360开源wayne调研

[liangdong@smzdm.com](mailto:liangdong@smzdm.com)

[安装wayne 1](#_Toc532308470)

[搭建phpmyadmin 3](#_Toc532308471)

[配置与分析 5](#_Toc532308472)

[创建集群cluster 5](#_Toc532308473)

[创建部门namespace 6](#_Toc532308474)

[创建项目app 7](#_Toc532308475)

[部署deployment 8](#_Toc532308476)

[研究deployment功能 16](#_Toc532308477)

[1， deployment yaml动态生成 16](#_Toc532308478)

[Deployment发布 18](#_Toc532308479)

[Deployment发布状态获取 23](#_Toc532308480)

[Deployment pod列表获取 27](#_Toc532308481)

[web ssh访问pod内容器 31](#_Toc532308482)

[获取pod日志 33](#_Toc532308483)

[RBAC设计 35](#_Toc532308484)

调研一下360的k8s开源管理平台：<https://github.com/Qihoo360/wayne>

# 安装wayne

准备：一台ubuntu虚拟机。

更新apt：

apt-get update

安装mysql：

sudo apt-get install mysql-server

密码设置为root

修改mysql监听IP：

vim /etc/mysql/mysql.conf.d/mysqld.cnf

修改：

bind-address = 0.0.0.0

重启mysql：

systemctl restart mysql

给mysql root帐号授权：

mysql -u root -proot

grant all privileges on \*.\* to 'root'@'%' identified by 'root';

flush privileges;

现在可以通过外部IP访问到mysql了：

mysql -h172.18.11.25 -u root -proot

安装docker：

sudo apt-get install docker.io

安装docker-compose：

apt install docker-compose

下载golang并解压：

https://golang.org/dl/

移动目录：

sudo mv go/ /usr/local/

软链接go程序：

sudo ln -s /usr/local/go/bin/go /usr/local/bin

准备gopath目录：

mkdir gopath

cd gopath/

导出GOPATH环境变量：

export GOPATH=`pwd`

下载wayne（时间比较长，等一下就行）：

go get github.com/Qihoo360/wayne

进入src/github.com/Qihoo360/wayne，然后vim src/backend/conf/app.conf，修改DBTns = "tcp(127.0.0.1:3306)"为：

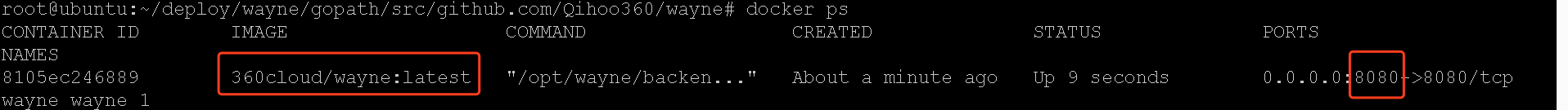
DBTns = "tcp(172.18.11.25:3306)"，意思就是让容器里的wayne访问宿主机上的mysql。

创建一个没有任何卵用，但是如果不存在就会启动失败的文件：  
touch src/backend/conf/dev.conf

现在通过docker-compose拉起wayne：

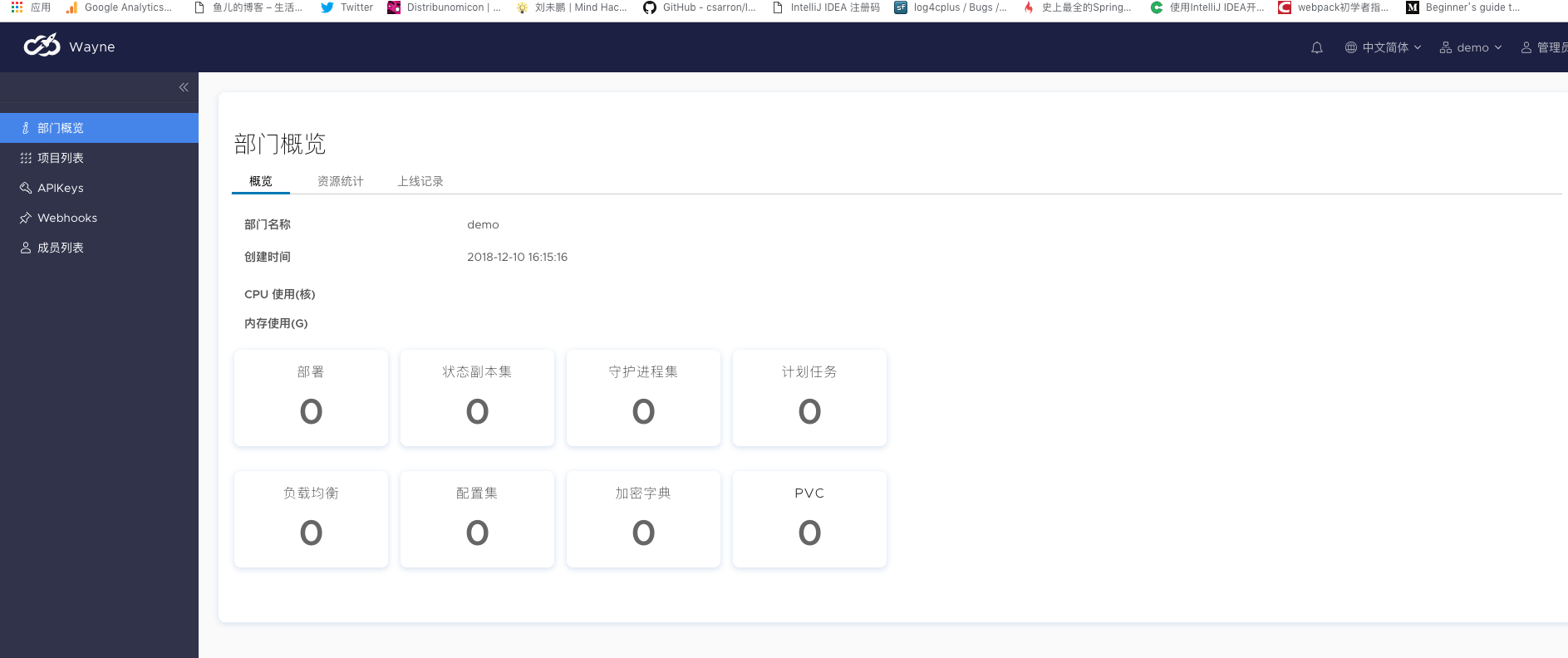
docker-compose up -d wayne

看一下容器docker ps：



现在用宿主机IP访问8080就可以访问到容器中的wayne了：

<http://172.18.11.25:8080/>，帐号密码admin admin



右上角改成中文即可。

# 搭建phpmyadmin

研究wayne比较好的方法就是看它数据库设计，所以启动一个phpmyadmin吧：

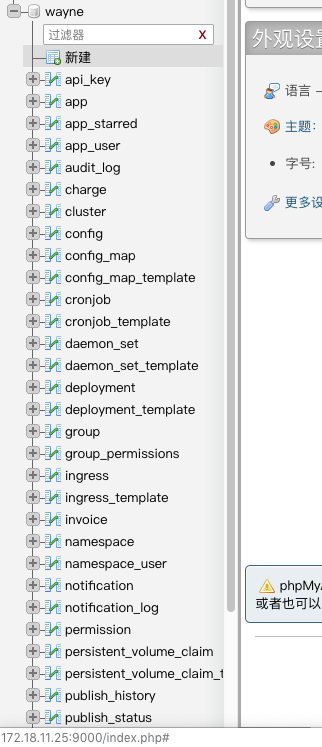
直接docker搞：

docker run --name myadmin -d -e PMA\_ARBITRARY=1 -p 9000:80 phpmyadmin/phpmyadmin

