    specifically, section 10.2.3 ("Information Requirements").
    [root@hdss7-200 certs]# ls apiserver* -l
    -rw-r--r-- 1 root root 1249 Jan  6 13:46 apiserver.csr
    -rw-r--r-- 1 root root  566 Jan  6 13:45 apiserver-csr.json
    -rw------- 1 root root 1675 Jan  6 13:46 apiserver-key.pem
    -rw-r--r-- 1 root root 1598 Jan  6 13:46 apiserver.pem

    * 证书下发
        [root@hdss7-200 certs]# for i in 21 22;do echo hdss7-$i;ssh
    hdss7-$i  "mkdir  /opt/apps/kubernetes/server/bin/certs";scp
    apiserver-key.pem    apiserver.pem    ca-key.pem    ca.pem
    client-key.pem                                    client.pem
    hdss7-$i:/opt/apps/kubernetes/server/bin/certs/;done
```

3.配置 apiserver 日志审计

aipserver 涉及的服务器：hdss7-21，hdss7-22

```
[root@hdss7-21 bin]# mkdir /opt/apps/kubernetes/conf
[root@hdss7-21 bin]# vim /opt/apps/kubernetes/conf/audit.yaml # 打开文件后，设置 :set paste，避免自动缩进
apiVersion: audit.k8s.io/v1beta1 # This is required.
kind: Policy
# Don't generate audit events for all requests in RequestReceived stage.
omitStages:
  - "RequestReceived"
rules:
  # Log pod changes at RequestResponse level
  - level: RequestResponse
    resources:
    - group: ""
      # Resource "pods" doesn't match requests to any subresource of pods,
      # which is consistent with the RBAC policy.
      resources: ["pods"]
  # Log "pods/log", "pods/status" at Metadata level
  - level: Metadata
    resources:
    - group: ""
      resources: ["pods/log", "pods/status"]

  # Don't log requests to a configmap called "controller-leader"
  - level: None
    resources:
    - group: ""
      resources: ["configmaps"]
      resourceNames: ["controller-leader"]

  # Don't log watch requests by the "system:kube-proxy" on endpoints or services
  - level: None
    users: ["system:kube-proxy"]
    verbs: ["watch"]
    resources:
    - group: "" # core API group
      resources: ["endpoints", "services"]

  # Don't log authenticated requests to certain non-resource URL paths.
  - level: None
    userGroups: ["system:authenticated"]
    nonResourceURLs:
    - "/api*" # Wildcard matching.
```

```
    - "/version"

  # Log the request body of configmap changes in kube-system.
  - level: Request
    resources:
    - group: "" # core API group
      resources: ["configmaps"]
    # This rule only applies to resources in the "kube-system" namespace.
    # The empty string "" can be used to select non-namespaced resources.
    namespaces: ["kube-system"]

  # Log configmap and secret changes in all other namespaces at the Metadata
level.
  - level: Metadata
    resources:
    - group: "" # core API group
      resources: ["secrets", "configmaps"]

  # Log all other resources in core and extensions at the Request level.
  - level: Request
    resources:
    - group: "" # core API group
    - group: "extensions" # Version of group should NOT be included.

  # A catch-all rule to log all other requests at the Metadata level.
  - level: Metadata
    # Long-running requests like watches that fall under this rule will not
    # generate an audit event in RequestReceived.
    omitStages:
      - "RequestReceived"
```

4.配置启动脚本

aipserver 涉及的服务器：hdss7-21，hdss7-22

* 创建启动脚本

```
[root@hdss7-21                    bin]#                    vim
/opt/apps/kubernetes/server/bin/kube-apiserver-startup.sh
#!/bin/bash

WORK_DIR=$(dirname $(readlink -f $0))
[ $? -eq 0 ] && cd $WORK_DIR || exit

/opt/apps/kubernetes/server/bin/kube-apiserver \
    --apiserver-count 2 \
    --audit-log-path
```

```
/data/logs/kubernetes/kube-apiserver/audit-log \
        --audit-policy-file ../../conf/audit.yaml \
        --authorization-mode RBAC \
        --client-ca-file ./certs/ca.pem \
        --requestheader-client-ca-file ./certs/ca.pem \
        --enable-admission-plugins
NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClas
s,DefaultTolerationSeconds,MutatingAdmissionWebhook,ValidatingAd
missionWebhook,ResourceQuota \
        --etcd-cafile ./certs/ca.pem \
        --etcd-certfile ./certs/client.pem \
        --etcd-keyfile ./certs/client-key.pem \
        --etcd-servers
https://10.4.7.12:2379,https://10.4.7.21:2379,https://10.4.7.22:
2379 \
        --service-account-key-file ./certs/ca-key.pem \
        --service-cluster-ip-range 192.168.0.0/16 \
        --service-node-port-range 3000-29999 \
        --target-ram-mb=1024 \
        --kubelet-client-certificate ./certs/client.pem \
        --kubelet-client-key ./certs/client-key.pem \
        --log-dir  /data/logs/kubernetes/kube-apiserver \
        --tls-cert-file ./certs/apiserver.pem \
        --tls-private-key-file ./certs/apiserver-key.pem \
        --v 2
```

* 配置 supervisor 启动配置

```
[root@hdss7-21 bin]# vim /etc/supervisord.d/kube-apiserver.ini
[program:kube-apiserver-7-21]
command=/opt/apps/kubernetes/server/bin/kube-apiserver-startup.sh
numprocs=1
directory=/opt/apps/kubernetes/server/bin
autostart=true
autorestart=true
startsecs=30
startretries=3
exitcodes=0,2
stopsignal=QUIT
stopwaitsecs=10
user=root
redirect_stderr=true
stdout_logfile=/data/logs/kubernetes/kube-apiserver/apiserver.stdout.log
stdout_logfile_maxbytes=64MB
stdout_logfile_backups=5
stdout_capture_maxbytes=1MB
stdout_events_enabled=false
[root@hdss7-21 bin]# supervisorctl update
[root@hdss7-21 bin]# supervisorctl status
etcd-server-7-21                 RUNNING   pid 23637, uptime 22:26:08
kube-apiserver-7-21              RUNNING   pid 32591, uptime 0:05:37
```

* 启停 apiserver

```
[root@hdss7-12 ~]# supervisorctl start kube-apiserver-7-21
[root@hdss7-12 ~]# supervisorctl stop kube-apiserver-7-21
[root@hdss7-12 ~]# supervisorctl restart kube-apiserver-7-21
[root@hdss7-12 ~]# supervisorctl status kube-apiserver-7-21
```

* 查看进程

1. netstat -lntp|grep api
2. ps uax|grep kube-apiserver|grep -v grep

3.3、配置 apiserver L4 代理

1. nginx 配置

L4 代理涉及的服务器：hdss7-11，hdss7-12

```
[root@hdss7-11 ~]# yum install -y nginx
[root@hdss7-11 ~]# vim /etc/nginx/nginx.conf
# 末尾加上以下内容，stream 只能加在 main 中
# 此处只是简单配置下nginx，实际生产中，建议进行更合理的配置
stream {
    log_format proxy '$time_local|$remote_addr|$upstream_addr|$protocol|$status|'
                     '$session_time|$upstream_connect_time|$bytes_sent|$bytes_received|'
                     '$upstream_bytes_sent|$upstream_bytes_received' ;

    upstream kube-apiserver {
        server 10.4.7.21:6443     max_fails=3 fail_timeout=30s;
        server 10.4.7.22:6443     max_fails=3 fail_timeout=30s;
    }
    server {
        listen 7443;
        proxy_connect_timeout 2s;
        proxy_timeout 900s;
        proxy_pass kube-apiserver;
        access_log /var/log/nginx/proxy.log proxy;
    }
}
[root@hdss7-11 ~]# systemctl start nginx; systemctl enable nginx
[root@hdss7-11 ~]# curl 127.0.0.1:7443  # 测试几次
Client sent an HTTP request to an HTTPS server.
[root@hdss7-11 ~]# cat /var/log/nginx/proxy.log
06/Jan/2020:21:00:27 +0800|127.0.0.1|10.4.7.21:6443|TCP|200|0.001|0.000|76|78|78|76
06/Jan/2020:21:05:03 +0800|127.0.0.1|10.4.7.22:6443|TCP|200|0.020|0.019|76|78|78|76
06/Jan/2020:21:05:04 +0800|127.0.0.1|10.4.7.21:6443|TCP|200|0.001|0.001|76|78|78|76
```

2.keepalived 配置

aipserver L4 代理涉及的服务器：hdss7-11，hdss7-12

* 安装 keepalive

```
[root@hdss7-11 ~]# yum install -y keepalived
[root@hdss7-11 ~]# vim /etc/keepalived/check_port.sh # 配置检查脚本
        #!/bin/bash
        if [ $# -eq 1 ] && [[ $1 =~ ^[0-9]+ ]];then
            [ $(netstat -lntp|grep ":$1 " |wc -l) -eq 0 ] && echo
"[ERROR] nginx may be not running!" && exit 1 || exit 0
        else
            echo "[ERROR] need one port!"
            exit 1
        fi
        [root@hdss7-11 ~]# chmod +x /etc/keepalived/check_port.sh
```

* 配置主节点：/etc/keepalived/keepalived.conf

主节点中，必须加上 nopreempt

因为一旦因为网络抖动导致 VIP 漂移，不能让它自动飘回来，必须要分析原因后手动迁移 VIP 到主节点！如主节点确认正常后，重启备节点的 keepalive，让 VIP 飘到主节点.

keepalived 的日志输出配置此处省略，生产中需要进行处理。

```
! Configuration File for keepalived
global_defs {
   router_id 10.4.7.11
}
vrrp_script chk_nginx {
    script "/etc/keepalived/check_port.sh 7443"
    interval 2
    weight -20
}
vrrp_instance VI_1 {
    state MASTER
    interface ens32
    virtual_router_id 251
    priority 100
    advert_int 1
    mcast_src_ip 10.4.7.11
    nopreempt

    authentication {
        auth_type PASS
        auth_pass 11111111
    }
    track_script {
         chk_nginx
    }
    virtual_ipaddress {
        10.4.7.10
    }
}
```

* 配置备节点：/etc/keepalived/keepalived.conf

```
! Configuration File for keepalived
global_defs {
   router_id 10.4.7.12
}
vrrp_script chk_nginx {
   script "/etc/keepalived/check_port.sh 7443"
   interval 2
   weight -20
}
vrrp_instance VI_1 {
   state BACKUP
   interface ens32
   virtual_router_id 251
   mcast_src_ip 10.4.7.12
   priority 90
   advert_int 1
   authentication {
      auth_type PASS
      auth_pass 11111111
   }
   track_script {
      chk_nginx
   }
   virtual_ipaddress {
      10.4.7.10
   }
}
```

* 启动 keepalived

[root@hdss7-11 ~]# systemctl start keepalived ; systemctl enable keepalived

[root@hdss7-11 ~]# ip addr show ens32

2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000

link/ether 00:0c:29:6d:b8:82 brd ff:ff:ff:ff:ff:ff

inet 10.4.7.11/24 brd 10.4.7.255 scope global noprefixroute ens32

valid_lft forever preferred_lft forever

inet 10.4.7.10/32 scope global ens32

valid_lft forever preferred_lft forever

.....

3.4、controller-manager 安装

controller-manager 涉及的服务器：hdss7-21，hdss7-22

controller-manager 设置为只调用当前机器的 apiserver，走 127.0.0.1 网卡，因此不配制 SSL 证书

```
[root@hdss7-21 ~]# vim /opt/apps/kubernetes/server/bin/kube-controller-manager-startup.sh
#!/bin/sh
WORK_DIR=$(dirname $(readlink -f $0))
[ $? -eq 0 ] && cd $WORK_DIR || exit

/opt/apps/kubernetes/server/bin/kube-controller-manager \
    --cluster-cidr 172.7.0.0/16 \
    --leader-elect true \
    --log-dir /data/logs/kubernetes/kube-controller-manager \
    --master http://127.0.0.1:8080 \
    --service-account-private-key-file ./certs/ca-key.pem \
    --service-cluster-ip-range 192.168.0.0/16 \
    --root-ca-file ./certs/ca.pem \
    --v 2
[root@hdss7-21 ~]# chmod u+x /opt/apps/kubernetes/server/bin/kube-controller-manager-startup.sh
```

```
[root@hdss7-21 ~]# vim /etc/supervisord.d/kube-controller-manager.ini
[program:kube-controller-manager-7-21]
command=/opt/apps/kubernetes/server/bin/kube-controller-manager-startup.sh
numprocs=1                                                                    ; number
directory=/opt/apps/kubernetes/server/bin                                         ; di
autostart=true                                                                ; start a
autorestart=true                                                              ; retstar
startsecs=30                                                                  ; number
startretries=3                                                                ; max # c
exitcodes=0,2                                                                 ; 'expect
stopsignal=QUIT                                                               ; signal
stopwaitsecs=10                                                               ; max num
user=root                                                                     ; setuid
redirect_stderr=true                                                          ; redirec
stdout_logfile=/data/logs/kubernetes/kube-controller-manager/controller.stdout.log  ; stder
stdout_logfile_maxbytes=64MB                                                  ; max # l
stdout_logfile_backups=4                                                      ; # of st
stdout_capture_maxbytes=1MB                                                   ; number
stdout_events_enabled=false                                                   ; emit ev
```

```
[root@hdss7-21 ~]# supervisorctl update
kube-controller-manager-7-21: stopped
kube-controller-manager-7-21: updated process group
[root@hdss7-21 ~]# supervisorctl status
etcd-server-7-21                 RUNNING   pid 23637, uptime 1 day, 0:16:54
kube-apiserver-7-21              RUNNING   pid 32591, uptime 1:56:23
kube-controller-manager-7-21     RUNNING   pid 33357, uptime 0:00:38
```

3.5、kube-scheduler 安装

kube-scheduler 涉及的服务器：hdss7-21，hdss7-22

kube-scheduler 设置为只调用当前机器的 apiserver，走 127.0.0.1 网卡，因此不配制 SSL 证书

```
[root@hdss7-21 ~]# vim /opt/apps/kubernetes/server/bin/kube-scheduler-startup.sh
#!/bin/sh
WORK_DIR=$(dirname $(readlink -f $0))
[ $? -eq 0 ] && cd $WORK_DIR || exit

/opt/apps/kubernetes/server/bin/kube-scheduler \
    --leader-elect  \
    --log-dir /data/logs/kubernetes/kube-scheduler \
    --master http://127.0.0.1:8080 \
    --v 2
[root@hdss7-21 ~]# chmod u+x /opt/apps/kubernetes/server/bin/kube-scheduler-startup.sh
[root@hdss7-21 ~]# mkdir -p /data/logs/kubernetes/kube-scheduler
```

```
[root@hdss7-21 ~]# vim /etc/supervisord.d/kube-scheduler.ini
[program:kube-scheduler-7-21]
command=/opt/apps/kubernetes/server/bin/kube-scheduler-startup.sh
numprocs=1
directory=/opt/apps/kubernetes/server/bin
autostart=true
autorestart=true
startsecs=30
startretries=3
exitcodes=0,2
stopsignal=QUIT
stopwaitsecs=10
user=root
redirect_stderr=true
stdout_logfile=/data/logs/kubernetes/kube-scheduler/scheduler.stdout.log
stdout_logfile_maxbytes=64MB
stdout_logfile_backups=4
stdout_capture_maxbytes=1MB
stdout_events_enabled=false
```

```
[root@hdss7-21 ~]# supervisorctl update
kube-scheduler-7-21: stopped
kube-scheduler-7-21: updated process group
[root@hdss7-21 ~]# supervisorctl status
etcd-server-7-21                 RUNNING   pid 23637, uptime 1 day, 0:26:53
kube-apiserver-7-21              RUNNING   pid 32591, uptime 2:06:22
kube-controller-manager-7-21     RUNNING   pid 33357, uptime 0:10:37
kube-scheduler-7-21              RUNNING   pid 33450, uptime 0:01:18
```

3.6、检查主控节点状态

```
[root@hdss7-21 ~]# ln -s /opt/apps/kubernetes/server/bin/kubectl /usr/local/bin/
[root@hdss7-21 ~]# kubectl get cs
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-1               Healthy   {"health": "true"}
etcd-0               Healthy   {"health": "true"}
etcd-2               Healthy   {"health": "true"}
```

```
[root@hdss7-22 ~]# ln -s /opt/apps/kubernetes/server/bin/kubectl /usr/local/bin/
[root@hdss7-22 ~]# kubectl get cs
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-2               Healthy   {"health": "true"}
etcd-1               Healthy   {"health": "true"}
etcd-0               Healthy   {"health": "true"}
```

4. 运算节点部署

4.1、kubelet 部署

1. 签发证书

证书签发在 hdss7-200 操作

[root@hdss7-200 ~]# cd /opt/certs/

[root@hdss7-200 certs]# vim kubelet-csr.json # 将所有可能的 kubelet 机器 IP 添加到 hosts 中

```
{
    "CN": "k8s-kubelet",
    "hosts": [
    "127.0.0.1",
    "10.4.7.10",
    "10.4.7.21",
    "10.4.7.22",
    "10.4.7.23",
    "10.4.7.24",
    "10.4.7.25",
    "10.4.7.26",
    "10.4.7.27",
    "10.4.7.28"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "beijing",
            "L": "beijing",
            "O": "od",
```

```
            "OU": "ops"
        }
    ]
}
```

[root@hdss7-200 certs]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=server kubelet-csr.json | cfssl-json -bare kubelet

2020/01/06 23:10:56 [INFO] generate received request

2020/01/06 23:10:56 [INFO] received CSR

2020/01/06 23:10:56 [INFO] generating key: rsa-2048

2020/01/06 23:10:56 [INFO] encoded CSR

2020/01/06 23:10:56 [INFO] signed certificate with serial number 612219427848569697387713705315595555767101820379

2020/01/06 23:10:56 [WARNING] This certificate lacks a "hosts" field. This makes it unsuitable for

websites. For more information see the Baseline Requirements for the Issuance and Management

of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabforum.org);

specifically, section 10.2.3 ("Information Requirements").

```
[root@hdss7-200 certs]# ls kubelet* -l
-rw-r--r-- 1 root root 1115 Jan  6 23:10 kubelet.csr
-rw-r--r-- 1 root root  452 Jan  6 23:10 kubelet-csr.json
-rw------- 1 root root 1675 Jan  6 23:10 kubelet-key.pem
-rw-r--r-- 1 root root 1468 Jan  6 23:10 kubelet.pem
```

[root@hdss7-200 certs]# scp kubelet.pem kubelet-key.pem hdss7-21:/opt/apps/kubernetes/server/bin/certs/

[root@hdss7-200 certs]# scp kubelet.pem kubelet-key.pem hdss7-22:/opt/apps/kubernetes/server/bin/certs/

2.创建 kubelet 配置

kubelet 配置在 hdss7-21 hdss7-22 操作\

* set-cluster # 创建需要连接的集群信息，可以创建多个 k8s 集群信息

```
[root@hdss7-21 ~]# kubectl config set-cluster myk8s \
--certificate-authority=/opt/apps/kubernetes/server/bin/certs/ca.pem \
--embed-certs=true \
--server=https://10.4.7.10:7443 \
--kubeconfig=/opt/apps/kubernetes/conf/kubelet.kubeconfig
```

* set-credentials # 创建用户账号，即用户登陆使用的客户端私有和证书，可以创建多个证书

```
[root@hdss7-21 ~]# kubectl config set-credentials k8s-node \
--client-certificate=/opt/apps/kubernetes/server/bin/certs/client.pem \
--client-key=/opt/apps/kubernetes/server/bin/certs/client-key.pem \
--embed-certs=true \
--kubeconfig=/opt/apps/kubernetes/conf/kubelet.kubeconfig
```

* set-context # 设置 context，即确定账号和集群对应关系

```
[root@hdss7-21 ~]# kubectl config set-context myk8s-context \
--cluster=myk8s \
--user=k8s-node \
--kubeconfig=/opt/apps/kubernetes/conf/kubelet.kubeconfig
```

* use-context # 设置当前使用哪个 context

[root@hdss7-21 ~]# kubectl config use-context myk8s-context --kubeconfig=/opt/apps/kubernetes/conf/kubelet.kubeconfig

3. 创建 kubelet 配置

此步骤只需要在一台 master 节点执行

授权 k8s-node 用户绑定集群角色 system:node ，让 k8s-node 成为具备运算节点的权限。

```
[root@hdss7-21 ~]# vim k8s-node.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: k8s-node
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:node
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: k8s-node
[root@hdss7-21 ~]# kubectl create -f k8s-node.yaml
clusterrolebinding.rbac.authorization.k8s.io/k8s-node created
[root@hdss7-21 ~]# kubectl get clusterrolebinding k8s-node
NAME       AGE
k8s-node   36s
```

4. 装备 pause 镜像

将 pause 镜像放入到 harbor 私有仓库中，仅在 hdss7-200 操作：

```
[root@hdss7-200 ~]# docker image pull kubernetes/pause
[root@hdss7-200 ~]# docker image tag kubernetes/pause:latest harbor.od.com/public/pause:latest
[root@hdss7-200 ~]# docker login -u admin harbor.od.com
[root@hdss7-200 ~]# docker image push harbor.od.com/public/pause:latest
```

5.创建启动脚本

在 node 节点创建脚本并启动 kubelet，涉及服务器： hdss7-21 hdss7-22

```
[root@hdss7-21 ~]# vim /opt/apps/kubernetes/server/bin/kubelet-startup.sh
#!/bin/sh

WORK_DIR=$(dirname $(readlink -f $0))
[ $? -eq 0 ] && cd $WORK_DIR || exit

/opt/apps/kubernetes/server/bin/kubelet \
    --anonymous-auth=false \
    --cgroup-driver systemd \
    --cluster-dns 192.168.0.2 \
    --cluster-domain cluster.local \
    --runtime-cgroups=/systemd/system.slice \
    --kubelet-cgroups=/systemd/system.slice \
    --fail-swap-on="false" \
    --client-ca-file ./certs/ca.pem \
    --tls-cert-file ./certs/kubelet.pem \
    --tls-private-key-file ./certs/kubelet-key.pem \
    --hostname-override hdss7-21.host.com \
    --image-gc-high-threshold 20 \
    --image-gc-low-threshold 10 \
    --kubeconfig ../../conf/kubelet.kubeconfig \
    --log-dir /data/logs/kubernetes/kube-kubelet \
    --pod-infra-container-image harbor.od.com/public/pause:latest \
    --root-dir /data/kubelet
[root@hdss7-21 ~]# chmod u+x /opt/apps/kubernetes/server/bin/kubelet-startup.sh
[root@hdss7-21 ~]# mkdir -p /data/logs/kubernetes/kube-kubelet /data/kubelet

[root@hdss7-21 ~]# vim /etc/supervisord.d/kube-kubelet.ini
[program:kube-kubelet-7-21]
command=/opt/apps/kubernetes/server/bin/kubelet-startup.sh
numprocs=1
directory=/opt/apps/kubernetes/server/bin
autostart=true
autorestart=true
startsecs=30
startretries=3
exitcodes=0,2
stopsignal=QUIT
stopwaitsecs=10
user=root
redirect_stderr=true
stdout_logfile=/data/logs/kubernetes/kube-kubelet/kubelet.stdout.log
stdout_logfile_maxbytes=64MB
stdout_logfile_backups=5
stdout_capture_maxbytes=1MB
stdout_events_enabled=false
```

```
[root@hdss7-21 ~]# supervisorctl update
[root@hdss7-21 ~]# supervisorctl status
etcd-server-7-21                 RUNNING   pid 23637, uptime 1 day, 14:56:25
kube-apiserver-7-21              RUNNING   pid 32591, uptime 16:35:54
kube-controller-manager-7-21     RUNNING   pid 33357, uptime 14:40:09
kube-kubelet-7-21                RUNNING   pid 37232, uptime 0:01:08
kube-scheduler-7-21              RUNNING   pid 33450, uptime 14:30:50
[root@hdss7-21 ~]# kubectl get node
NAME                STATUS   ROLES    AGE     VERSION
hdss7-21.host.com   Ready    <none>   3m13s   v1.15.2
hdss7-22.host.com   Ready    <none>   3m13s   v1.15.2
```

6. 修改节点角色

使用 kubectl get nodes 获取的 Node 节点角色为空，可以按照以下方式修改

```
[root@hdss7-21 ~]# kubectl get node
NAME                STATUS   ROLES    AGE     VERSION
hdss7-21.host.com   Ready    <none>   3m13s   v1.15.2
hdss7-22.host.com   Ready    <none>   3m13s   v1.15.2
[root@hdss7-21 ~]# kubectl label node hdss7-21.host.com node-role.kubernetes.io/node=
node/hdss7-21.host.com labeled
[root@hdss7-21 ~]# kubectl label node hdss7-21.host.com node-role.kubernetes.io/master=
node/hdss7-21.host.com labeled
[root@hdss7-21 ~]# kubectl label node hdss7-22.host.com node-role.kubernetes.io/master=
node/hdss7-22.host.com labeled
[root@hdss7-21 ~]# kubectl label node hdss7-22.host.com node-role.kubernetes.io/node=
node/hdss7-22.host.com labeled
[root@hdss7-21 ~]# kubectl get node
NAME                STATUS   ROLES         AGE     VERSION
hdss7-21.host.com   Ready    master,node   7m44s   v1.15.2
hdss7-22.host.com   Ready    master,node   7m44s   v1.15.2
```

4.2. kube-proxy 部署

1. 签发证书

证书签发在 hdss7-200 操作

[root@hdss7-200 ~]# cd /opt/certs/
[root@hdss7-200 certs]# vim kube-proxy-csr.json # CN 其实是 k8s 中的角色

```
{
    "CN": "system:kube-proxy",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
```

```
            "ST": "beijing",
            "L": "beijing",
            "O": "od",
            "OU": "ops"
        }
    ]
}
```

[root@hdss7-200 certs]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=client kube-proxy-csr.json |cfssl-json -bare kube-proxy-client

2020/01/07 21:45:53 [INFO] generate received request

2020/01/07 21:45:53 [INFO] received CSR

2020/01/07 21:45:53 [INFO] generating key: rsa-2048

2020/01/07 21:45:53 [INFO] encoded CSR

2020/01/07 21:45:53 [INFO] signed certificate with serial number 620191685968917036075463174423999296907693104226

2020/01/07 21:45:53 [WARNING] This certificate lacks a "hosts" field. This makes it unsuitable for

websites. For more information see the Baseline Requirements for the Issuance and Management

of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabforum.org);

[root@hdss7-200 certs]# ls kube-proxy-c* -l # 因为 kube-proxy 使用的用户是 kube-proxy，不能使用 client 证书，必须要重新签发自己的证书

-rw-r--r-- 1 root root 1005 Jan 7 21:45 kube-proxy-client.csr

-rw------- 1 root root 1675 Jan 7 21:45 kube-proxy-client-key.pem

-rw-r--r-- 1 root root 1375 Jan 7 21:45 kube-proxy-client.pem

-rw-r--r-- 1 root root 267 Jan 7 21:45 kube-proxy-csr.json

[root@hdss7-200 certs]# scp kube-proxy-client-key.pem kube-proxy-client.pem hdss7-21:/opt/apps/kubernetes/server/bin/certs/

100% 1375 870.6KB/s 00:00

[root@hdss7-200 certs]# scp kube-proxy-client-key.pem kube-proxy-client.pem hdss7-22:/opt/apps/kubernetes/server/bin/certs/

2.创建 kube-proxy 配置

在所有 node 节点创建，涉及服务器：hdss7-21 ，hdss7-22

[root@hdss7-21 ~]# kubectl config set-cluster myk8s \

--certificate-authority=/opt/apps/kubernetes/server/bin/certs/ca.pem \

--embed-certs=true \

--server=https://10.4.7.10:7443 \

--kubeconfig=/opt/apps/kubernetes/conf/kube-proxy.kubeconfig

```
[root@hdss7-21 ~]# kubectl config set-credentials kube-proxy \
--client-certificate=/opt/apps/kubernetes/server/bin/certs/kube-proxy-client.pem \
--client-key=/opt/apps/kubernetes/server/bin/certs/kube-proxy-client-key.pem \
--embed-certs=true \
--kubeconfig=/opt/apps/kubernetes/conf/kube-proxy.kubeconfig

[root@hdss7-21 ~]# kubectl config set-context myk8s-context \
--cluster=myk8s \
--user=kube-proxy \
--kubeconfig=/opt/apps/kubernetes/conf/kube-proxy.kubeconfig

[root@hdss7-21 ~]# kubectl config use-context myk8s-context --kubeconfig=/opt/apps/kubernetes/conf/kube-proxy.kubeconfig
```

3.加载 ipvs 模块

kube-proxy 共有 3 种流量调度模式，分别是 namespace，iptables，ipvs，其中 ipvs 性能最好。

```
[root@hdss7-21 ~]# for i in $(ls /usr/lib/modules/$(uname -r)/kernel/net/netfilter/ipvs|grep -o "^[^.]*");do echo $i; /sbin/modinfo -F filename $i >/dev/null 2>&1 && /sbin/modprobe $i;done
[root@hdss7-21 ~]# lsmod | grep ip_vs  # 查看 ipvs 模块
```

4.创建启动脚本

```
[root@hdss7-21 ~]# vim /opt/apps/kubernetes/server/bin/kube-proxy-startup.sh
#!/bin/sh

WORK_DIR=$(dirname $(readlink -f $0))
[ $? -eq 0 ] && cd $WORK_DIR || exit

/opt/apps/kubernetes/server/bin/kube-proxy \
  --cluster-cidr 172.7.0.0/16 \
  --hostname-override hdss7-21.host.com \
  --proxy-mode=ipvs \
  --ipvs-scheduler=nq \
  --kubeconfig ../../conf/kube-proxy.kubeconfig
[root@hdss7-21 ~]# chmod u+x /opt/apps/kubernetes/server/bin/kube-proxy-startup.sh
[root@hdss7-21 ~]# mkdir -p /data/logs/kubernetes/kube-proxy
[root@hdss7-21 ~]# vim /etc/supervisord.d/kube-proxy.ini
[program:kube-proxy-7-21]
command=/opt/apps/kubernetes/server/bin/kube-proxy-startup.sh
numprocs=1
directory=/opt/apps/kubernetes/server/bin
autostart=true
autorestart=true
startsecs=30
startretries=3
exitcodes=0,2
stopsignal=QUIT
stopwaitsecs=10
user=root
redirect_stderr=true
stdout_logfile=/data/logs/kubernetes/kube-proxy/proxy.stdout.log
stdout_logfile_maxbytes=64MB
stdout_logfile_backups=5
stdout_capture_maxbytes=1MB
stdout_events_enabled=false

[root@hdss7-21 ~]# supervisorctl update
```

5. 验证集群

```
[root@hdss7-21 ~]# supervisorctl status
etcd-server-7-21                 RUNNING   pid 23637, uptime 2 days, 0:27:18
kube-apiserver-7-21              RUNNING   pid 32591, uptime 1 day, 2:06:47
kube-controller-manager-7-21     RUNNING   pid 33357, uptime 1 day, 0:11:02
kube-kubelet-7-21                RUNNING   pid 37232, uptime 9:32:01
kube-proxy-7-21                  RUNNING   pid 47088, uptime 0:06:19
kube-scheduler-7-21              RUNNING   pid 33450, uptime 1 day, 0:01:43

[root@hdss7-21 ~]# yum install -y ipvsadm
[root@hdss7-21 ~]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.0.1:443 nq
  -> 10.4.7.21:6443               Masq    1      0          0
  -> 10.4.7.22:6443               Masq    1      0          0
```

```
[root@hdss7-21 ~]# curl -I 172.7.21.2
HTTP/1.1 200 OK
Server: nginx/1.17.6
Date: Tue, 07 Jan 2020 14:28:46 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 19 Nov 2019 12:50:08 GMT
Connection: keep-alive
ETag: "5dd3e500-264"
Accept-Ranges: bytes

[root@hdss7-21 ~]# curl -I 172.7.22.2  # 缺少网络插件，无法跨节点通信
```

5. 核心插件部署

5.1 CNI 网络插件

kubernetes 设计了网络模型，但是 pod 之间通信的具体实现交给了 CNI 往插件。常用的 CNI 网络插件有：Flannel 、Calico、Canal、Contiv 等，其中 Flannel 和 Calico 占比接近 80%，Flannel 占比略多于 Calico。本次部署使用 Flannel 作为网络插件。涉及的机器 hdss7-21,hdss7-22

1. 安装 Flannel

github 地址：https://github.com/coreos/flannel/releases

涉及的机器 hdss7-21,hdss7-22

[root@hdss7-21 ~]# cd /opt/src/

[root@hdss7-21 src]# wget https://github.com/coreos/flannel/releases/download/v0.11.0/flannel-v0.11.0-linux-amd64.tar.gz

[root@hdss7-21 src]# mkdir /opt/release/flannel-v0.11.0 # 因为 flannel 压缩包内部没有套目录

[root@hdss7-21 src]# tar -xf flannel-v0.11.0-linux-amd64.tar.gz -C /opt/release/flannel-v0.11.0

[root@hdss7-21 src]# ln -s /opt/release/flannel-v0.11.0 /opt/apps/flannel

[root@hdss7-21 src]# ll /opt/apps/flannel

lrwxrwxrwx 1 root root 28 Jan 9 22:33 /opt/apps/flannel -> /opt/release/flannel-v0.11.0

2. 拷贝证书

flannel 需要以客户端的身份访问 etcd，需要相关证书

[root@hdss7-21 src]# mkdir /opt/apps/flannel/certs

[root@hdss7-200 ~]# cd /opt/certs/

[root@hdss7-200 certs]# scp ca.pem client-key.pem client.pem

hdss7-21:/opt/apps/flannel/certs/

3.创建启动脚本

涉及的机器 hdss7-21,hdss7-22

[root@hdss7-21 src]# vim /opt/apps/flannel/subnet.env # 创建子网信息，7-22 的 subnet 需要修改

```
FLANNEL_NETWORK=172.7.0.0/16
FLANNEL_SUBNET=172.7.21.1/24
FLANNEL_MTU=1500
FLANNEL_IPMASQ=false
```

[root@hdss7-21 src]# /opt/apps/etcd/etcdctl set /coreos.com/network/config '{"Network": "172.7.0.0/16", "Backend": {"Type": "host-gw"}}'

[root@hdss7-21 src]# /opt/apps/etcd/etcdctl get /coreos.com/network/config # 只需要在一台 etcd 机器上设置就可以了

```
{"Network": "172.7.0.0/16", "Backend": {"Type": "host-gw"}}
```

```
# public-ip 为本机IP，iface 为当前宿主机对外网卡
[root@hdss7-21 src]# vim /opt/apps/flannel/flannel-startup.sh
#!/bin/sh

WORK_DIR=$(dirname $(readlink -f $0))
[ $? -eq 0 ] && cd $WORK_DIR || exit

/opt/apps/flannel/flanneld \
    --public-ip=10.4.7.21 \
    --etcd-endpoints=https://10.4.7.12:2379,https://10.4.7.21:2379,https://10.4.7.22:2379
    --etcd-keyfile=./certs/client-key.pem \
    --etcd-certfile=./certs/client.pem \
    --etcd-cafile=./certs/ca.pem \
    --iface=ens32 \
    --subnet-file=./subnet.env \
    --healthz-port=2401
[root@hdss7-21 src]# chmod u+x /opt/apps/flannel/flannel-startup.sh
```

```
[root@hdss7-21 src]# vim /etc/supervisord.d/flannel.ini
[program:flanneld-7-21]
command=/opt/apps/flannel/flannel-startup.sh                    ; the program (relative use
numprocs=1                                                      ; number of processes copie
directory=/opt/apps/flannel                                     ; directory to cwd to befor
autostart=true                                                  ; start at supervisord star
autorestart=true                                                ; retstart at unexpected qu
startsecs=30                                                    ; number of secs prog must
startretries=3                                                  ; max # of serial start fai
exitcodes=0,2                                                   ; 'expected' exit codes for
stopsignal=QUIT                                                 ; signal used to kill proce
stopwaitsecs=10                                                 ; max num secs to wait b4 S
user=root                                                       ; setuid to this UNIX accou
redirect_stderr=true                                            ; redirect proc stderr to s
stdout_logfile=/data/logs/flanneld/flanneld.stdout.log          ; stderr log path, NONE for
stdout_logfile_maxbytes=64MB                                    ; max # logfile bytes b4 ro
stdout_logfile_backups=5                                        ; # of stdout logfile backu
stdout_capture_maxbytes=1MB                                     ; number of bytes in 'captu
stdout_events_enabled=false                                     ; emit events on stdout wri
[root@hdss7-21 src]# mkdir -p /data/logs/flanneld/
[root@hdss7-21 src]# supervisorctl update
flanneld-7-21: added process group
[root@hdss7-21 src]# supervisorctl status
etcd-server-7-21                 RUNNING   pid 1058, uptime -1 day, 16:33:25
flanneld-7-21                    RUNNING   pid 13154, uptime 0:00:30
kube-apiserver-7-21              RUNNING   pid 1061, uptime -1 day, 16:33:25
kube-controller-manager-7-21     RUNNING   pid 1068, uptime -1 day, 16:33:25
kube-kubelet-7-21                RUNNING   pid 1052, uptime -1 day, 16:33:25
kube-proxy-7-21                  RUNNING   pid 1082, uptime -1 day, 16:33:25
kube-scheduler-7-21              RUNNING   pid 1089, uptime -1 day, 16:33:25
```

4. 验证跨网络访问