打开浏览器即可：



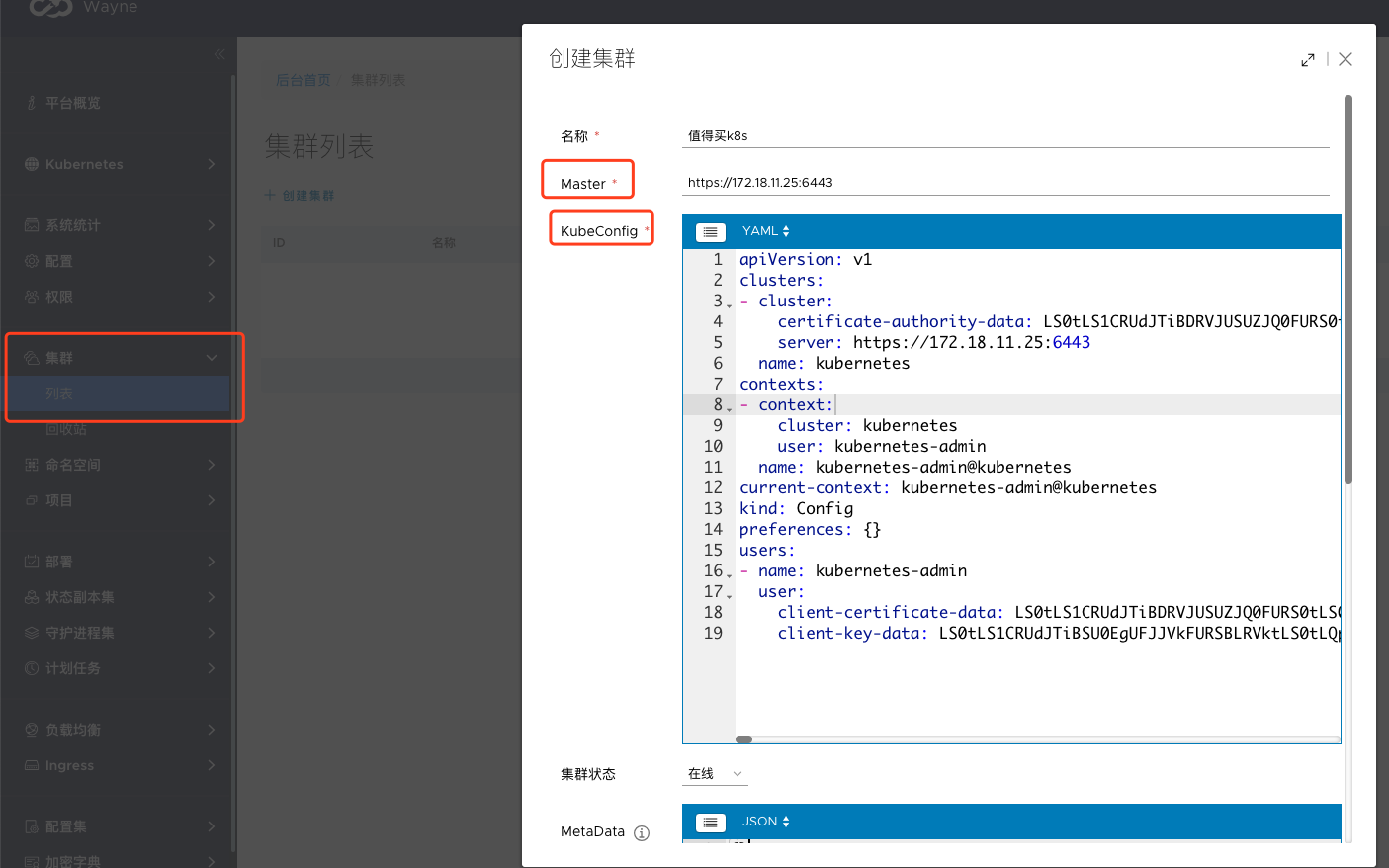
能看到所有wayne的表，这太重要了：



# 配置与分析

## 创建集群cluster

第一步就是把k8s集群的配置上传一下，这样wayne才能访问到k8s的apiserver：



配置方法很简单，在k8s master执行：

cat /etc/kubernetes/admin.conf

把内容粘贴到kubeconfig框里，master地址就贴master的访问地址。

现在可以查看k8s nodes了：



配置的集群信息存在cluster表：



Meta\_data是wayne支持的其他配置项，比如可以给所有部门下的容器配置公共的环境变量等，具体参考：<https://github.com/Qihoo360/wayne/wiki/Wayne-admin-cluster> 。

## 创建部门namespace

首先我发现wayne自己又创造了一个命名空间的概念，<https://github.com/Qihoo360/wayne/wiki/Wayne-admin-namespace> ，其实是部门的意思，比如PHP部门，关键是它其实对应一个k8s namespace，也就是说一个部门的应用会部署在一个特定的k8s namespace下面，wayne就是这么简单的实现原理。