```
[root@hdss7-21 src]# kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP           NODE                NOM
nginx-ds-7db29   1/1     Running   1          2d    172.7.22.2   hdss7-22.host.com   <no
nginx-ds-vvsz7   1/1     Running   1          2d    172.7.21.2   hdss7-21.host.com   <no
[root@hdss7-21 src]# curl -I 172.7.22.2
HTTP/1.1 200 OK
Server: nginx/1.17.6
Date: Thu, 09 Jan 2020 14:55:21 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 19 Nov 2019 12:50:08 GMT
Connection: keep-alive
ETag: "5dd3e500-264"
Accept-Ranges: bytes
```

5. 解决 pod 间 IP 透传问题

所有 Node 上操作，即优化 NAT 网络

```
# 从pod a跨宿主机访问pod b时，在pod b中能看到的地址为 pod a 宿主机地址
[root@nginx-ds-jdp7q /]# tail -f /usr/local/nginx/logs/access.log
10.4.7.22 - - [13/Jan/2020:13:13:39 +0000] "GET / HTTP/1.1" 200 12 "-" "curl/7.29.0"
10.4.7.22 - - [13/Jan/2020:13:14:27 +0000] "GET / HTTP/1.1" 200 12 "-" "curl/7.29.0"
10.4.7.22 - - [13/Jan/2020:13:54:20 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/7.29.0"
10.4.7.22 - - [13/Jan/2020:13:54:25 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/7.29.0"
[root@hdss7-21 ~]# iptables-save |grep POSTROUTING|grep docker # 引发问题的规则
-A POSTROUTING -s 172.7.21.0/24 ! -o docker0 -j MASQUERADE
```

[root@hdss7-21 ~]# yum install -y iptables-services

[root@hdss7-21 ~]# systemctl start iptables.service ; systemctl enable iptables.service

需要处理的规则:

[root@hdss7-21 ~]# iptables-save |grep POSTROUTING|grep docker

-A POSTROUTING -s 172.7.21.0/24 ! -o docker0 -j MASQUERADE

[root@hdss7-21 ~]# iptables-save | grep -i reject

-A INPUT -j REJECT --reject-with icmp-host-prohibited

-A FORWARD -j REJECT --reject-with icmp-host-prohibited

处理方式:

[root@hdss7-21 ~]# iptables -t nat -D POSTROUTING -s 172.7.21.0/24 ! -o docker0 -j MASQUERADE

[root@hdss7-21 ~]# iptables -t nat -I POSTROUTING -s 172.7.21.0/24 ! -d 172.7.0.0/16 ! -o docker0 -j MASQUERADE

[root@hdss7-21 ~]# iptables -t filter -D INPUT -j REJECT --reject-with icmp-host-prohibited

[root@hdss7-21 ~]# iptables -t filter -D FORWARD -j REJECT --reject-with icmp-host-prohibited

[root@hdss7-21 ~]# iptables-save > /etc/sysconfig/iptables

```
# 此时跨宿主机访问pod时，显示pod的IP
[root@nginx-ds-jdp7q /]# tail -f /usr/local/nginx/logs/access.log
172.7.22.2 - - [13/Jan/2020:14:15:39 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/7.29.0"
172.7.22.2 - - [13/Jan/2020:14:15:47 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/7.29.0"
172.7.22.2 - - [13/Jan/2020:14:15:48 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/7.29.0"
172.7.22.2 - - [13/Jan/2020:14:15:48 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/7.29.0"
```

5.2 CoreDNS

CoreDNS 用于实现 service --> cluster IP 的 DNS 解析。以容器的方式交付到 k8s 集群，由 k8s 自行管理，降低人为操作的复杂度。

1. 配置 yaml 文件库

在 hdss7-200 中配置 yaml 文件库，后期通过 Http 方式去使用 yaml 清单文件。

* 配置 nginx 虚拟主机(hdss7-200)

```
[root@hdss7-200 ~]# vim /etc/nginx/conf.d/k8s-yaml.od.com.conf
server {
    listen       80;
    server_name  k8s-yaml.od.com;

    location / {
        autoindex on;
        default_type text/plain;
        root /data/k8s-yaml;
    }
}
[root@hdss7-200 ~]# mkdir /data/k8s-yaml;
[root@hdss7-200 ~]# nginx -qt && nginx -s reload
```

* 配置 dns 解析(hdss7-11)

```
[root@hdss7-11 ~]# vim /var/named/od.com.zone
[root@hdss7-11 ~]# cat /var/named/od.com.zone
$ORIGIN od.com.
$TTL 600  ; 10 minutes
@       IN SOA  dns.od.com. dnsadmin.od.com. (
        2020011301 ; serial
        10800      ; refresh (3 hours)
        900        ; retry (15 minutes)
        604800     ; expire (1 week)
        86400      ; minimum (1 day)
        )
        NS   dns.od.com.
$TTL 60 ; 1 minute
dns                A    10.4.7.11
harbor             A    10.4.7.200
k8s-yaml           A    10.4.7.200
[root@hdss7-11 ~]# systemctl restart named
```

2.coredns 的资源清单文件

清单文件存放到 hdss7-200:/data/k8s-yaml/coredns/coredns_1.6.1/

* rabc.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: coredns
  namespace: kube-system
  labels:
      kubernetes.io/cluster-service: "true"
      addonmanager.kubernetes.io/mode: Reconcile
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
    addonmanager.kubernetes.io/mode: Reconcile
  name: system:coredns
rules:
- apiGroups:
  - ""
  resources:
  - endpoints
  - services
  - pods
  - namespaces
  verbs:
  - list
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
    addonmanager.kubernetes.io/mode: EnsureExists
  name: system:coredns
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:coredns
subjects:
- kind: ServiceAccount
  name: coredns
  namespace: kube-system
```

* configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
        errors
        log
        health
        ready
        kubernetes cluster.local 192.168.0.0/16
        forward . 10.4.7.11
        cache 30
        loop
        reload
        loadbalance
    }
```

* deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: coredns
  namespace: kube-system
  labels:
    k8s-app: coredns
    kubernetes.io/name: "CoreDNS"
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: coredns
  template:
    metadata:
      labels:
        k8s-app: coredns
    spec:
      priorityClassName: system-cluster-critical
      serviceAccountName: coredns
      containers:
      - name: coredns
        image: harbor.od.com/public/coredns:v1.6.1
        args:
        - -conf
        - /etc/coredns/Corefile
        volumeMounts:
        - name: config-volume
          mountPath: /etc/coredns
        ports:
        - containerPort: 53
          name: dns
          protocol: UDP
        - containerPort: 53
          name: dns-tcp
          protocol: TCP
        - containerPort: 9153
          name: metrics
          protocol: TCP
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
      dnsPolicy: Default
      volumes:
        - name: config-volume
          configMap:
            name: coredns
            items:
            - key: Corefile
              path: Corefile
```

* service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: coredns
  namespace: kube-system
  labels:
    k8s-app: coredns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: "CoreDNS"
spec:
  selector:
    k8s-app: coredns
  clusterIP: 192.168.0.2
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
  - name: metrics
    port: 9153
    protocol: TCP
```

3. 交付 coredns 到 K8s

准备镜像

[root@hdss7-200 ~]# docker pull coredns/coredns:1.6.1 [root@hdss7-200 ~]# docker image tag coredns/coredns:1.6.1 harbor.od.com/public/coredns:v1.6.1

[root@hdss7-200 ~]# docker image push harbor.od.com/public/coredns:v1.6.1

交付 coredns

[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/coredns/coredns_1.6.1/rbac.yaml

[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/coredns/coredns_1.6.1/configmap.yaml

[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/coredns/coredns_1.6.1/deployment.yaml

[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/coredns/coredns_1.6.1/service.yaml

[root@hdss7-21 ~]# kubectl get all -n kube-system -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

pod/coredns-6b6c4f9648-4vtcl 1/1 Running 0 38s 172.7.21.3 hdss7-21.host.com <none> <none>

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR

service/coredns ClusterIP 192.168.0.2 <none> 53/UDP,53/TCP,9153/TCP 29s k8s-app=coredns

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
CONTAINERS   IMAGES                                  SELECTOR
deployment.apps/coredns   1/1     1            1           39s   coredns
harbor.od.com/public/coredns:v1.6.1   k8s-app=coredns

NAME                                   DESIRED   CURRENT   READY   AGE
CONTAINERS   IMAGES                                  SELECTOR
replicaset.apps/coredns-6b6c4f9648   1         1         1       39s   coredns
harbor.od.com/public/coredns:v1.6.1
k8s-app=coredns,pod-template-hash=6b6c4f9648
```

4. 测试 dns

```
# 创建 service
[root@hdss7-21 ~]# kubectl create deployment nginx-web
--image=harbor.od.com/public/nginx:src_1.14.2
[root@hdss7-21 ~]# kubectl expose deployment nginx-web --port=80 --target-port=80
[root@hdss7-21 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   192.168.0.1      <none>        443/TCP   8d
nginx-web    ClusterIP   192.168.164.230  <none>        80/TCP    8s
# 测试 DNS，集群外必须使用 FQDN(Fully Qualified Domain Name)，全域名
[root@hdss7-21 ~]# dig -t A nginx-web.default.svc.cluster.local @192.168.0.2 +short # 内网解析 OK
192.168.164.230
[root@hdss7-21 ~]# dig -t A www.baidu.com @192.168.0.2 +short # 外网解析 OK
www.a.shifen.com.
180.101.49.11
180.101.49.12
```

5.3 Ingress-Controller

service 是将一组 pod 管理起来，提供了一个 cluster ip 和 service name 的统一访问入口，屏蔽了 pod 的 ip 变化。 ingress 是一种基于七层的流量转发策略，即将符合条件的域名或者 location 流量转发到特定的 service 上，而 ingress 仅仅是一种规则，k8s 内部并没有自带代理程序完成这种规则转发。

ingress-controller 是一个代理服务器，将 ingress 的规则能真正实现的方式，常用的有 nginx,traefik,haproxy。但是在 k8s 集群中，建议使用 traefik，性能比 haroxy 强大，更新配置不需要重载服务，是首选的 ingress-controller。

github 地址：https://github.com/containous/traefik

1. 配置 traefik 资源清单

清单文件存放到 hdss7-200:/data/k8s-yaml/traefik/traefik_1.7.2

* rbac.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: traefik-ingress-controller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: traefik-ingress-controller
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
      - secrets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-ingress-controller
subjects:
- kind: ServiceAccount
  name: traefik-ingress-controller
  namespace: kube-system
```

* daemonset.yaml

```yaml
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: traefik-ingress
  namespace: kube-system
  labels:
    k8s-app: traefik-ingress
spec:
  template:
    metadata:
      labels:
        k8s-app: traefik-ingress
        name: traefik-ingress
    spec:
      serviceAccountName: traefik-ingress-controller
      terminationGracePeriodSeconds: 60
      containers:
      - image: harbor.od.com/public/traefik:v1.7.2
        name: traefik-ingress
        ports:
        - name: controller
          containerPort: 80
          hostPort: 81
        - name: admin-web
          containerPort: 8080
        securityContext:
          capabilities:
            drop:
            - ALL
            add:
            - NET_BIND_SERVICE
        args:
        - --api
        - --kubernetes
        - --logLevel=INFO
        - --insecureskipverify=true
        - --kubernetes.endpoint=https://10.4.7.10:7443
        - --accesslog
        - --accesslog.filepath=/var/log/traefik_access.log
        - --traefiklog
        - --traefiklog.filepath=/var/log/traefik.log
        - --metrics.prometheus
```

* service.yaml

```yaml
kind: Service
apiVersion: v1
metadata:
  name: traefik-ingress-service
  namespace: kube-system
spec:
  selector:
    k8s-app: traefik-ingress
  ports:
    - protocol: TCP
      port: 80
      name: controller
    - protocol: TCP
      port: 8080
      name: admin-web
```

* ingress.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: traefik-web-ui
  namespace: kube-system
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: traefik.od.com
    http:
      paths:
      - path: /
        backend:
          serviceName: traefik-ingress-service
          servicePort: 8080
```

* 准备镜像

[root@hdss7-200 traefik_1.7.2]# docker pull traefik:v1.7.2-alpine

[root@hdss7-200 traefik_1.7.2]# docker image tag traefik:v1.7.2-alpine harbor.od.com/public/traefik:v1.7.2

[root@hdss7-200 traefik_1.7.2]# docker push harbor.od.com/public/traefik:v1.7.2

2. 交付 traefik 到 k8s

```
[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/traefik/traefik_1.7.2/rbac.yaml
[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/traefik/traefik_1.7.2/daemonset.yaml
[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/traefik/traefik_1.7.2/service.yaml
[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/traefik/traefik_1.7.2/ingress.yaml
```

```
[root@hdss7-21 ~]# kubectl get pods -n kube-system -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP           NODE
coredns-6b6c4f9648-4vtcl   1/1     Running   1          24h   172.7.21.3   hdss7-21.host.com
traefik-ingress-4gm4w      1/1     Running   0          77s   172.7.21.5   hdss7-21.host.com
traefik-ingress-hwr2j      1/1     Running   0          77s   172.7.22.3   hdss7-22.host.com
[root@hdss7-21 ~]# kubectl get ds -n kube-system
NAME              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
traefik-ingress   2         2         2       2            2           <none>          107s
```

3.配置外部 nginx 负载均衡

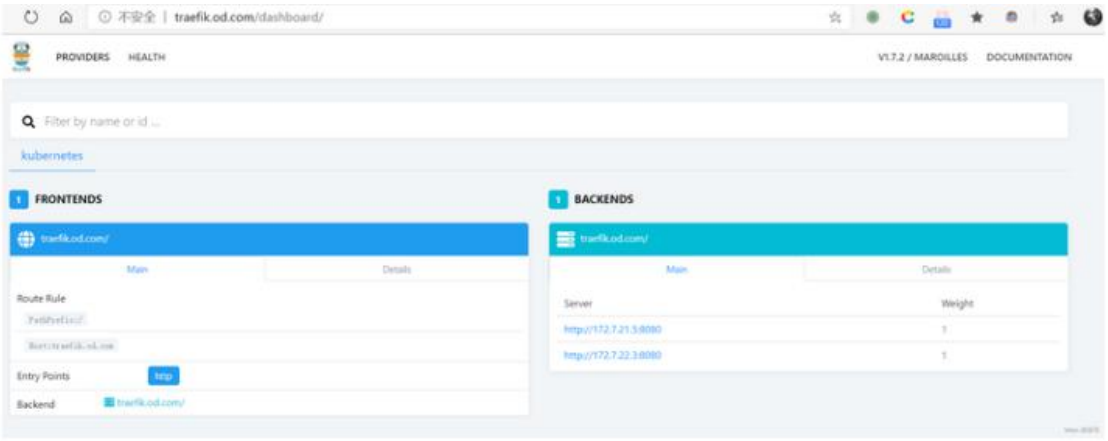
* 在 hdss7-11,hdss7-12 配置 nginx L7 转发

```
[root@hdss7-11 ~]# vim /etc/nginx/conf.d/od.com.conf
server {
    server_name *.od.com;

    location / {
        proxy_pass http://default_backend_traefik;
        proxy_set_header Host       $http_host;
        proxy_set_header x-forwarded-for $proxy_add_x_forwarded_for;
    }
}

upstream default_backend_traefik {
    # 所有的nodes都放到upstream中
    server 10.4.7.21:81    max_fails=3 fail_timeout=10s;
    server 10.4.7.22:81    max_fails=3 fail_timeout=10s;
}
[root@hdss7-11 ~]# nginx -tq && nginx -s reload
```

* 配置 dns 解析

```
[root@hdss7-11 ~]# vim /var/named/od.com.zone
$ORIGIN od.com.
$TTL 600  ; 10 minutes
@       IN SOA  dns.od.com. dnsadmin.od.com. (
        2020011302 ; serial
        10800      ; refresh (3 hours)
        900        ; retry (15 minutes)
        604800     ; expire (1 week)
        86400      ; minimum (1 day)
        )
        NS   dns.od.com.
$TTL 60 ; 1 minute
dns                A    10.4.7.11
harbor             A    10.4.7.200
k8s-yaml           A    10.4.7.200
traefik            A    10.4.7.10
[root@hdss7-11 ~]# systemctl restart named
```

* 查看 traefik 网页



5.4 Dashboard

1. 配置资源清单

清单文件存放到 hdss7-200:/data/k8s-yaml/dashboard/dashboard_1.10.1

* 准备镜像

```
# 镜像准备
# 因不可描述原因，无法访问 k8s.gcr.io，改成 registry.aliyuncs.com/google_containers
[root@hdss7-200 ~]# docker image pull registry.aliyuncs.com/google_containers/kubernetes-dashboard-amd64:v1.10.1
[root@hdss7-200 ~]# docker image tag f9aed6605b81 harbor.od.com/public/kubernetes-dashboard-amd64:v1.10.1
[root@hdss7-200 ~]# docker image push harbor.od.com/public/kubernetes-dashboard-amd64:v1.10.1
```

* rbac.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
    addonmanager.kubernetes.io/mode: Reconcile
  name: kubernetes-dashboard-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard-admin
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    addonmanager.kubernetes.io/mode: Reconcile
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard-admin
  namespace: kube-system
```

* deployment.yaml

```yaml
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: traefik-ingress
  namespace: kube-system
  labels:
    k8s-app: traefik-ingress
spec:
  template:
    metadata:
      labels:
        k8s-app: traefik-ingress
        name: traefik-ingress
    spec:
      serviceAccountName: traefik-ingress-controller
      terminationGracePeriodSeconds: 60
      containers:
      - image: harbor.od.com/public/traefik:v1.7.2
        name: traefik-ingress
        ports:
        - name: controller
          containerPort: 80
          hostPort: 81
        - name: admin-web
          containerPort: 8080
        securityContext:
          capabilities:
            drop:
            - ALL
            add:
            - NET_BIND_SERVICE
        args:
        - --api
        - --kubernetes
        - --logLevel=INFO
        - --insecureskipverify=true
        - --kubernetes.endpoint=https://10.4.7.10:7443
        - --accesslog
        - --accesslog.filepath=/var/log/traefik_access.log
        - --traefiklog
        - --traefiklog.filepath=/var/log/traefik.log
        - --metrics.prometheus
```

* service.yaml