删掉它，我们建一个php-team部门：



仍旧把应用部到default里就好了，否则还得去k8s里先建namespace。

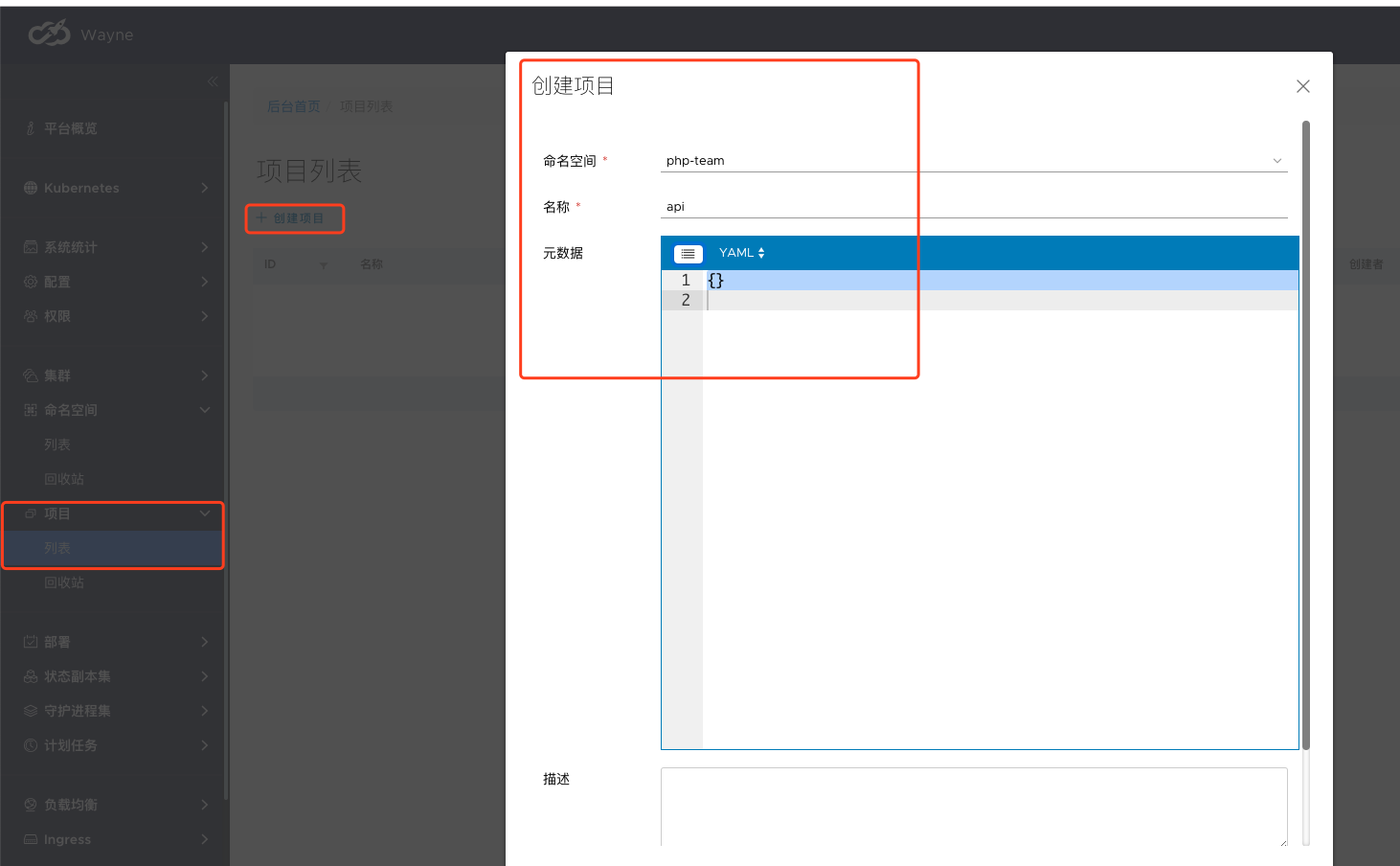
Wayne用json自定义保存了一些关于部门的配置信息：



## 创建项目app

部门下面，可以创建项目。

这里创建php-team部门下的api项目：



数据库：



## 部署deployment

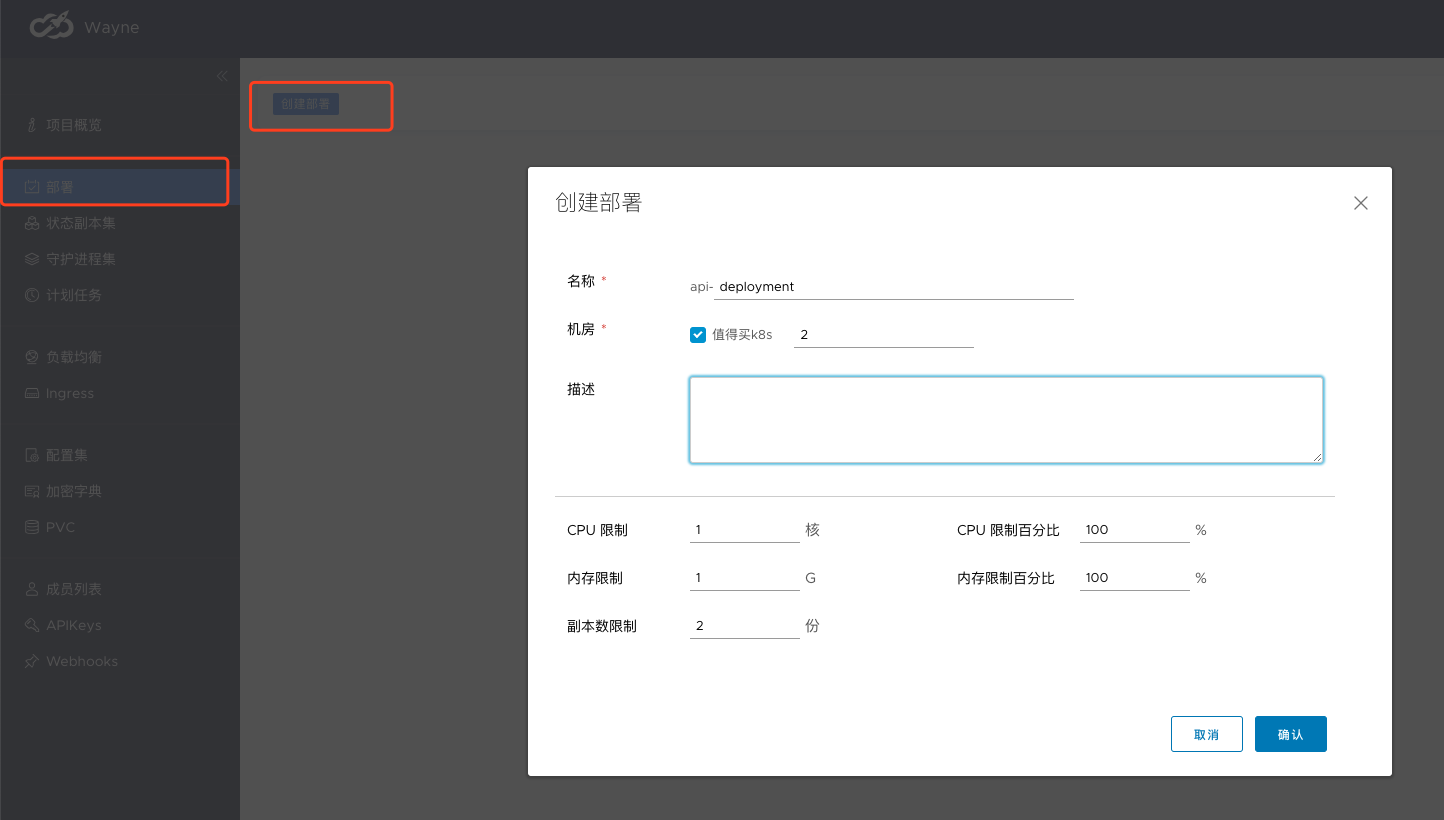
这一块是重中之重，先从wayne的操作流程，数据库结构这些表象，快速了解一下wayne部署deployment的功能。

然后再从源码，看一下每个功能的代码实现原理，因为wayne就是一个web应用，应该不会有什么复杂的东西在里面，主要就是对k8s client的操作和资源yaml的理解。

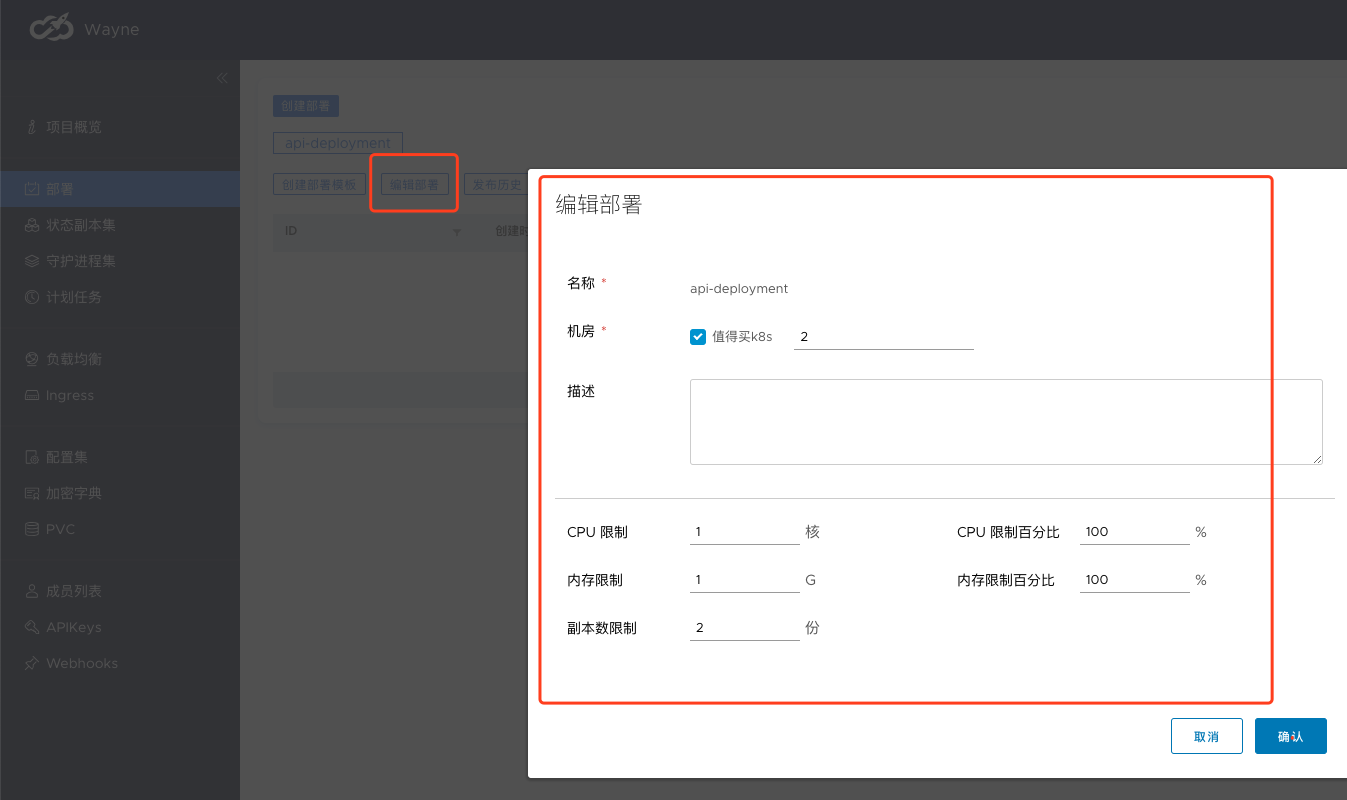
Wayne有自己的deployment概念，我们需要到wayne前台进行可视化配置：

点击api项目：

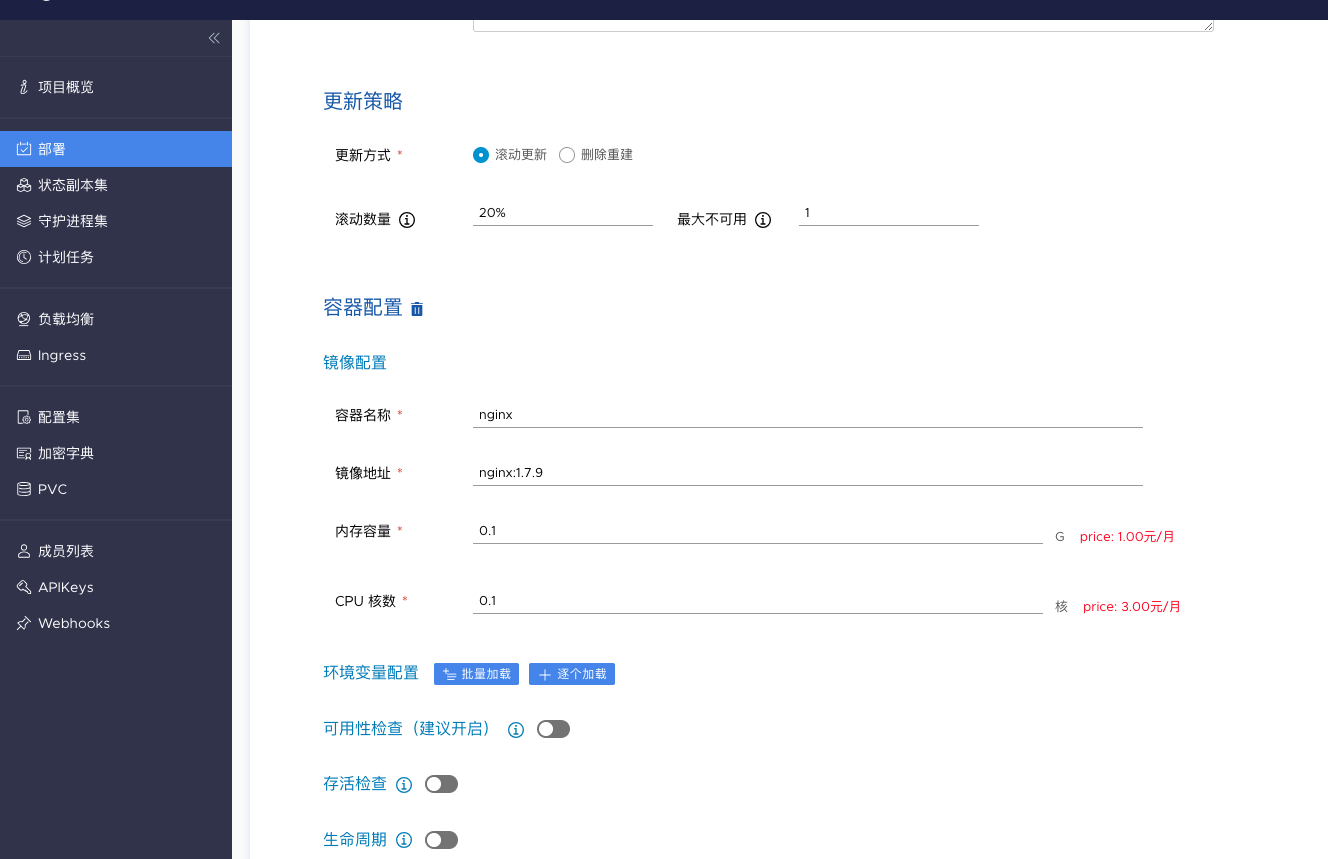


首先创建deployment，这里面的各种资源限制会影响到下面要创建的部署模板，wayne的设计还是很蹩脚的，继续往后看吧： 

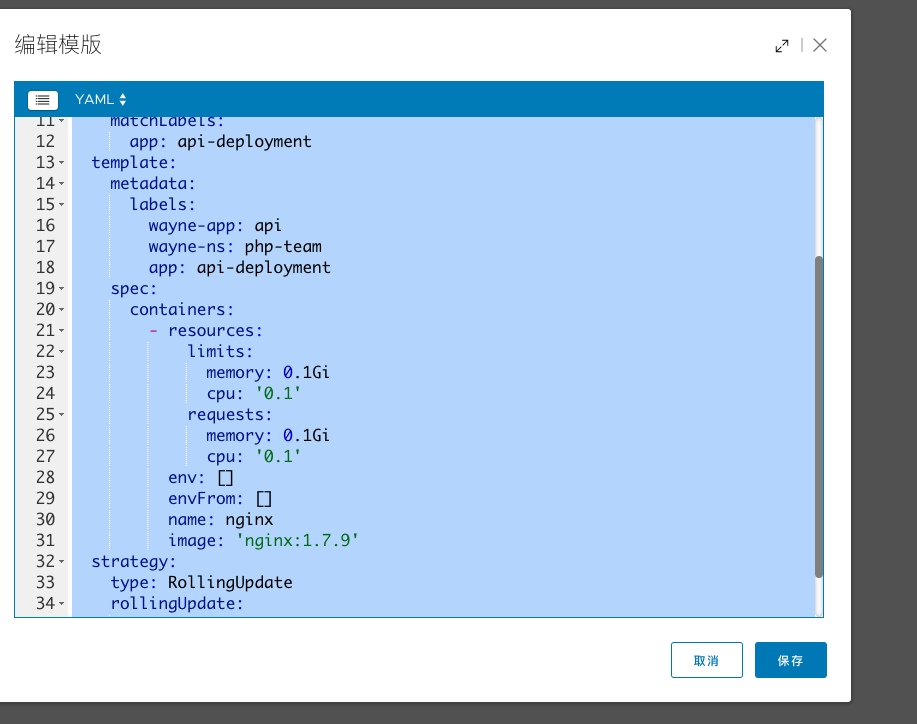
点击编辑部署，可以修改上述配置：



要真的发布k8s deployment，得在wayne部署下面创建wayne部署模板，点击”创建部署模板”：

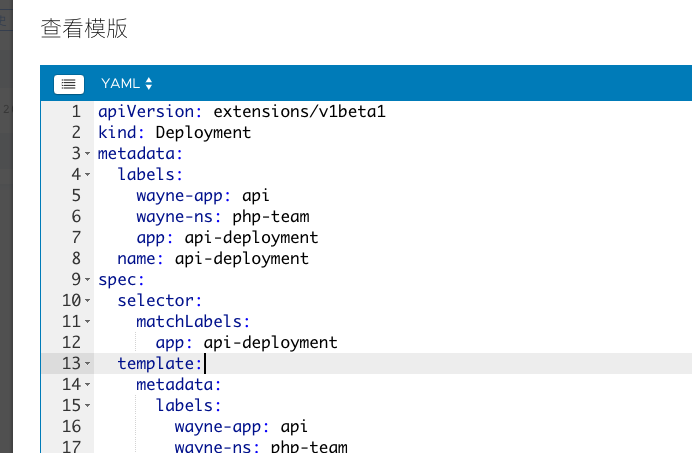


其实就是一个可视化编辑yaml的web界面，点高级配置可以直接编辑yaml：

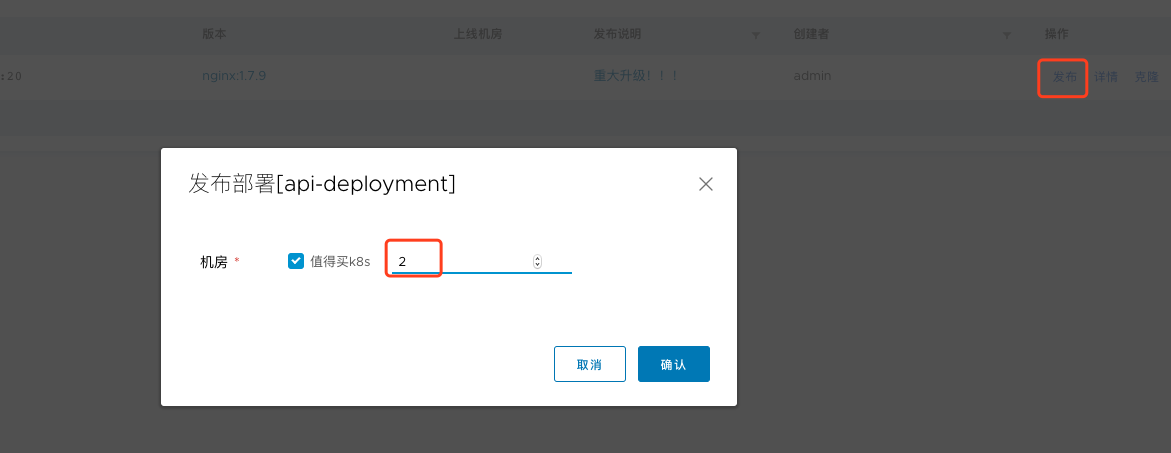


部署模板和部署的关系，就是部署会约束模板配置的最大内存，最大CPU，最大副本数量等等。

在部署模板里是没写replicas的，只有真正发布的时候才会拼接进去：



点发布，可以选replicas数量，但不能超过部署里的限制：

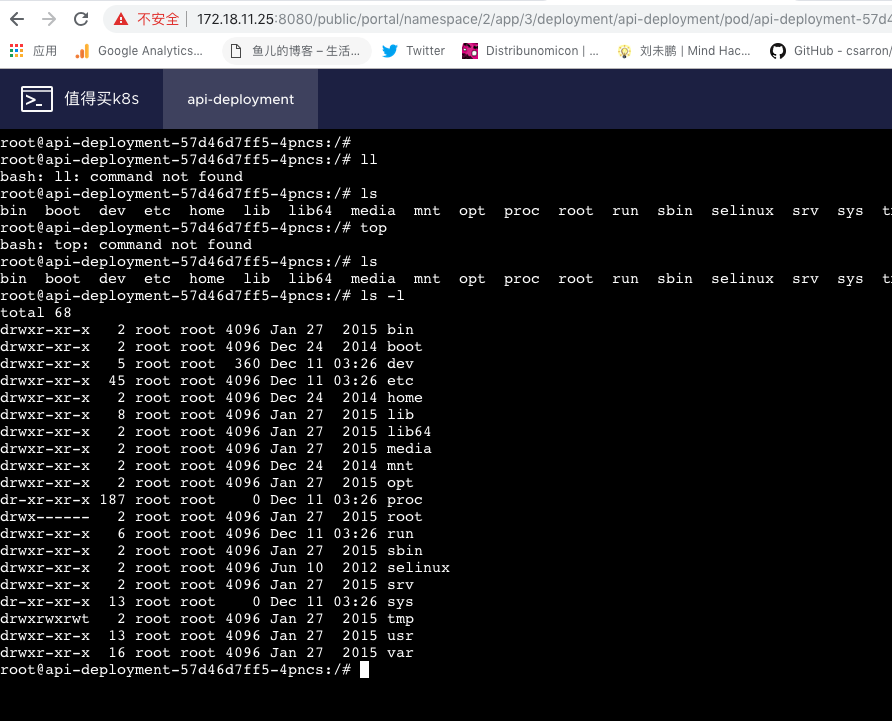


然后看到：

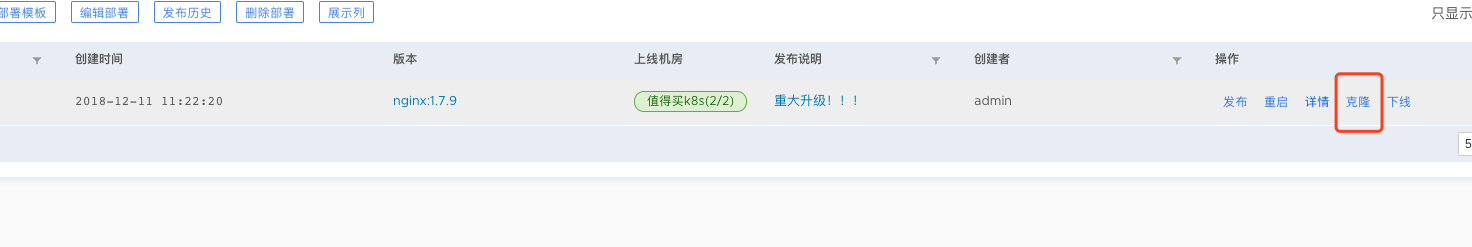