```yaml
kind: Service
apiVersion: v1
metadata:
  name: traefik-ingress-service
  namespace: kube-system
spec:
  selector:
    k8s-app: traefik-ingress
  ports:
    - protocol: TCP
      port: 80
      name: controller
    - protocol: TCP
      port: 8080
      name: admin-web
```

* ingress.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: traefik-web-ui
  namespace: kube-system
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: traefik.od.com
    http:
      paths:
      - path: /
        backend:
          serviceName: traefik-ingress-service
          servicePort: 8080
```

* 准备镜像

[root@hdss7-200 traefik_1.7.2]# docker pull traefik:v1.7.2-alpine
[root@hdss7-200 traefik_1.7.2]# docker image tag traefik:v1.7.2-alpine harbor.od.com/public/traefik:v1.7.2
[root@hdss7-200 traefik_1.7.2]# docker push harbor.od.com/public/traefik:v1.7.2

2.交付 traefik 到 k8s

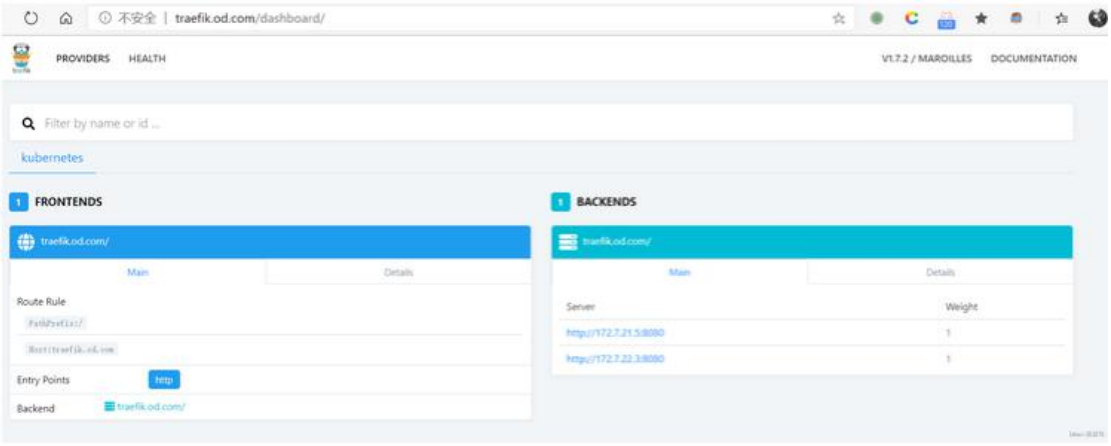
* 在 hdss7-11,hdss7-12 配置 nginx L7 转发

```
[root@hdss7-11 ~]# vim /etc/nginx/conf.d/od.com.conf
server {
    server_name *.od.com;

    location / {
        proxy_pass http://default_backend_traefik;
        proxy_set_header Host       $http_host;
        proxy_set_header x-forwarded-for $proxy_add_x_forwarded_for;
    }
}

upstream default_backend_traefik {
    # 所有的nodes都放到upstream中
    server 10.4.7.21:81    max_fails=3 fail_timeout=10s;
    server 10.4.7.22:81    max_fails=3 fail_timeout=10s;
}
[root@hdss7-11 ~]# nginx -tq && nginx -s reload
```

* 配置 dns 解析

```
[root@hdss7-11 ~]# vim /var/named/od.com.zone
$ORIGIN od.com.
$TTL 600  ; 10 minutes
@       IN SOA  dns.od.com. dnsadmin.od.com. (
        2020011302 ; serial
        10800      ; refresh (3 hours)
        900        ; retry (15 minutes)
        604800     ; expire (1 week)
        86400      ; minimum (1 day)
        )
        NS   dns.od.com.
$TTL 60 ; 1 minute
dns                A    10.4.7.11
harbor             A    10.4.7.200
k8s-yaml           A    10.4.7.200
traefik            A    10.4.7.10
[root@hdss7-11 ~]# systemctl restart named
```

* 查看 traefik 网页



5.5 Dashboard

1. 配置资源清单

清单文件存放到 hdss7-200:/data/k8s-yaml/dashboard/dashboard_1.10.1

* 准备镜像

```
# 镜像准备
# 因不可描述原因，无法访问 k8s.gcr.io，改成 registry.aliyuncs.com/google_containers
[root@hdss7-200 ~]# docker image pull registry.aliyuncs.com/google_containers/kubernetes-dashboard-amd64:v1.10.1
[root@hdss7-200 ~]# docker image tag f9aed6605b81 harbor.od.com/public/kubernetes-dashboard-amd64:v1.10.1
[root@hdss7-200 ~]# docker image push harbor.od.com/public/kubernetes-dashboard-amd64:v1.10.1
```

* rbac.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
    addonmanager.kubernetes.io/mode: Reconcile
  name: kubernetes-dashboard-admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard-admin
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    addonmanager.kubernetes.io/mode: Reconcile
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard-admin
  namespace: kube-system
```

* deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      priorityClassName: system-cluster-critical
      containers:
      - name: kubernetes-dashboard
        image: harbor.od.com/public/kubernetes-dashboard-amd64:v1.10.1
        resources:
          limits:
            cpu: 100m
            memory: 300Mi
          requests:
            cpu: 50m
            memory: 100Mi
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          # PLATFORM-SPECIFIC ARGS HERE
          - --auto-generate-certificates
        volumeMounts:
        - name: tmp-volume
          mountPath: /tmp
        livenessProbe:
          httpGet:
            scheme: HTTPS
            path: /
            port: 8443
          initialDelaySeconds: 30
          timeoutSeconds: 30
      volumes:
      - name: tmp-volume
        emptyDir: {}
      serviceAccountName: kubernetes-dashboard-admin
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
```

* service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  selector:
    k8s-app: kubernetes-dashboard
  ports:
  - port: 443
    targetPort: 8443
```

* ingress.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: dashboard.od.com
    http:
      paths:
      - backend:
          serviceName: kubernetes-dashboard
          servicePort: 443
```

2. 交付 dashboard 到 k8s

[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/dashboard/dashboard_1.10.1/rbac.yaml
[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/dashboard/dashboard_1.10.1/deployment.yaml
[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/dashboard/dashboard_1.10.1/service.yaml
[root@hdss7-21 ~]# kubectl apply -f http://k8s-yaml.od.com/dashboard/dashboard_1.10.1/ingress.yaml

3. 配置 DNS 解析

```
[root@hdss7-11 ~]# vim /var/named/od.com.zone
$ORIGIN od.com.
$TTL 600  ; 10 minutes
@       IN SOA  dns.od.com. dnsadmin.od.com. (
        2020011303 ; serial
        10800      ; refresh (3 hours)
        900        ; retry (15 minutes)
        604800     ; expire (1 week)
        86400      ; minimum (1 day)
        )
        NS   dns.od.com.
$TTL 60 ; 1 minute
dns                A    10.4.7.11
harbor             A    10.4.7.200
k8s-yaml           A    10.4.7.200
traefik            A    10.4.7.10
dashboard          A    10.4.7.10
[root@hdss7-11 ~]# systemctl restart named.service
```

4.签发 SSL 证书

[root@hdss7-200 ~]# cd /opt/certs/

[root@hdss7-200 certs]# (umask 077; openssl genrsa -out dashboard.od.com.key 2048)

[root@hdss7-200 certs]# openssl req -new -key dashboard.od.com.key -out dashboard.od.com.csr -subj "/CN=dashboard.od.com/C=CN/ST=BJ/L=Beijing/O=OldboyEdu/OU=ops"

[root@hdss7-200 certs]# openssl x509 -req -in dashboard.od.com.csr -CA ca.pem -CAkey ca-key.pem -CAcreateserial -out dashboard.od.com.crt -days 3650

[root@hdss7-200 certs]# ll dashboard.od.com.*

-rw-r--r-- 1 root root 1196 Jan 29 20:52 dashboard.od.com.crt

-rw-r--r-- 1 root root 1005 Jan 29 20:51 dashboard.od.com.csr

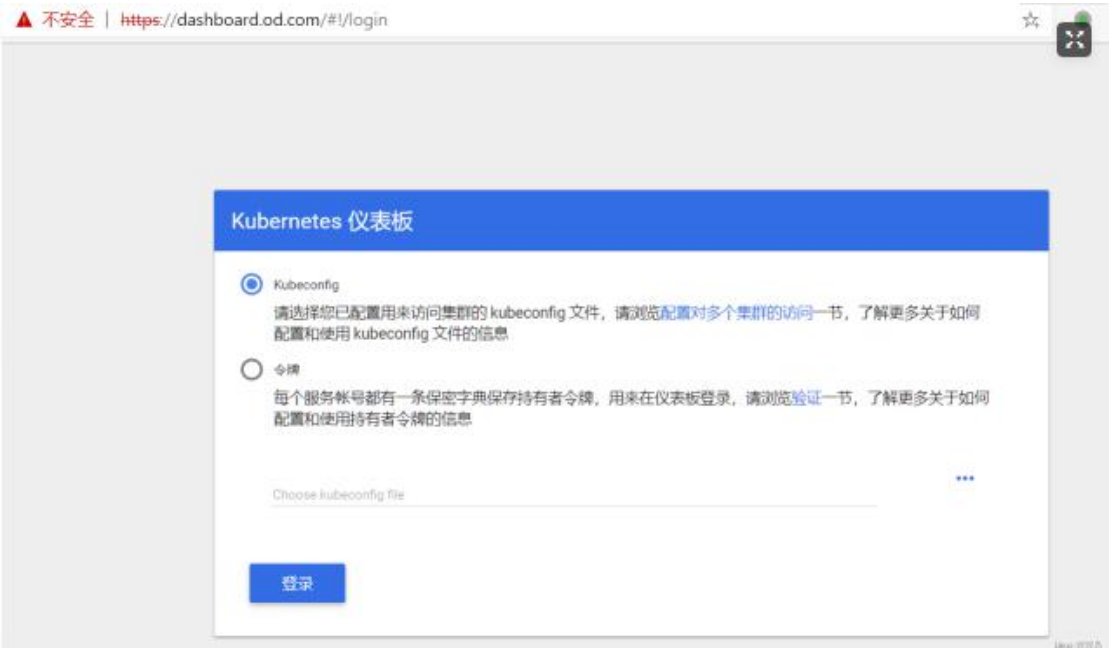
-rw------- 1 root root 1675 Jan 29 20:51 dashboard.od.com.key

[root@hdss7-200 certs]# scp dashboard.od.com.key dashboard.od.com.crt hdss7-11:/etc/nginx/certs/

[root@hdss7-200 certs]# scp dashboard.od.com.key dashboard.od.com.crt hdss7-12:/etc/nginx/certs/

5.配置 Nginx

```
# hdss7-11和hdss7-12都需要操作
[root@hdss7-11 ~]# vim /etc/nginx/conf.d/dashborad.conf
server {
    listen       80;
    server_name  dashboard.od.com;
    rewrite ^(.*)$ https://${server_name}$1 permanent;
}

server {
    listen       443 ssl;
    server_name  dashboard.od.com;

    ssl_certificate "certs/dashboard.od.com.crt";
    ssl_certificate_key "certs/dashboard.od.com.key";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass http://default_backend_traefik;
        proxy_set_header Host       $http_host;
        proxy_set_header x-forwarded-for $proxy_add_x_forwarded_for;
    }
}
[root@hdss7-11 ~]# nginx -t && nginx -s reload
```



6. 测试 token 登陆

```
[root@hdss7-21 ~]# kubectl get secret -n kube-system|grep kubernetes-dashboard-token
[root@hdss7-21 ~]# kubectl describe secret kubernetes-dashboard-token-hr5rj -n kube-system|grep ^token
```

6. 在 k8s 中部署 redash

6.1 制作 docker 镜像

1. 创建镜像制作目录

mkdir /opt/img_builder && cd /opt/img_builder

2. 克隆镜像

git clone https://github.com/dazdata/redash.git

3. Build 镜像

cd redash && docker build run -t harbor.dazdata.com/public/redash:v1 .

4. Push 镜像到 harbor 仓库

docker push harbor.dazdata.com/redash:v1

5. 从 dockerhub 拉取 pgsql 和 redis 镜像，并推送到 harbor

docker pull postgersql：9.6-alpine

docker pull redis：5.0-alpine

docker tag harbor.dazdata.com/public/postgresql:9.6-alpine

docker tag harbor.dazdata.com/public/redis:5.0-alpine

6.2 准备 k8s 资源配置清单

1. Postgresql 资源清单

注：测试用的本地硬盘挂在到 pgsql，正式环境需要用到网络硬盘，可以使用 nfs 或者 ceph 等网络存储

- Pv

[root@manager21 yaml]# cat postgres-pv.yaml

apiVersion: v1

kind: PersistentVolume

```
apiVersion: v1
metadata:
  name: postgres-claim0
  namespace: dazdata
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/pods/redash/postgres-data"
```

- Pvc

```
[root@manager21 yaml]# cat postgres-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    io.kompose.service: postgres-claim0
  name: postgres-claim0
  namespace: dazdata
spec:
  storageClassName: manual
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

- Dp

```
[root@manager21 yaml]# cat postgres-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    io.kompose.service: postgres
  name: postgres
  namespace: dazdata
spec:
  replicas: 1
```

```
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        io.kompose.service: postgres
    spec:
      containers:
      - env:
        - name: POSTGRES_PASSWORD
          value: E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ
        - name: PYTHONUNBUFFERED
          value: "0"
        - name: REDASH_COOKIE_SECRET
          value: yKCscHTaWKJDzoFFwg2zZRZvKZCbiUb0
        - name: REDASH_DATABASE_URL
          value:
postgresql://postgres:E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ@postgres/postgres
        - name: REDASH_LOG_LEVEL
          value: INFO
        - name: REDASH_REDIS_URL
          value: redis://redis:6379/0
        - name: REDASH_SECRET_KEY
          value: XUXC57MgLISMLoDNmEciPdgP9pYJs2EH
        image: harbor.dazdata.com/public/postgres:9.6-alpine
        name: postgres
        resources: {}
        volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgres-claim0
      restartPolicy: Always
      volumes:
      - name: postgres-claim0
        persistentVolumeClaim:
          claimName: postgres-claim0
```

- Svc

```
[root@manager21 yaml]# cat postgres-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    io.kompose.service: postgres
```

```
    name: postgres
    namespace: dazdata
spec:
    clusterIP: None
    selector:
        io.kompose.service: postgres
status:
    loadBalancer: {}
```

2. Redis 资源清单

- Dp

```
[root@manager21 yaml]# cat redis-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
    labels:
        io.kompose.service: redis
    name: redis
    namespace: dazdata
spec:
    replicas: 1
    strategy: {}
    template:
        metadata:
            labels:
                io.kompose.service: redis
        spec:
            containers:
            - image: harbor.dazdata.com/public/redis:5.0-alpine
                name: redis
                resources: {}
            restartPolicy: Always
```

- Svc

```
[root@manager21 yaml]# cat redis-service.yaml
apiVersion: v1
kind: Service
metadata:
    labels:
        io.kompose.service: redis
    name: redis
    namespace: dazdata
```

```
spec:
  clusterIP: None
  selector:
    io.kompose.service: redis
status:
  loadBalancer: {}
```

3. Redash-server 资源清单

- Dp

```
[root@manager21 yaml]# cat server-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    io.kompose.service: server
  name: server
  namespace: dazdata
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      labels:
        io.kompose.service: server
    spec:
      containers:
      - args:
        - server
        env:
        - name: POSTGRES_PASSWORD
          value: E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ
        - name: PYTHONUNBUFFERED
          value: "0"
        - name: REDASH_COOKIE_SECRET
          value: yKCscHTaWKJDzoFFwg2zZRZvKZCbiUb0
        - name: REDASH_DATABASE_URL
          value:
postgresql://postgres:E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ@postgres/postgres
        - name: REDASH_LOG_LEVEL
          value: INFO
        - name: REDASH_REDIS_URL
          value: redis://redis:6379/0
```

```
        - name: REDASH_SECRET_KEY
          value: XUXC57MgLlSMLoDNmEciPdgP9pYJs2EH
        - name: REDASH_WEB_WORKERS
          value: "4"
        image: harbor.dazdata.com/public/redash:v1
        name: server
        ports:
        - containerPort: 5000
        resources: {}
      restartPolicy: Always
```

- Svc

```
[root@manager21 yaml]# cat server-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    io.kompose.service: server
  name: server
  namespace: dazdata
spec:
  ports:
  - name: "5000"
    port: 5000
    targetPort: 5000
  selector:
    io.kompose.service: server
```

- Ingress

```
[root@manager21 yaml]# cat ingress.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: server
  namespace: dazdata
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: redash.dazhi.com
    http:
      paths:
```

```
      - backend:
          serviceName: server
          servicePort: 5000
```

4. Redash-scheduler 资源清单

- Dp

```
[root@manager21 yaml]# cat scheduler-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    io.kompose.service: scheduler
  name: scheduler
  namespace: dazdata
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      labels:
        io.kompose.service: scheduler
    spec:
      containers:
      - args:
        - scheduler
        env:
        - name: POSTGRES_PASSWORD
          value: E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ
        - name: PYTHONUNBUFFERED
          value: "0"
        - name: QUEUES
          value: celery
        - name: REDASH_COOKIE_SECRET
          value: yKCscHTaWKJDzoFFwg2zZRZvKZCbiUb0
        - name: REDASH_DATABASE_URL
          value:
postgresql://postgres:E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ@postgres/postgres
        - name: REDASH_LOG_LEVEL
          value: INFO
        - name: REDASH_REDIS_URL
          value: redis://redis:6379/0
        - name: REDASH_SECRET_KEY
```

```
          value: XUXC57MgLlSMLoDNmEciPdgP9pYJs2EH
        - name: WORKERS_COUNT
          value: "1"
        image: harbor.dazdata.com/public/redash:v1
        name: scheduler
        resources: {}
      restartPolicy: Always
```

- Svc

```
[root@manager21 yaml]# cat scheduler-service.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: scheduler
  name: scheduler
  namespace: dazdata
spec:
  clusterIP: None
  selector:
    io.kompose.service: scheduler
status:
  loadBalancer: {}
```

5. Redash-worker 资源清单

- Dp

```
[root@manager21 yaml]# cat worker-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    io.kompose.service: scheduled-worker
  name: scheduled-worker
  namespace: dazdata
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      labels:
        io.kompose.service: scheduled-worker
```

```
    spec:
      containers:
      - args:
        - worker
        env:
        - name: POSTGRES_PASSWORD
          value: E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ
        - name: PYTHONUNBUFFERED
          value: "0"
        - name: QUEUES
          value: scheduled_queries,schemas
        - name: REDASH_COOKIE_SECRET
          value: yKCscHTaWKJDzoFFwg2zZRZvKZCbiUb0
        - name: REDASH_DATABASE_URL
          value:
postgresql://postgres:E2DHh1ZrbNeAciyV0sf6AjJ7OkeaBTPJ@postgres/postgres
        - name: REDASH_LOG_LEVEL
          value: INFO
        - name: REDASH_REDIS_URL
          value: redis://redis:6379/0
        - name: REDASH_SECRET_KEY
          value: XUXC57MgLISMLoDNmEciPdgP9pYJs2EH
        - name: WORKERS_COUNT
          value: "1"
        image: harbor.dazdata.com/public/redash:v1
        name: scheduled-worker
        resources: {}
      restartPolicy: Always
```

- Svc

```
[root@manager21 yaml]# cat worker-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    io.kompose.service: scheduled-worker
  name: scheduled-worker
  namespace: dazdata
spec:
  clusterIP: None
  selector:
    io.kompose.service: scheduled-worker
```

7. 部署 k8s 资源

7.1 按资源配置清单顺序 apply yaml 文件

```
kubectl apply  -f  postgres-pv.yaml
kubectl apply  -f  postgres-pvc.yaml
kubectl apply  -f  postgres-deployment.yaml
kubectl apply  -f  postgres-service.yaml
kubectl apply  -f  server-deployment.yaml
kubectl apply  -f  server-service.yaml
kubectl apply  -f  ingress.yaml
kubectl apply  -f  scheduler-deployment.yaml
kubectl apply  -f  scheduler-service.yaml
kubectl apply  -f  worker-deployment.yaml
kubectl apply  -f  worker-service.yaml
```

添加 dns 记录到 dns server，记录域名为 ingress 配置域名，A 记录为 proxyserver 的 vip。
在浏览器直接访问域名就进入 redash

第九章：配置常见数据源

1、Centos8 连接 oracle 数据源

1)、执行命令安装 mysql:

sudo yum install mysql

2)、验证是否安装成功:

which mysql
which mysqldump

3)、登陆进入 mysql 数据库,创建远程登录用户并授权:

例子:(其中 root 是登陆的用户,123456 是登陆密码)

GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;

4)、然后开启 MySQL 的远程登录账号:

flush privileges;

5)、查看你的 mysql 服务器 ip 地址,获取到 ip 地址之后进入 centos8 下执行:

(root 代表 mysql 数据库登陆用户名,123456 代表 mysql 数据库登陆密码,192.168.137.1 代表 mysql 数据库 ip 地址,3306 是端口,test 是数据库)

mysql -u root -p123456 -h 192.168.137.1 -P 3306 -D test

6)、安装 mysql 数据源在 redash 中可以连接

1.在 centos8 命令行安装 mysql-devel:

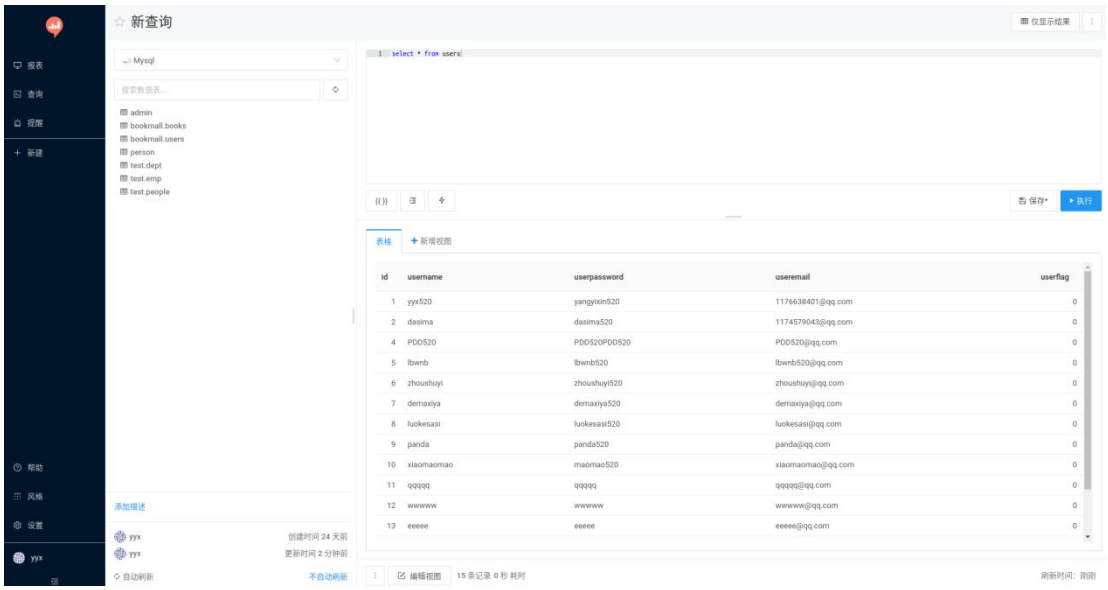
sudo yum install mysql-devel

2.进入 redash 目录下执行:

source venv/bin/activate
pip3 install mysqlclient

三.运行完成即可登陆 redash 在新建数据源里面选择 mysql 数据源填入相关参数即可

连接成功就可以在查询里面进行查询



2、Centos8 连接 oracle 数据源

1)、首先从 oracle 官网下载 3 个安装包

oracle-instantclient11.2-basic-11.2.0.4.0.x86_64.zip
oracle-instantclient11.2-sqlplus-11.2.0.4.0.x86_64.zip
oracle-instantclient11.2-sdk-11.2.0.4.0.x86_64.zip

2)、创建/opt/oracle 文件夹，赋予权限

sudo mkdir /opt/oracle
sudo chmod 777 /opt/oracle

将上述包解压至/opt/oracle/instantclient_11_2/文件夹下

3)、配置用户变量，在~/.bashrc 文件最后追加环境变量配置信息,在/etc/profile 里面最好也加上

ORACLE_HOME=/opt/oracle/instantclient_11_2
PATH=$ORACLE_HOME:$PATH
LD_LIBRARY_PATH=$ORACLE_HOME:$LD_LIBRARY_PATH
export TNS_ADMIN=$ORACLE_HOME/network/admin
export NLS_LANG=AMERICAN_AMERICA.AL32UTF8
export ORACLE_HOME PATH LD_LIBRARY_PATH

配置完毕后,刷新使环境变量生效 source ~/.bashrc 还有 source /etc/profile

4)、创建软连接:

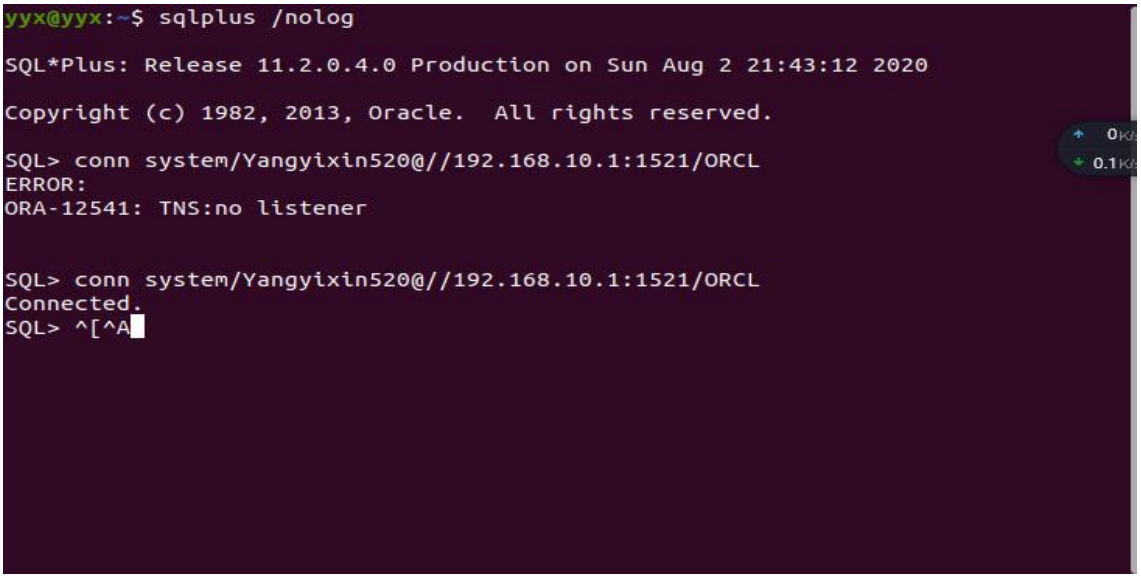
ln -f -s /opt/oracle/instantclient_11_2/lclntsh.so.11.1 /usr/lib/lclntsh.so
ln -f -s /opt/oracle/instantclient_11_2/libclntsh.so.11.1 /usr/lib/libclntsh.so

Ubuntu 安装:sudo apt-get install libaio-dev

Centos8 安装:sudo yum install libnsl.x86_64

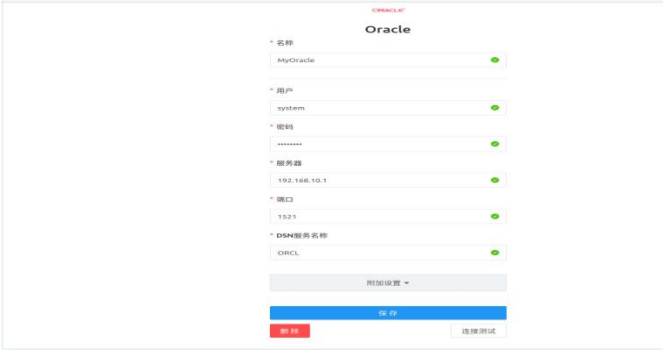
5).同时在本地需要修改 oracle 的 tnsnames.ora 文件和 listener.ora 文件,修改他们的监听 ip

6).测试: 执行 sqlplus /nolog 连接本地 oracle 出现 Connected 则成功



7. 在虚拟环境中执行安装 oracle 数据源:pip3 install -r requirements_oracle_ds.txt

8、 在 Redash 界面中配置数据源如图:



3、Centos8 安装并连接 mysql 数据源：

1).安装 mysql 并且在 centos8 里面可以访问

1.在命令行执行安装 mysql:

sudo yum install mysql

2.验证是否安装成功:

which mysql

which mysqldump

3.登陆进入 mysql 数据库,创建

远程登录用户并授权:

例子:(其中 root 是登陆的用户,123456 是登陆密码)

GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;

4.然后开启 MySQL 的远程登录账号:

flush privileges;

5.查看你的 mysql 服务器 ip 地址,获取到 ip 地址之后进入 centos8 下命令行输入:

(root 代表 mysql 数据库登陆用户名,123456 代表 mysql 数据库登陆密码,192.168.137.1 代表 mysql 数据库 ip 地址,3306 是端口,test 是数据库)

mysql -u root -p123456 -h 192.168.137.1 -P 3306 -D test

2).安装 mysql 数据源在 redash 中可以连接

1.在 centos8 命令行安装 mysql-devel:

sudo yum install mysql-devel

2.进入 redash 目录下执行:

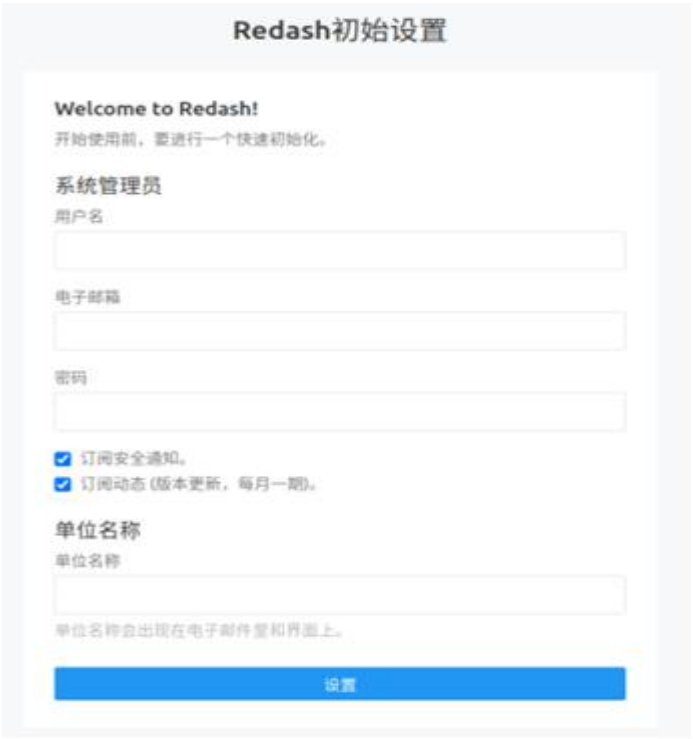
source venv/bin/activate

pip3 install mysqlclient

3）.运行完成即可登陆 redash 在新建数据源里面选择 mysql 数据源填入相关参数即

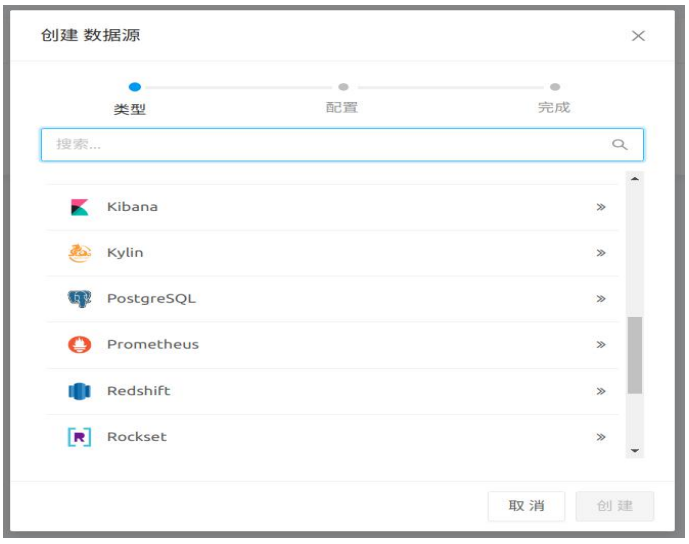
第十章：初次使用配置

Redash 安装完毕登陆 localhost:5000 后出现初始设置页面，如果出现则说明安装配置成功：



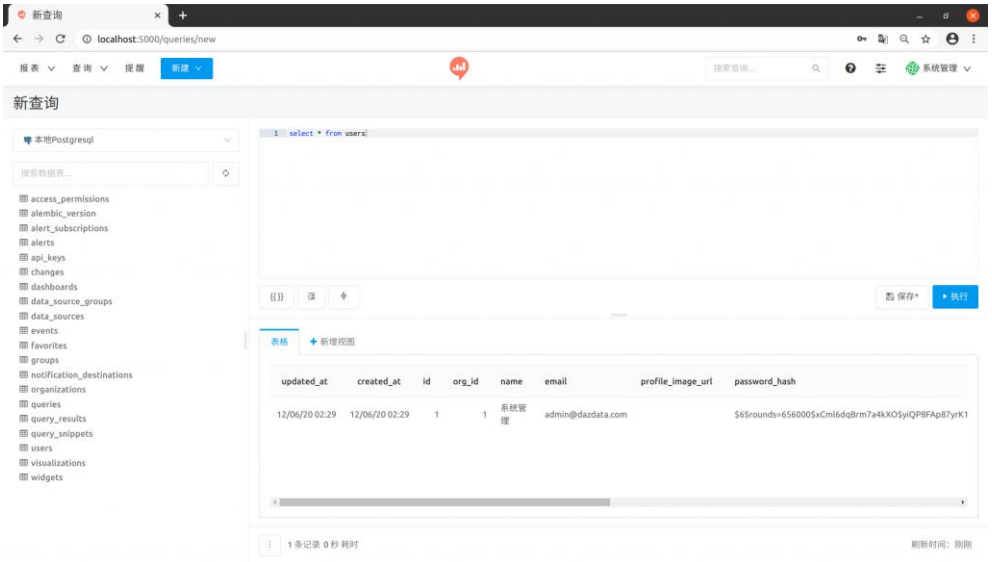
初始设置，输入系统管理员用户名（可以是昵称）、邮箱（登录用）、密码，和单位名称即可，非常简单，要牢记管理员密码。

登录进入系统界面后，找右上角下拉菜单，选择“数据源”

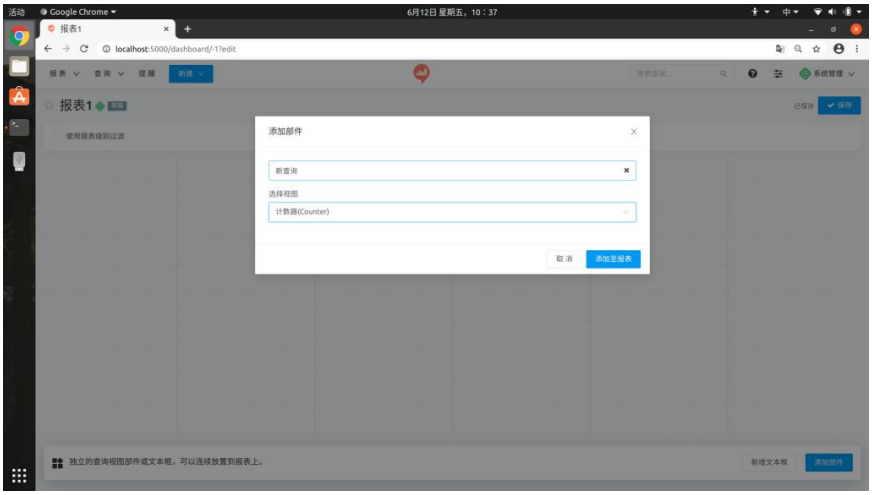


可以先用 Redash 自身的 PostgreSQL 数据库作为数据源试试看：单机安装下，服务器：localhost，端口：5432，用户名：postgres，密码：安装脚本中输入的密码，数据库：postgres，保存测试连接；Docker 安装下，服务器：redash_postgres_1，端口：5432，用户名：postgres，密码：<空>，数据库：postgres，创建->保存->测试连接。

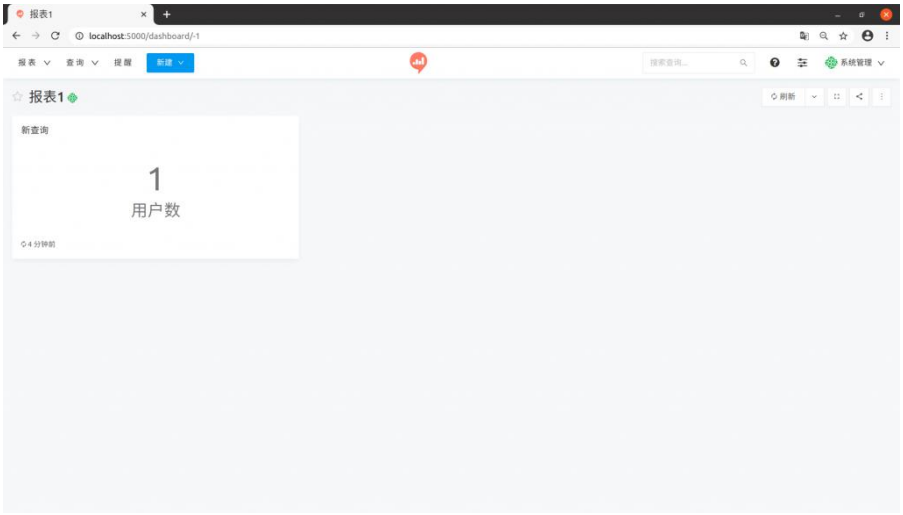
建好数据源后下一步就是创建查询，点顶部“新建->新建查询”菜单：



选刚才建立的新查询，选计数器(Counter)类型，输入标题，选统计行数，一切 ok 了。下一步便是建立报表了，选“新建->新建报表”菜单，输入报表名称，点添加部件，选中刚才建立的计数器：



一幅极简所见即所得的报表便出来了，恭喜，你已经搞定 Redash 了，高大上的产品其实非常简单有没有!



附件 1：Redash 环境变量清单

名称	描述	默认值
REDASH_REDIS_URL	**Redis 连接字符串**	**redis://localhost：6379/0**
REDASH_PROXIES_COUNT		1
REDASH_STATSD_HOST	状态监控	127.0.0.1
REDASH_STATSD_PORT	状态监控端口	8125
REDASH_STATSD_PREFIX	状态监控信息前缀	redash
REDASH_STATSD_USE_TAGS	是否使用 StatsD 指标中的标签（InfluxDB 的格式）	false
REDASH_DATABASE_URL 或 DATABASE_URL	**postgresql:///postgres 表示以当前 linux 用户登录 postgresql 数据库**	**postgresql:///postgres 表示以当前 linux 用户无密码，登录本机 postgres 实例**
REDASH_QUERY_RESULTS_CLEANUP_ENABLED	是否自动清除查询的结果集	true
REDASH_QUERY_RESULTS_CLEANUP_COUNT	自动清除上限数量（个）：每查询结果集超过该数量即自动清除。Redash 会为每查询保留多个结果集，每次执行 sql 都会保留。	100
REDASH_QUERY_RESULTS_CLEANUP_MAX_AGE	自动清除时间下限(天)：超过天数自动清除	7
REDASH_SCHEMAS_REFRESH_SCHEDULE	刷新数据源数据结构(表字段等模式信息）的频率（以分钟为单位）	30
REDASH_AUTH_TYPE	redash 集成授权模式	api_key
REDASH_ENFORCE_HTTPS	强制使用 https 链接	false
REDASH_INVITATION_TOKEN_MAX_AGE	邀请用户注册链接失效时间：秒	60 _ 60 _ 24 * 7
REDASH_MULTI_ORG	多租户部署	false
REDASH_GOOGLE_CLIENT_ID	集成 google 账户	
REDASH_GOOGLE_CLIENT_SECRET	集成 google 账户	
REDASH_REMOTE_USER_LOGIN_ENABLED	允许远程登录	false
REDASH_REMOTE_USER_HEADER	远程登录请求头信息	X-Forwarded-Remote-User
REDASH_LDAP_LOGIN_ENABLED	允许 LDAP 登录	false
REDASH_LDAP_URL	LDAP 链接	none
REDASH_LDAP_BIND_DN	LDAP 绑定域名	none
REDASH_LDAP_BIND_DN_PASSWORD	LDAP 绑定密码	
REDASH_LDAP_DISPLAY_NAME_KEY	LDAP 显示名称	displayName
REDASH_LDAP_EMAIL_KEY	LDAP 邮箱	mail

名称	描述	默认值
REDASH_LDAP_CUSTOM_USERNAME_PROMPT	LDAP 用户名称提示	LDAP/AD/SSO username:
REDASH_LDAP_SEARCH_TEMPLATE	LDAP 搜索模板	(cn=%(username)s)
REDASH_LDAP_SEARCH_DN	LDAP 搜索域名	REDASHSEARCHDN
REDASH_STATIC_ASSETS_PATH	**默认前端资源文件位置**	**../client/dist/**
REDASH_JOB_EXPIRY_TIME	队列查询任务过期时间：秒	3600 * 12
REDASH_COOKIE_SECRET	Cookie 信息加密密钥	c292a0a3aa32397cdb050e233733900f
REDASH_LOG_LEVEL	日志级别	INFO
REDASH_MAIL_SERVER	邮件服务器地址	localhost
REDASH_MAIL_PORT	邮件服务器端口	25
REDASH_MAIL_USE_TLS	邮件服务器启用 TLS	false
REDASH_MAIL_USE_SSL	邮件服务器启用 SSL	false
REDASH_MAIL_USERNAME	邮件服务器用户名	None
REDASH_MAIL_PASSWORD	邮件服务器密码	None
REDASH_MAIL_DEFAULT_SENDER	邮件服务器默认发件人	None
REDASH_MAIL_MAX_EMAILS	邮件服务器最大邮件数	None
REDASH_MAIL_ASCII_ATTACHMENTS	邮件服务器 ASCII 附件	false
REDASH_HOST	Redash Server 地址	
REDASH_ALERTS_DEFAULT_MAIL_SUBJECT_TEMPLATE	预警信息邮件主题默认模板	（{state}）{alert_name}
REDASH_THROTTLE_LOGIN_PATTERN	登录频次限制	50/hour
REDASH_LIMITER_STORAGE		REDIS_URL
REDASH_CORS_ACCESS_CONTROL_ALLOW_ORIGIN	跨请求允许的域	
REDASH_CORS_ACCESS_CONTROL_ALLOW_CREDENTIALS	跨域请求需要证书	false
REDASH_CORS_ACCESS_CONTROL_REQUEST_METHOD	允许跨域请求类型	GET，POST，PUT
REDASH_CORS_ACCESS_CONTROL_ALLOW_HEADERS	跨域请求头信息	Content-Type
REDASH_ENABLED_QUERY_RUNNERS	允许附加的数据源查询执行程序	",".join(defaultqueryrunners)
REDASH_ADDITIONAL_QUERY_RUNNERS	附加的数据源查询执行程序	
REDASH_DISABLED_QUERY_RUNNERS	禁止的数据源查询执行程序	
REDASH_ADHOC_QUERY_TIME_LIMIT	ADHOC 查询时间限制	None
REDASH_ENABLED_DESTINATIONS		",".join(default_destinations)
REDASH_ADDITIONAL_DESTINATIONS		
REDASH_EVENT_REPORTING_WEBHOOKS		

名称	描述	默认值
REDASH_SENTRY_DSN	Sentry 事件日志	
REDASH_ALLOW_SCRIPTS_IN_USER_INPUT	用户输入禁用脚本	false
REDASH_DASHBOARD_REFRESH_INTERVALS	配置可用的报表刷新频率(秒)	“60,300,600,1800,3600,43200,86400”
REDASH_QUERY_REFRESH_INTERVALS	配置可用的查询刷新频率(秒)	“60,300,600,900,1800,3600,7200,10800,14400,18000,21600,25200,28800,32400,36000,39600,43200,86400,604800,1209600,2592000”
REDASH_PASSWORD_LOGIN_ENABLED	允许用户名密码方式登录	true
REDASH_SAML_METADATA_URL		
REDASH_SAML_ENTITY_ID		
REDASH_SAML_NAMEID_FORMAT		
REDASH_DATE_FORMAT	默认日期格式：	DD / MM / YY
REDASH_JWT_LOGIN_ENABLED		假
REDASH_JWT_AUTH_ISSUER		
REDASH_JWT_AUTH_PUBLIC_CERTS_URL		
REDASH_JWT_AUTH_AUDIENCE		
REDASH_JWT_AUTH_ALGORITHMS		HS256，RS256，ES256
REDASH_JWT_AUTH_COOKIE_NAME		
REDASH_JWT_AUTH_HEADER_NAME		
REDASH_FEATURE_SHOW_QUERY_RESULTS_COUNT	显示查询结果的数量	true
REDASH_VERSION_CHECK	版本检查	true
REDASH_FEATURE_DISABLE_REFRESH_QUERIES	禁用查询刷新	false
REDASH_FEATURE_SHOW_PERMISSIONS_CONTROL	显示权限明细	false
REDASH_FEATURE_ALLOW_CUSTOM_JS_VISUALIZATIONS	允许自定义可视化类型	false
REDASH_FEATURE_DUMB_RECENTS		false
REDASH_FEATURE_AUTO_PUBLISH_NAMED_QUERIES	查询更名后自动发布	true
REDASH_BIGQUERY_HTTP_TIMEOUT	查询链接超时(秒)	600
REDASH_SCHEMA_RUN_TABLE_SIZE_CALCULATIONS	表结构显示记录数	False
REDASH_WEB_WORKERS	默认 Worker 进程数量	4

附件 2：从开源版本替换为商业版

开源版本配置好的环境变量，替换过程不受影响，替换完成后继续有效。
同时使用提供的打包好的商业版前后端程序包进行覆盖升级,期间不需要执行 npm run build,
直接使用

1、升级成商业版数据库

开源版数据库一定要先备份再开始升级！如果想清空开源数据库再执行升级，请先执行 2 清除数据库记录脚本。下列脚本中的 redash 必须改为数据库登录角色。