点击查看详情：



可以进入容器操作shell，以及查看容器日志，甚至直接杀死某个pod：



我们每次发版，就克隆一个新的部署模板：



改改镜像：



其他保持不变，提交！

现在，我们要发布api-deployment的新版本了：



上线机房信息已经转移到了新模板下面。

可惜1.8.0这个版本的nginx并不存在，deployment滚动升级停滞了，所以我们再把之前的模板发布一下：



就算回滚了，过会服务就恢复了正常：



对应数据库，deployment表存储wayne的部署配置：



Meta\_data以json格式保存了对deployment资源的基本约束信息。

部署模板在deployment\_template表，通过deployment\_id关联到deployment表：



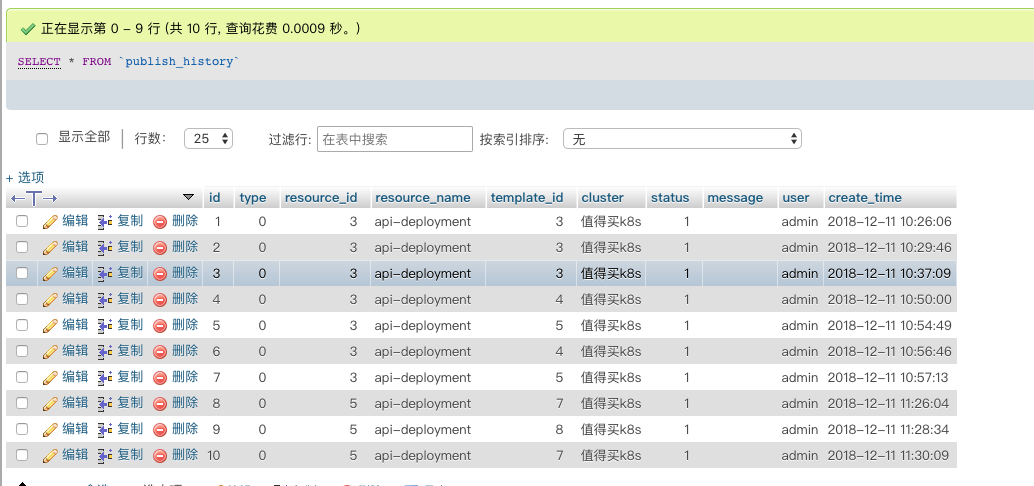
同一时刻只有一个template处于发布状态，所以wayne在publish\_status表里，记录了一下发布状态：



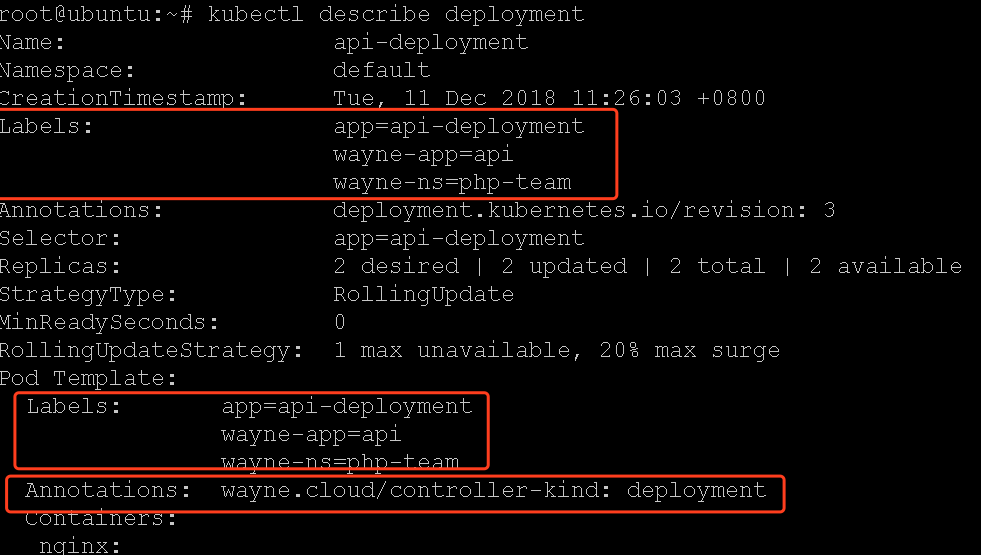
估计type==0就是指deployment资源，其他资源类型有statefulset、job等等。