```
//枚举
CREATE TABLE public.enums (
    id serial,
    org_id integer default 0,
    code character varying(100) NOT NULL,
    name character varying(255) NOT NULL,
    is_builtin boolean default false,
    items    json[] DEFAULT '{}'::json[]
);

ALTER TABLE public.enums
ADD CONSTRAINT enums_pkey PRIMARY KEY(id);
CREATE unique INDEX enums_org_code ON public.enums USING btree (org_id,code);
ALTER TABLE public.enums OWNER TO redash;

//门户
CREATE TABLE public.dashboard_groups (
id serial,
dashboard_id integer NOT NULL,
group_id integer NOT NULL
);

ALTER TABLE ONLY public.dashboard_groups ADD CONSTRAINT dashboard_groups_pkey PRIMARY KEY (id);
ALTER TABLE ONLY public.dashboard_groups ADD CONSTRAINT dashboard_groups_dashboard_id_fkey FOREIGN KEY (dashboard_id) REFERENCES public.dashboards(id);
ALTER TABLE ONLY public.dashboard_groups ADD CONSTRAINT dashboard_groups_group_id_fkey FOREIGN KEY (group_id) REFERENCES public.groups(id);
CREATE UNIQUE INDEX dashboard_groups_dashboard_group ON public.dashboard_groups USING btree (dashboard_id, group_id);

ALTER TABLE public.dashboard_groups OWNER TO redash;
```

```
CREATE TABLE public.portal (
        id serial,
        org_id integer default 0,
        code character varying(100) NOT NULL,
        name character varying(255) NOT NULL,
        is_builtin boolean default false,
        items    json[] DEFAULT '{}'::json[]
);

ALTER TABLE public.portal ADD CONSTRAINT portal_pkey PRIMARY KEY(id);
CREATE unique INDEX portal_org_code ON public.portal USING btree (org_id,code);
ALTER TABLE public.portal OWNER TO redash;

//用户
ALTER TABLE public.users add column mobile character varying(15);
CREATE UNIQUE INDEX users_org_id_mobile ON public.users USING btree (org_id, mobile);
```

2、清除数据库记录

如果是开源版数据库，请注释前三行。一定要先做好备份才能清除记录!

```
delete from enums;
delete from portal;
delete from dashboard_groups;

delete from data_source_groups;
delete from groups;

delete from events;
delete from changes;
delete from widgets;
delete from dashboards;
delete from visualizations;
delete from queries;
delete from api_keys;
delete from users;

delete from query_results;
delete from data_sources;
delete from organizations;

delete from query_snippets;
delete from notification_destinations;
delete from favorites;
```

delete from alert_subscriptions;
delete from alerts;
delete from alembic_version;
delete from access_permissions;

3、安装商业版前端

进入开源版本~/安装目录/client/dist 文件夹，复制商业版本前端打包后的文件夹 dist 到此位置覆盖即可。前提条件是后端 API 服务器部署在 localhost:5000，如果不是该默认配置，则需要告诉武汉达之云公司单独打包。

4、安装商业版后端

商业版后端包括 web server 后端、worker、scheduler 三种应用环境，一般情况这三种应用环境都等同配置。

先删除开源版后端~/安装目录/redash/文件夹，复制商业版本的该文件至此位置即可。商业版本后端部分文件为*.so 库文件格式，为获得准确的库文件包，需要先告诉武汉达之云公司本次安装所在的服务器 Linux 系统名称和版本号。

对于 Excel 插件，还需要安装几个 python 依赖包，拷贝 requirements_excel.txt 文件至~/安装目录/，执行:

source venv/bin/activate
pip install -r requirements_excel.txt
deactivate

5、配置商业版许可证

商业版本许可证是根据机器物理信息、数据库连接信息、最大用户数量和试用到期日期等因素计算出来的，需要许可证的地方会显示这些硬件信息，请将这些信息告诉武汉达之云公司，以获得许可证。未获得许可证的商业版功能将不能运行，但不影响开源版本所有功能。

(1) 配置许可证步骤:

在/etc/profile 文件最后添加环境变量:

export REDASH_DAZ_COMMON="此处为本机许可证"
export REDASH_DAZ_DATE="使用终止日期"

(2)保存并且使配置生效

source /etc/profile

附件 3：商业版配置弹窗跨域访问

1、适用于 Widget 插件和 Portal 插件

Readsh 具有极高的安全性，为防止跨站请求伪造（CSRF），Redash 里的 iframe 只允许访问可信网站，默认值下只能访问 redash.io。

要让弹窗可以访问信任网站，要具备两个条件：其一是目标网站允许来自 redash 安装网站的 iframe 访问；其二是 redash 安装网站配置目标网站为信任网站。

2、目标网站配置

目标网站允许 iframe 访问，这需要配置 www 服务器的 X-Frame-Options 四个参数:

1、ALLOWALL

表示该页面允许所有网站在 frame 中展示，例如：www.sohu.com 就允许被任何网站进行 iframe 调用。

2、DENY

表示该页面不允许在 frame 中展示，即便是在相同域名的页面中嵌套也不允许。

3、SAMEORIGIN

表示该页面可以在相同域名页面的 frame 中展示。

4、ALLOW-FROM uri

表示该页面可以在指定来源的 frame 中展示。

换一句话说，如果设置为 DENY，不光在别人的网站 frame 嵌入时会无法加载，在同域名页面中同样会无法加载。另一方面，如果设置为 SAMEORIGIN，那么页面就可以在同域名页面的 frame 中嵌套。要允许 Redash 访问，必须在目标网站配置 ALLOWALL 或 ALLOW-FROM 允许 redash 安装网站来进行 iframe 访问。

Nginx 配置

可以添加到 'http', 'server' 或者 'location' 配置项的节中，一般配置在'server' 中默认情况下都是使用 SAMEORIGIN 参数，允许同域嵌套

```
add_header X-Frame-Options SAMEORIGIN;
```

允许单个域名 iframe 嵌套

```
add_header X-Frame-Options ALLOW-FROM http://portal.dazdata.com/;
```

允许多个域名 iframe 嵌套，注意这里是用逗号分隔

```
add_header X-Frame-Options "ALLOW-FROM http://portal.dazdata.com/,http://previw.dazdata.com/";
```

3、Redash 安装网站配置

Redash 网站网站有一个隐含环境变量来进行配置 iframe 信任网站。

环境变量名：REDASH_CONTENT_SECURITY_POLICY

默认值类似于："default-src 'self' *; worker-src blob:; child-src blob:; style-src 'self' 'unsafe-inline' *; script-src 'self' 'unsafe-eval' *; font-src 'self' data:; img-src 'self' http: https:

data: blob:; object-src 'none'; frame-ancestors 'none'; frame-src redash.io;",

通过设置环境变量支持更多的信任网站，可以在/etc/profile 或~/.bashrc 里增加环境变量设置如下：

export REDASH_CONTENT_SECURITY_POLICY="default-src 'self' *; worker-src blob:; child-src blob:; style-src 'self' 'unsafe-inline' *; script-src 'self' 'unsafe-eval' *; font-src 'self' data:; img-src 'self' http: https: data: blob:; object-src 'none'; frame-ancestors 'none'; frame-src redash.io *.sohu.com;"

就是复制默认值，在 frame-src 节增加空格和新的信任域名，支持通配符号，如此类推可以支持更多的信任域名,例如:*.sohu.com *.baidu.com *.bilibili.com 等等

由于 Redash 本身有很多的帮助指向 redash.io，建议不要删除该信任域，否则导致很多帮助不可用。

附件 4：配置上传文件路径

Readsh 中文商业版 Excel 插件具备上传本地 Excel、CSV、Json 文件至服务器的能力。REDASH_DAZ_FILES 环境变量配置上传文件在服务器端的路径，支持相对路径，改环境变量默认为~/redash/client/dist/files/。单机部署情况下，可改可不改，在多机集群部署方式下，上传文件必须集中存放，不能分散于各个服务器，所以需要配置该环境变量，指向集中内网文件路径。

附件 5：Redash 中文版配置邮件服务

Redash 中文版可以使用自己架设的邮件服务器，也可以使用 web 公共邮件系统的 SMTP 服务。的此处以腾讯企业邮箱为例说明 redash 中文版如何配置邮件发件服务。在环境变量设置相应参数，进入 redash 中文版服务器在/etc/profile 中加入:

export REDASH_MAIL_SERVER=hwsmtp.exmail.qq.com

export REDASH_MAIL_PORT=465

export REDASH_MAIL_USE_SSL=true

export REDASH_MAIL_USERNAME=你的 qq 企业邮箱登陆用户名

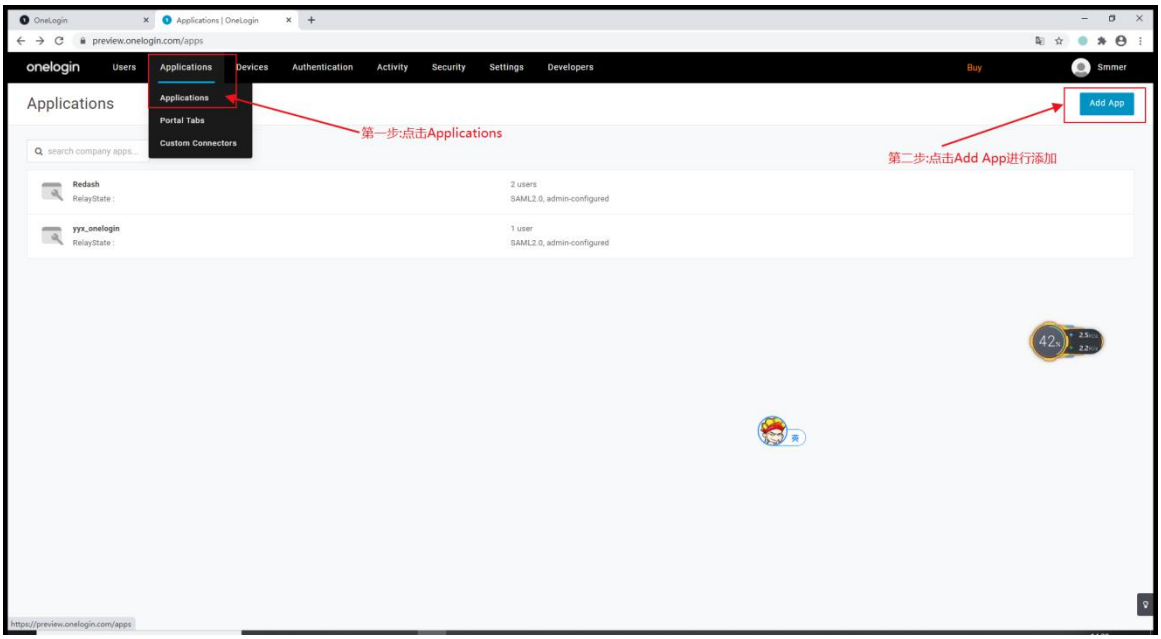
export REDASH_MAIL_PASSWORD=你的 qq 企业邮箱登陆密码

export REDASH_MAIL_DEFAULT_SENDER=默认的发送邮箱(一般就是登陆邮箱)

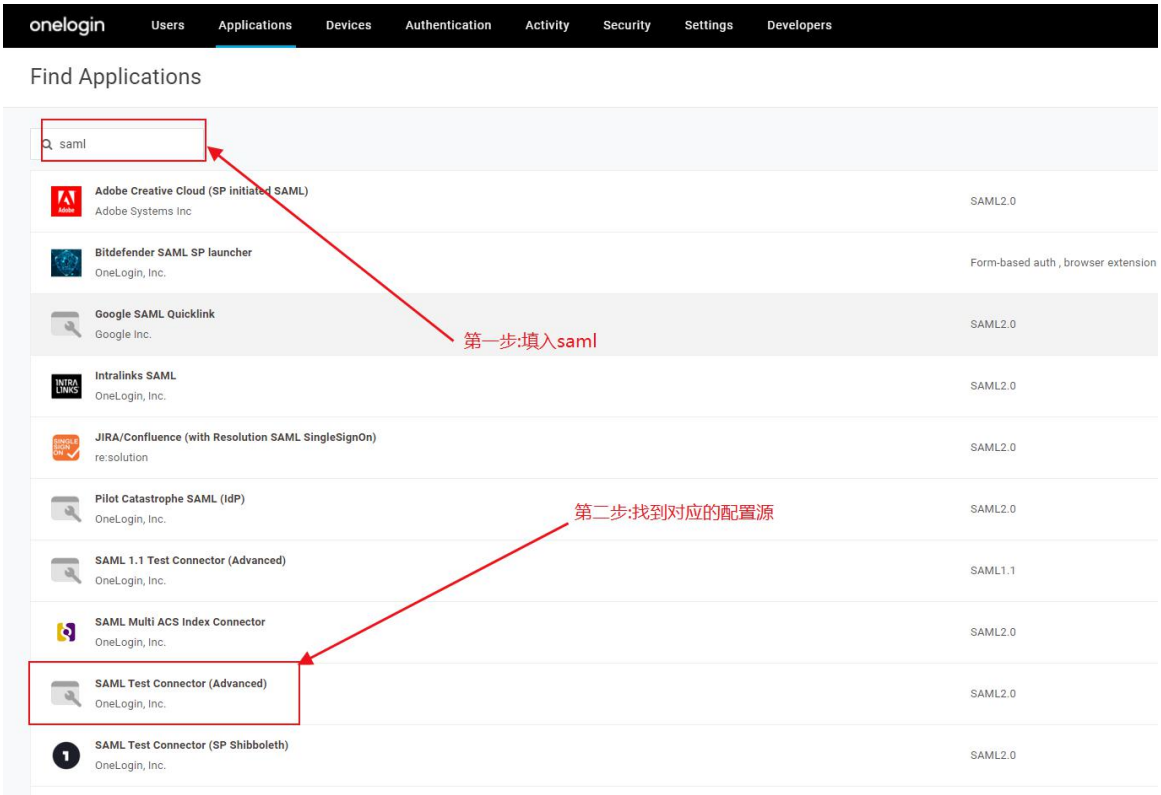
保存退出，source /etc/profile 加载环境变量，重启 Redash 服务，即配置好了发件服务。虚拟机环境需要重启服务器才能重新加载环境变量。

附件 6：Redash 中文版配置 Saml 单点登录

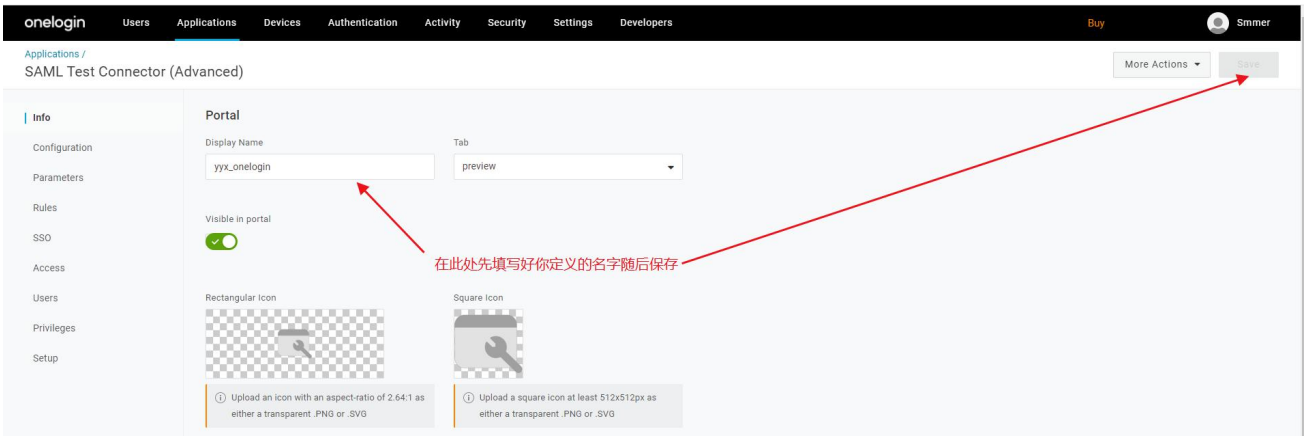
1.我们使用 onelogin 的统一认证服务器进行测试需要准备一个 onelogin 的账号
2.登录之后点击 Applications 下的 Add APP 进行添加



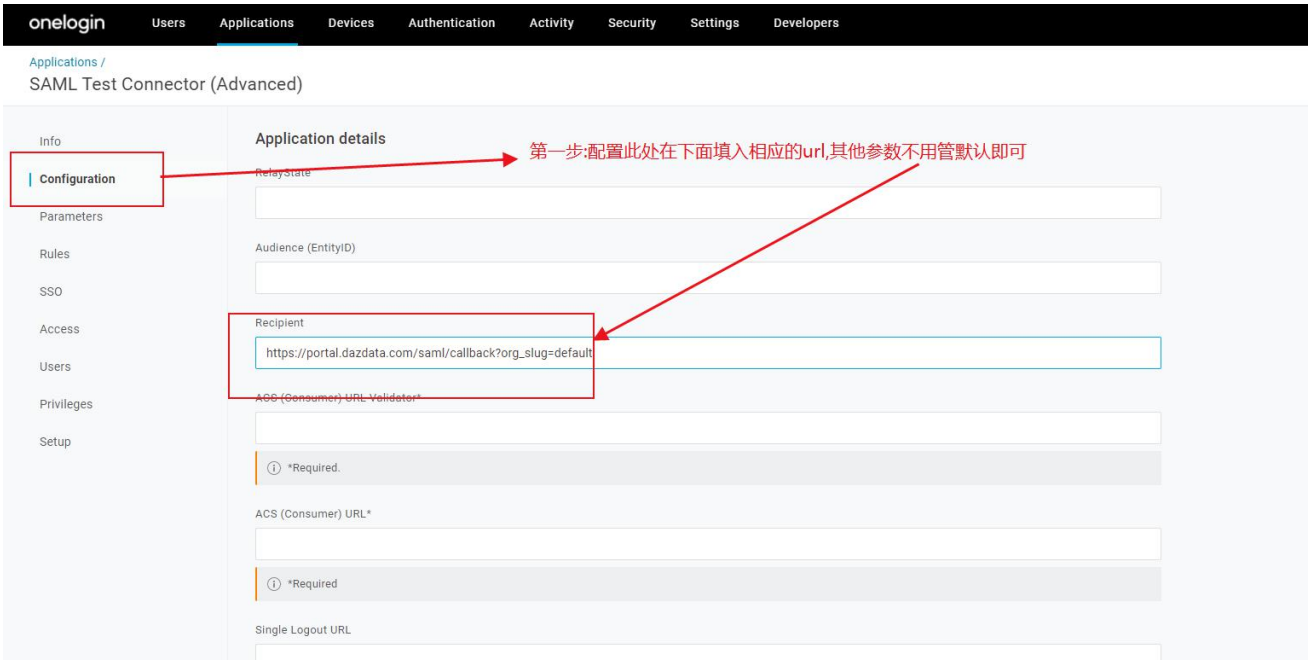
3.在搜索框内输入 saml 找到对应的配置源:



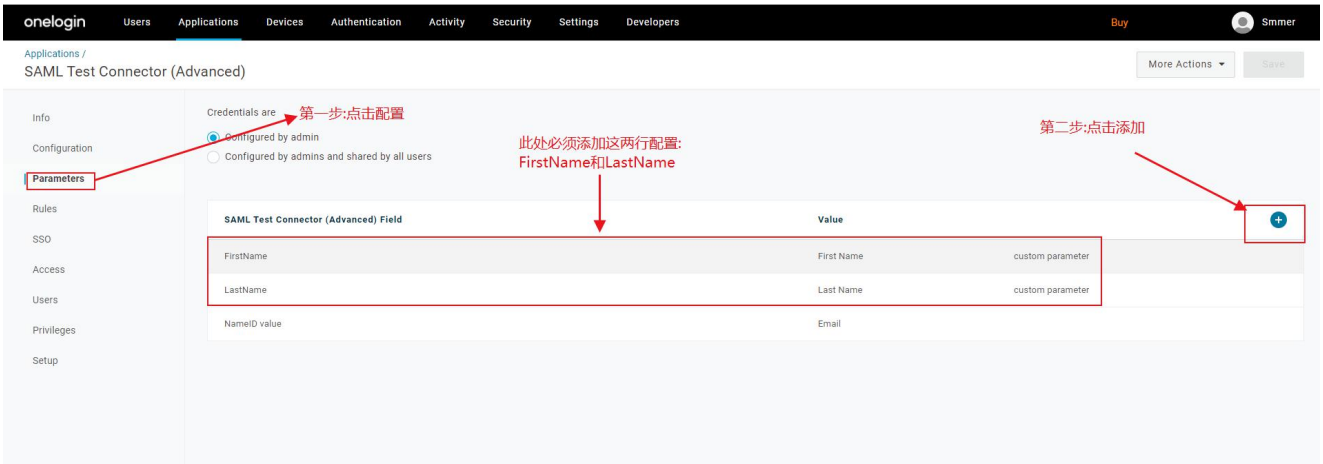
给源配置名称



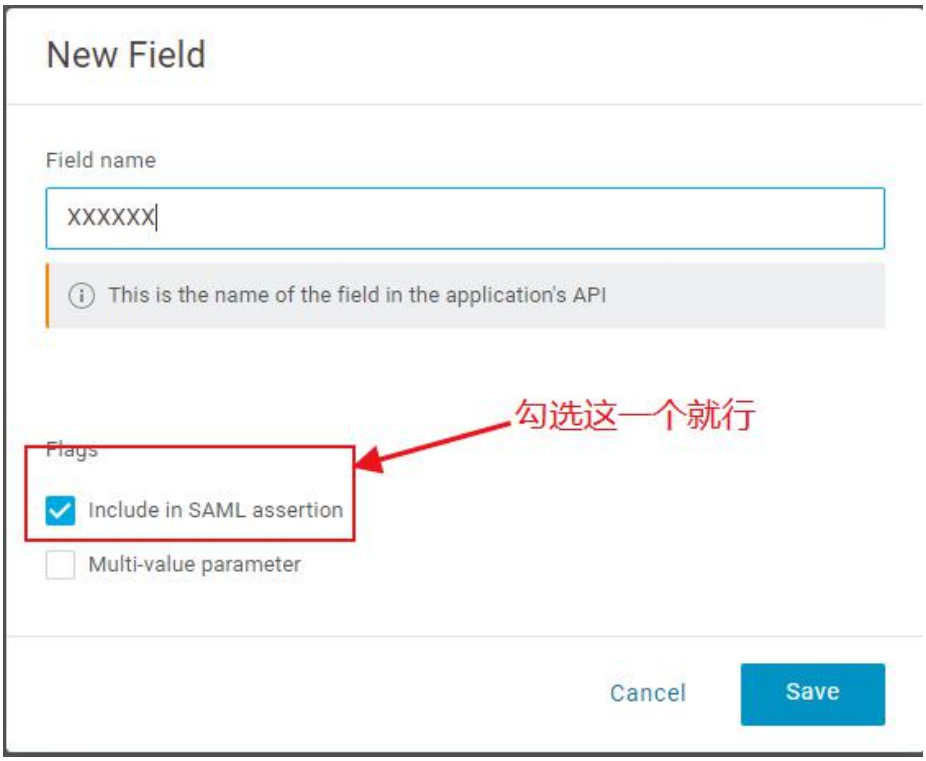
接下来配置 **Configuration** 里面的选项:



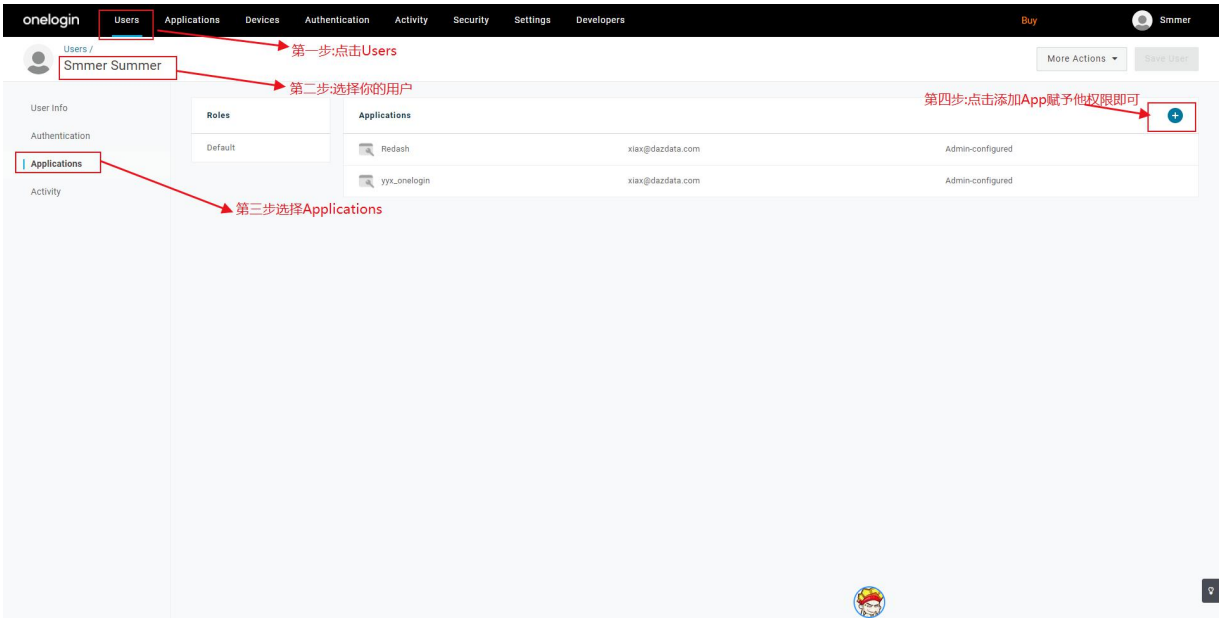
然后配置 **Parameters** 里面的参数:



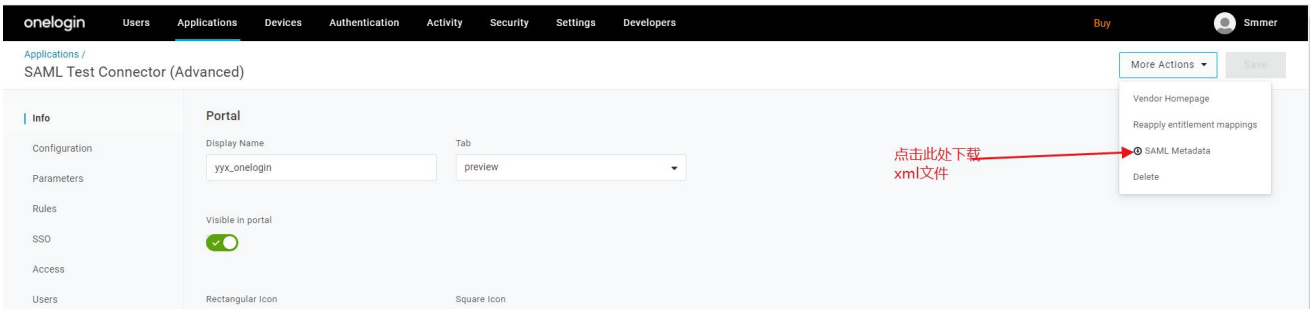
特别注意:

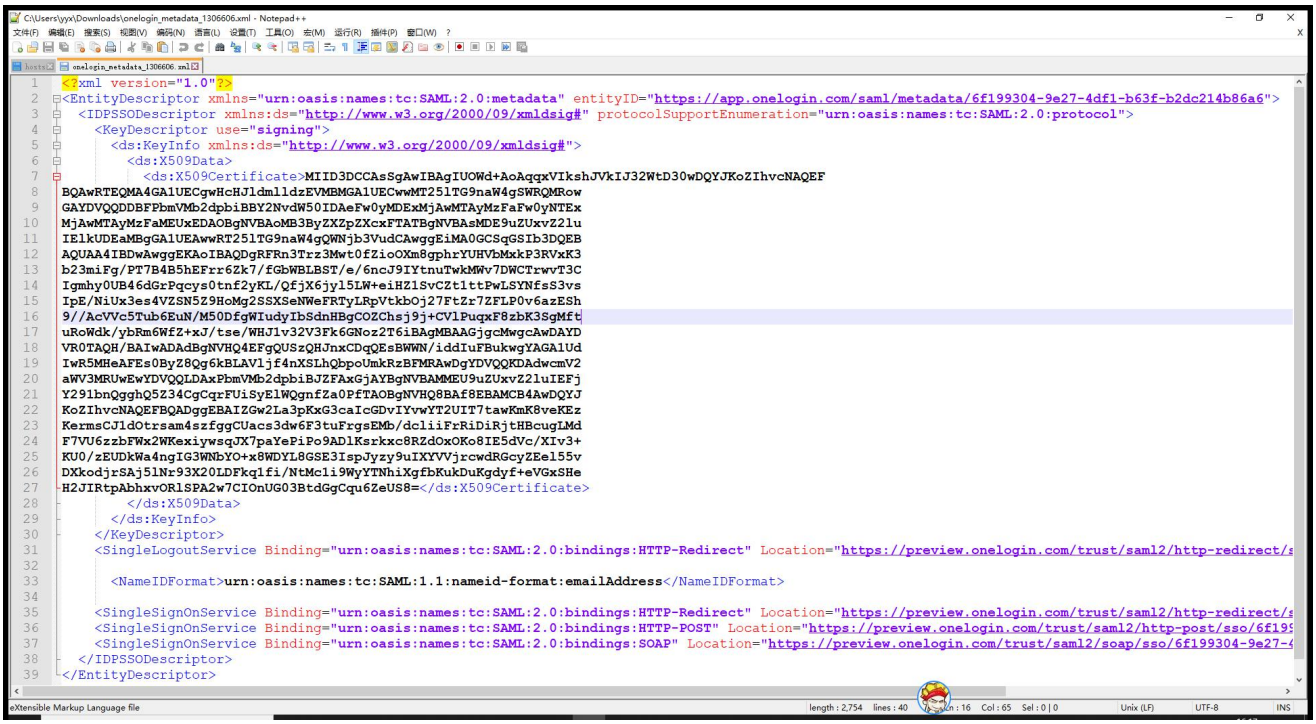


4.进入 Users 把你配置的 App 添加进来赋予权限

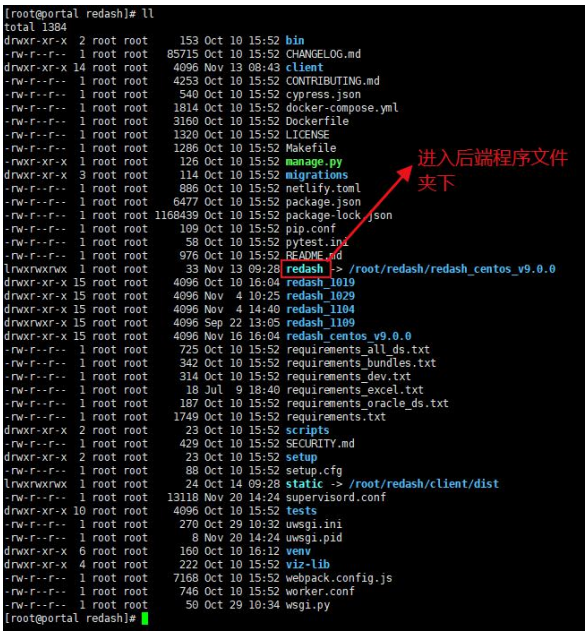


5.所有参数配置完成之后下载相对应的 xml 文件可以打开看看:





6.进入 redash 后端程序文件夹下