Resource\_id是deployment的id，template\_id是deployment\_template的id，也就是说deployment 5的当前生效模板是7。

发布历史在Publish\_history表：



去K8s中看deployment的label/annotations也可以看出来，wayne并没有在deployment里面埋什么关联信息，纯粹是以wayne自身的mysql数据为依据：



就是些很基础的信息，感觉是没什么用的，并且wayne拼出来的YAML很简单，没有什么复杂特性。

# 研究deployment功能

**现在需要看wayne的deployment这块源码了，主要关注几个点：**

1. **Deployment Yaml动态生成**
2. **Deployment 发布**
3. **Deployment发布状态获取**
4. **Deployment POD列表获取**
5. **Web ssh访问POD内容器**
6. **获取POD日志**

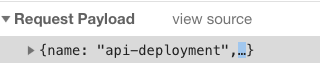
## deployment yaml动态生成

也就是在页面上点点点，然后生成yaml的过程。

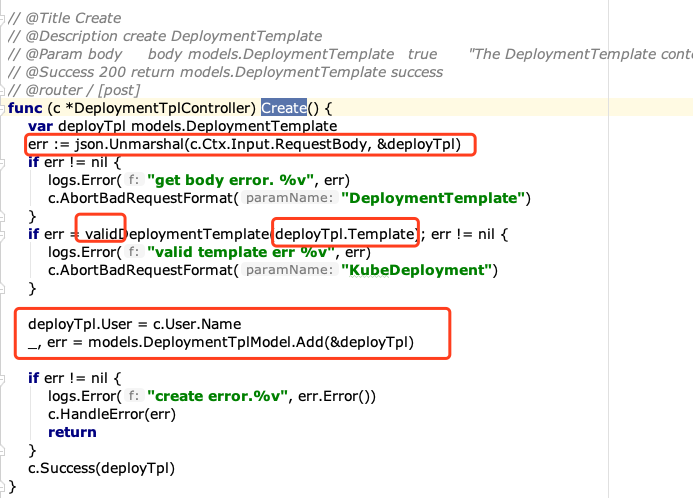
提交请求：



提交了一个JSON：



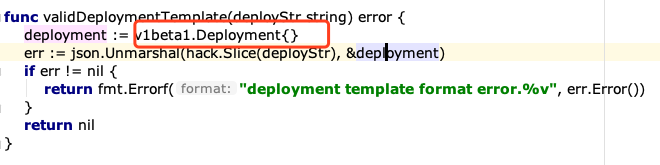
对应源码：/Users/liangdong/Documents/github/golang/src/github.com/Qihoo360/wayne/src/backend/controllers/deployment/deployment\_tpl.go



反序列化json，然后校验一下template字段是否合法：



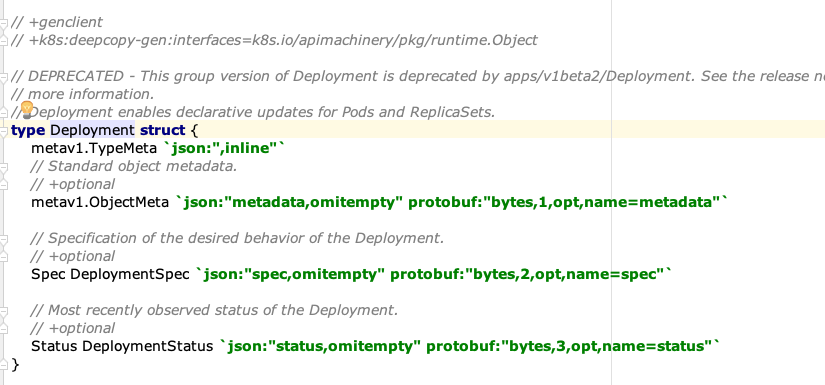
可见，前端直接拼好了deployment json上传了后端，后端简单用k8s deployment结构反序列化，没错就当作没错了：



然后就是插入到数据库把deployment\_tpl保存起来了。



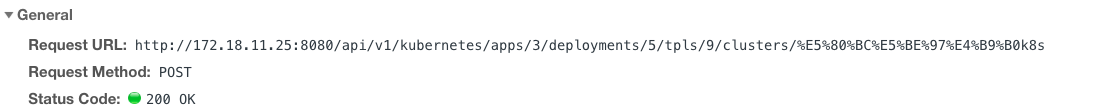
可见，平时我们人工编写都是yaml语法，但是json显然更适合程序处理，所以k8s定义的资源对象基本都是支持json和protobuf两种序列化格式（yaml的话需要先转json），并没有yaml：



关于yaml动态生成就是这样，前端搞json，后端可以结构化操作后序列化为Json。

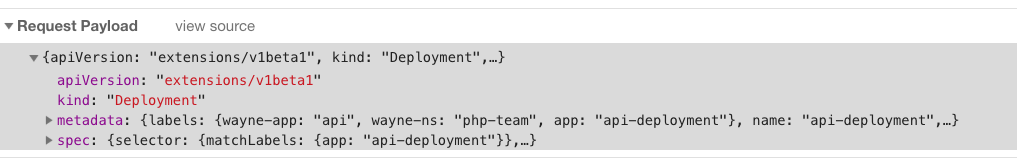
## Deployment发布

发布调用的接口：



哪个namespace的哪个app的哪个deployment的哪个模板，restful风格。

Body部分就是一个完整的deployment json：

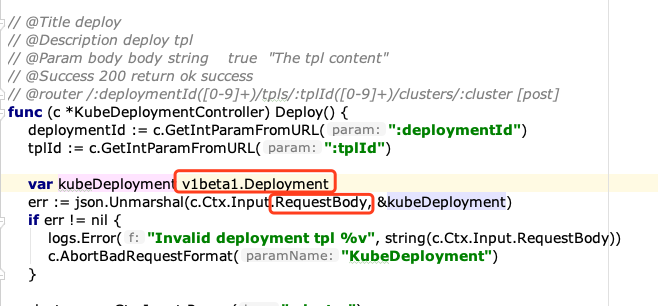


其中spec.replicas是前端拼接上去的，每次发布都可以选replicas数量：



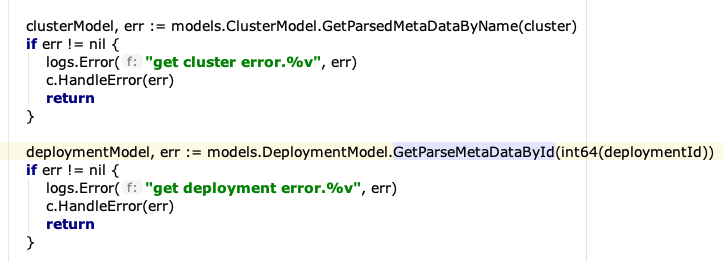
后端源码：

/Users/liangdong/Documents/github/golang/src/github.com/Qihoo360/wayne/src/backend/controllers/kubernetes/deployment/deployment.go