```
[root@portal redash]# ll
total 48
-rw-r--r-- 1 root root 1686 Aug 19 02:16 app.py
drwxr-xr-x 3 root root  195 Nov 20 14:09 authentication
drwxr-xr-x 3 root root  173 Nov 13 14:44 cli
drwxr-xr-x 3 root root  195 Nov 13 14:44 destinations
-rw-r--r-- 1 root root 3240 Aug 19 02:16 extensions.py
drwxr-xr-x 3 root root 4096 Nov 13 14:46 handlers
-rw-r--r-- 1 root root 1595 Nov 13 14:17 __init__.py
drwxr-xr-x 3 root root   81 Nov 13 14:44 metrics
drwxr-xr-x 3 root root  198 Nov 13 14:46 models
-rw-r--r-- 1 root root 3636 Aug 19 02:16 monitor.py
-rw-r--r-- 1 root root 3365 Nov 16 15:53 permissions.py
drwxr-xr-x 2 root root 4096 Nov 16 15:54 __pycache__
drwxr-xr-x 4 root root 4096 Nov 13 14:46 query_runner
-rw-r--r-- 1 root root 2646 Sep 22 21:05 security.py
drwxr-xr-x 3 root root   67 Nov 13 14:44 serializers
drwxr-xr-x 3 root root  112 Nov 20 14:23 settings
drwxr-xr-x 4 root root  176 Nov 13 14:44 tasks
drwxr-xr-x 5 root root  259 Nov 13 14:44 templates
drwxr-xr-x 3 root root  151 Nov 13 14:46 utils
-rw-r--r-- 1 root root 3130 Aug 19 02:16 version_check.py
-rw-r--r-- 1 root root 1420 Aug 19 02:16 worker.py
-rw-r--r-- 1 root root   50 Aug 19 02:16 wsgi.py
[root@portal redash]#
```

进入这个目录下

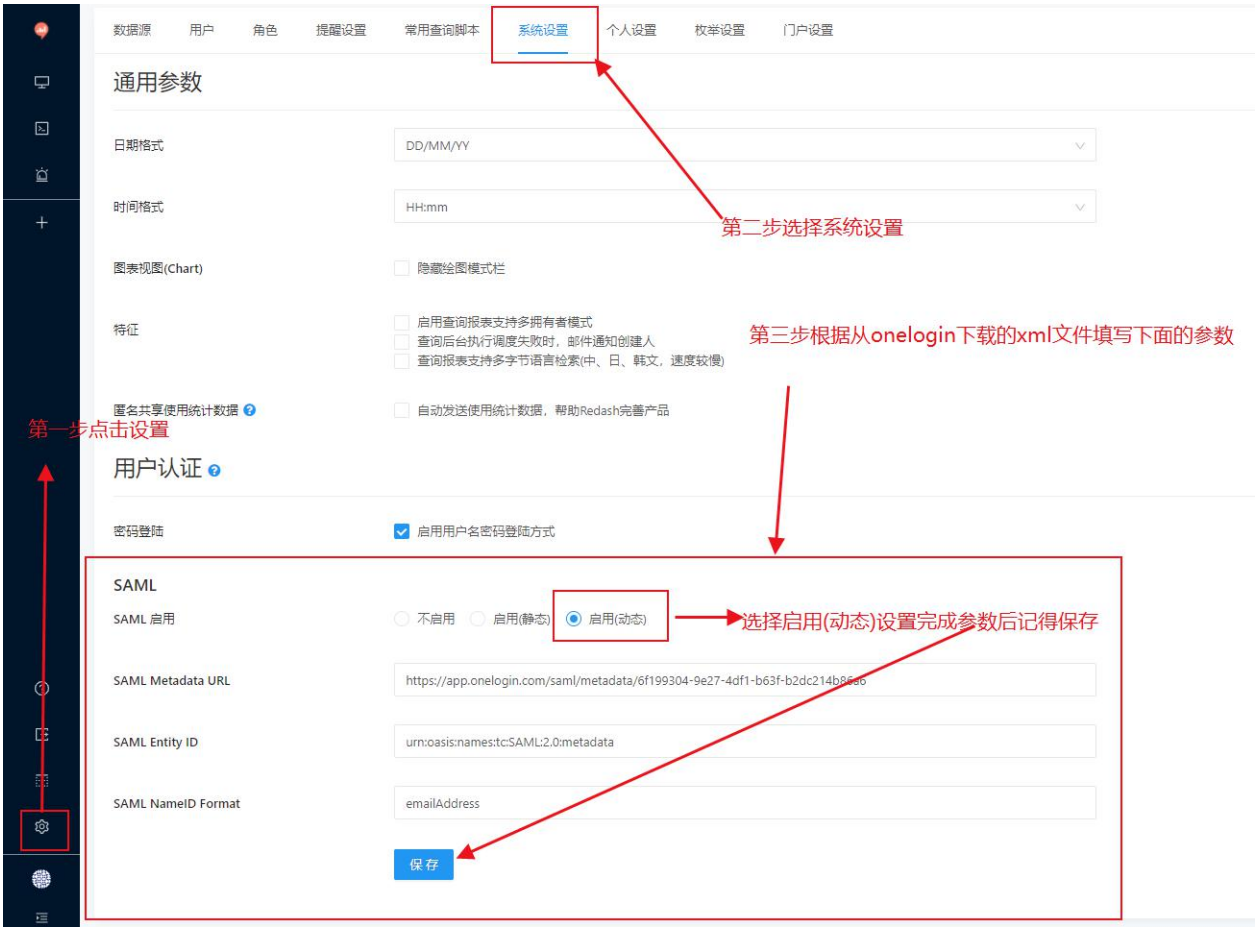
```
[root@portal redash]# cd authentication/
[root@portal authentication]# ll
total 48
-rw-r--r-- 1 root root 2544 Aug 19 02:16 account.py
-rw-r--r-- 1 root root 3916 Aug 19 02:16 google_oauth.py
-rw-r--r-- 1 root root 9506 Nov  6 00:14 __init__.py
-rw-r--r-- 1 root root 1976 Aug 19 02:16 jwt_auth.py
-rw-r--r-- 1 root root 2936 Aug 19 02:16 ldap_auth.py
-rw-r--r-- 1 root root  564 Aug 19 02:16 org_resolving.py
drwxr-xr-x 2 root root 4096 Nov 20 14:10 __pycache__
-rw-r--r-- 1 root root 1814 Aug 19 02:16 remote_user_auth.py
-rw-r--r-- 1 root root 6895 Nov 20 14:09 saml_auth.py
[root@portal authentication]# vim saml_auth.py
```

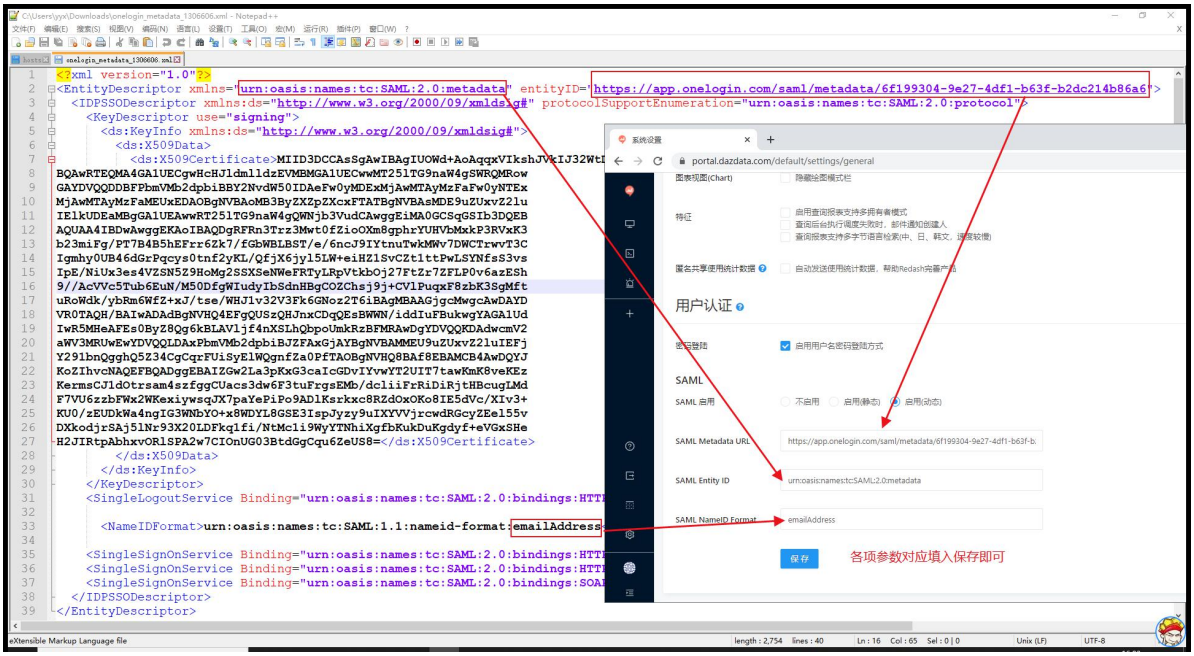
修改它

```
37              _external=True,
38              _scheme=settings.SAML_SCHEME_OVERRIDE,
39          )
40      else:
41          acs_url = url_for("saml_auth.idp_initiated", org_slug=org.slug, _external=True)
42
43      saml_settings = {
44          "metadata": {"remote": [{"url": metadata_url}]},
45          "service": {
46              "sp": {
47                  "endpoints": {
48                      "assertion_consumer_service": [
49                          (acs_url, BINDING_HTTP_REDIRECT),
50                          (acs_url, BINDING_HTTP_POST),
51                      ]
52                  },
53                  # Don't verify that the incoming requests originate from us via
54                  # the built-in cache for authn request ids in pysaml2
55                  "allow_unsolicited": True,
56                  # Don't sign authn requests, since signed requests only make
57                  # sense in a situation where you control both the SP and IdP
58                  "authn_requests_signed": False,
59                  "logout_requests_signed": True,
60                  "want_assertions_signed": False,
61                  "want_response_signed": False,
62              }
63          },
64      }
65
66      if settings.SAML_ENCRYPTION_ENABLED:
67          encryption_dict = {
68              "xmlsec_binary": get_xmlsec_binary(),
69              "encryption_keypairs": [
70                  {
71                      "key_file": settings.SAML_ENCRYPTION_PEM_PATH,
72                      "cert_file": settings.SAML_ENCRYPTION_CERT_PATH,
73                  }
74              ],
75          }
76          saml_settings.update(encryption_dict)
77
78      if saml_type is not None and saml_type == "static":
79          metadata_inline = mustache_render(
80              inline_metadata_template,
```

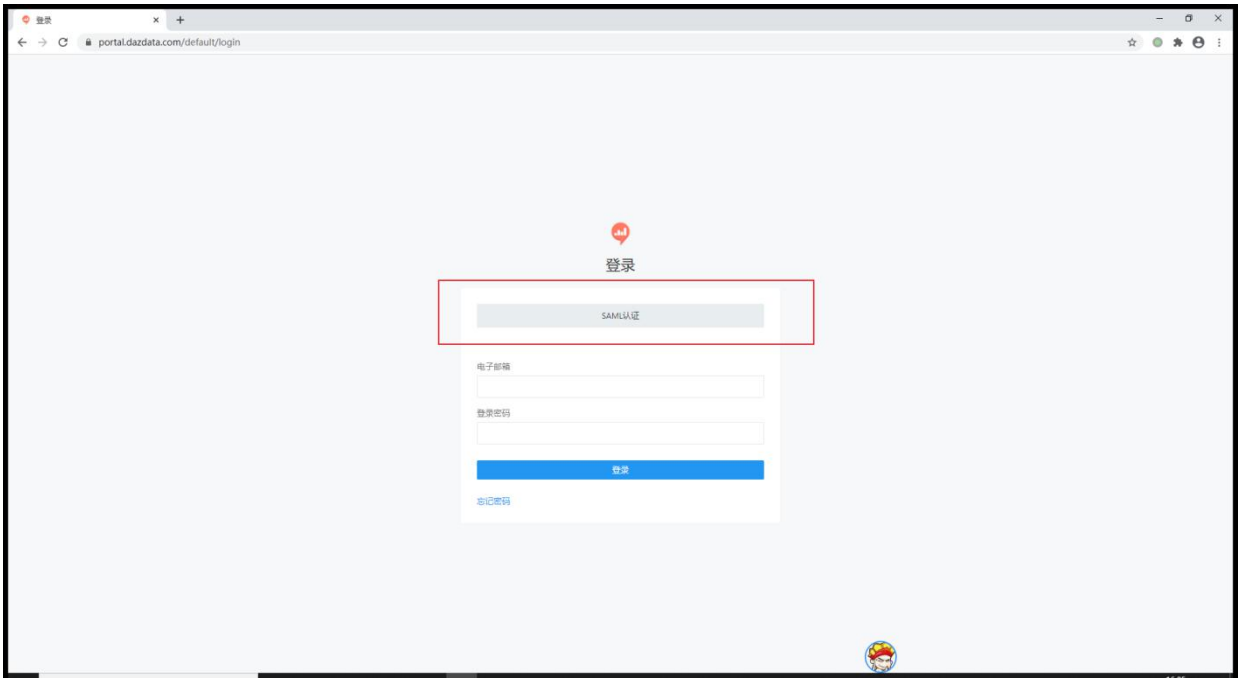
参数改为False即可

随后重启 **Redash** 的服务即可接下来进入 **redash** 在设置中配置 **saml** 的配置





接下来退出重新登录即可看到 **Saml** 单点登录了


登录
portal.dazdata.com/default/login
登录
SAML认证
电子邮箱
登录密码
登录
忘记密码