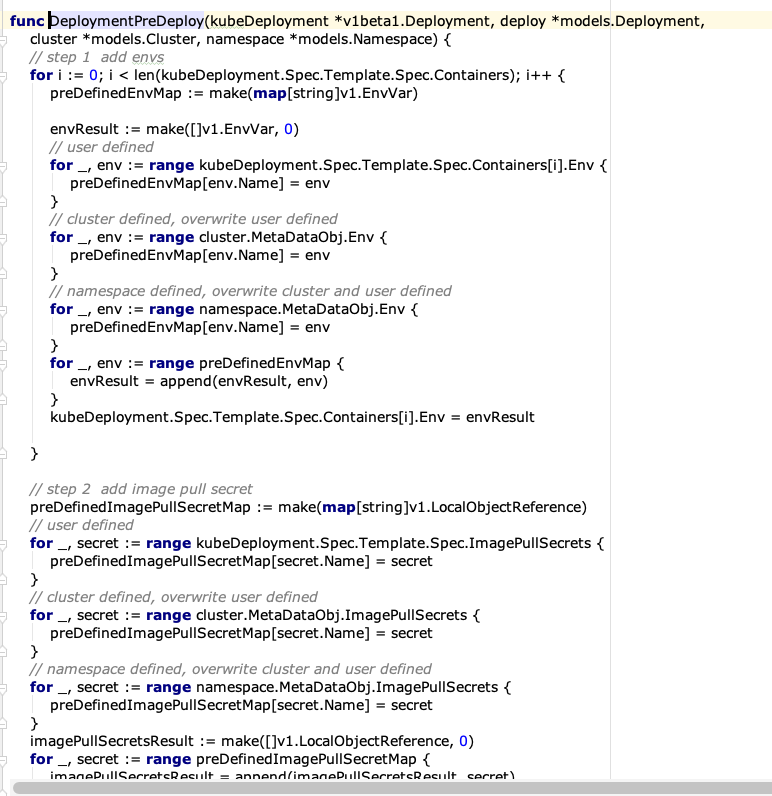
把deployment json反序列化一下。

取了一下数据库里的deployment表和cluster表：



然后对deployment json做一波处理：

实现如下：



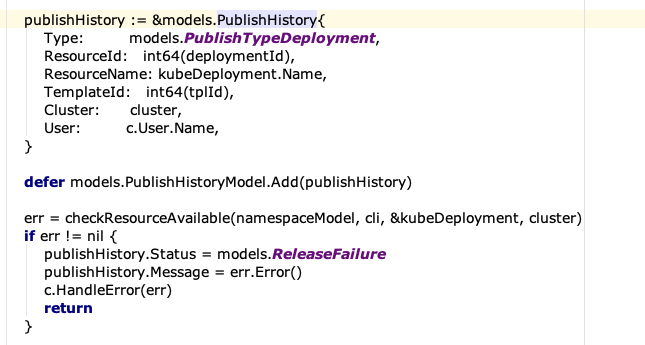
其中，deployment.spec.template.spec.containers就是遍历每个容器，把cluster表配置的默认环境变量merge到每个容器的环境变量里，最后再更新回deployment json里。

类似的还有其他一些基本的东西都拼到deployment json里，比如deployment属于哪个k8s namespace，还有给deployment的podTemplate打上一个annotations：

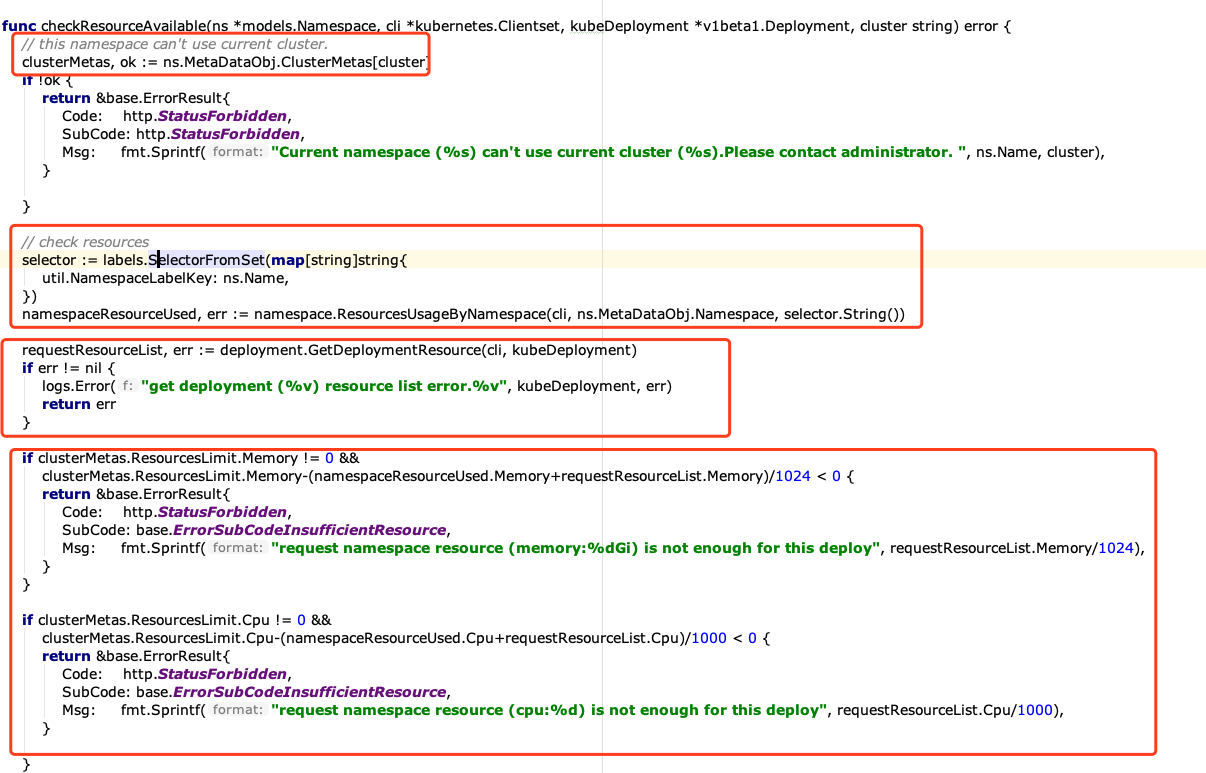


接下来，准备了一条发布历史记录，准备 最后插入。

自己检查了一下k8s集群剩余资源够不够：

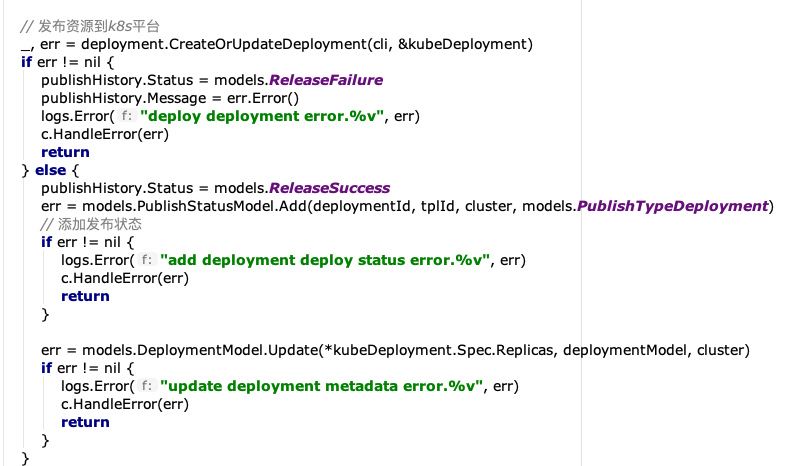


checkResourceAvalable是wayne自己做的资源充足检查逻辑，在配置wayne namespace的时候配置过一个metadata，限制了wayne可以在namespace下使用的资源总量。

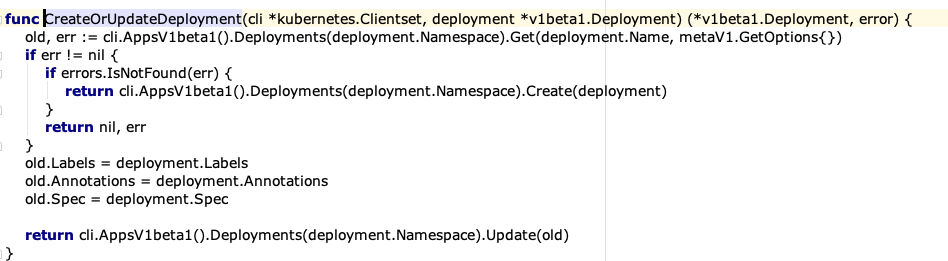


逻辑分4部分，先取cluster表的资源总限制，再取k8s中该namespace的已使用量，再计算要部署的deployment想比运行中的Deployment多占用了多少资源，最后算一下是否充足。

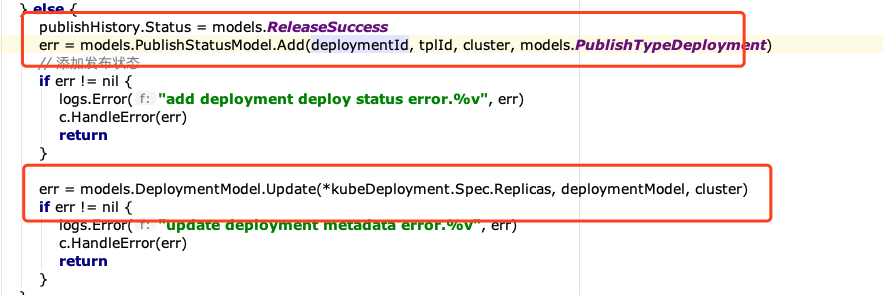
最后，就是发布环节了，把deployment json推给k8s：



其中CreateOrUpdateDeployment负责k8s推送：先查找deployment，不存在就create，已存在就update，就这样的一个逻辑：



推上去之后，就更新了publish\_status表：



也更新了一下deployment表，应该是在meta\_data中记录了本次发布的replicas是多少：



## Deployment发布状态获取

前端效果：



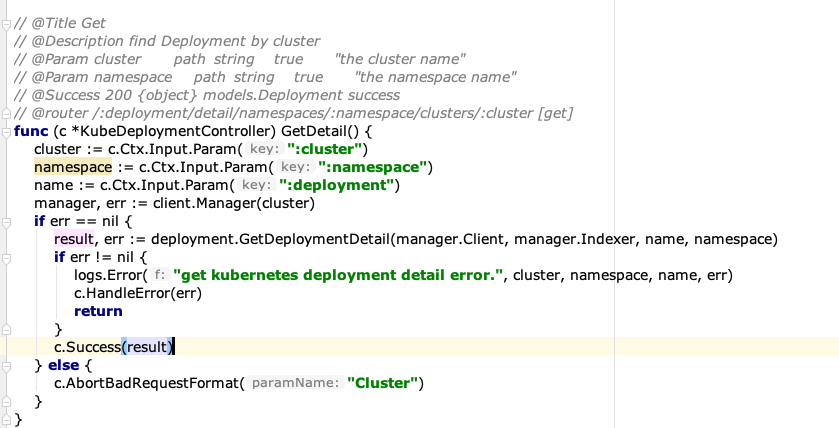
前端请求的接口是：



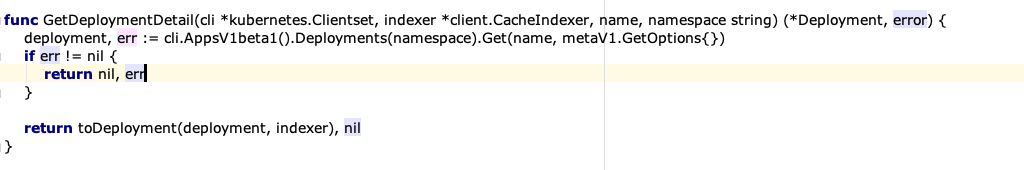
后端源码是：

/Users/liangdong/Documents/github/golang/src/github.com/Qihoo360/wayne/src/backend/controllers/kubernetes/deployment/deployment.go

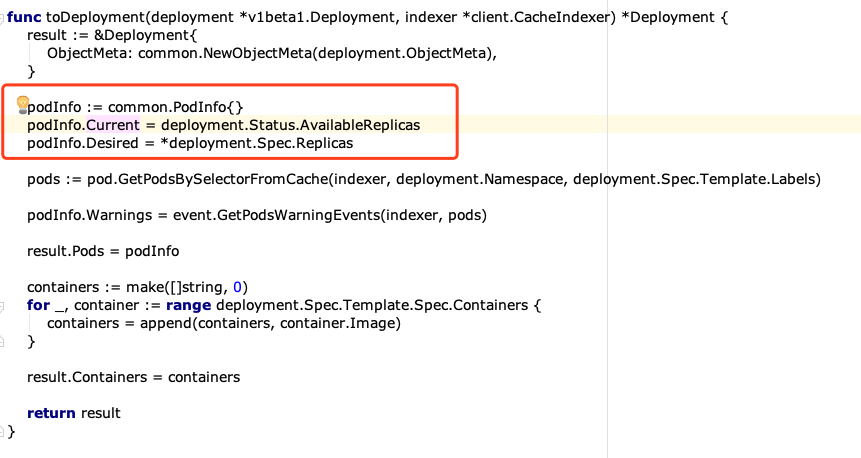
代码：



其中name变量就是api-deployment这个名字，详细看GetDeploymentDetail实现：

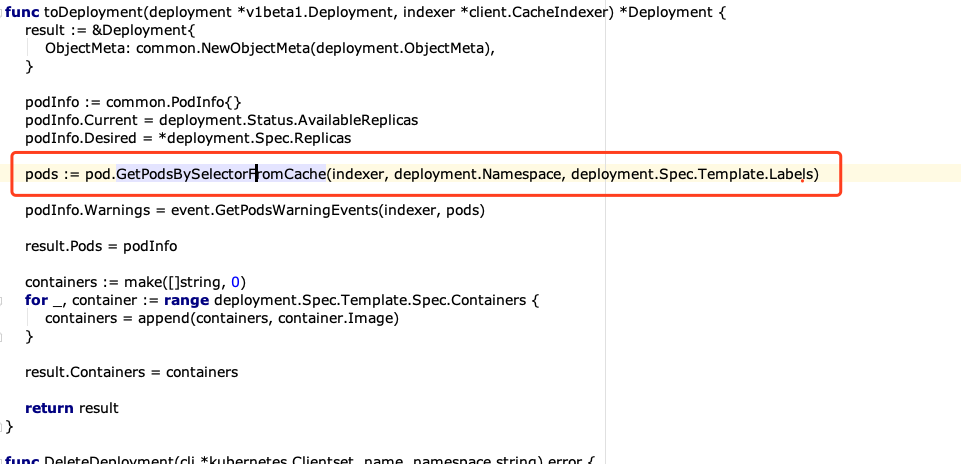


直接调了k8s api获取了deployment信息，toDeployment是啥呢？

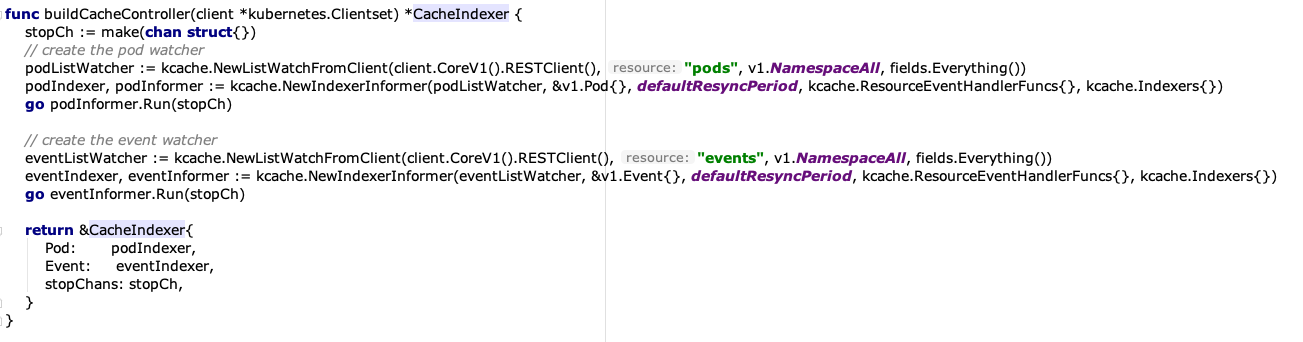


红框内，它就是取了一下deployment的replicas作为期望的pod数量，avaliableReplicas作为当前有效的pod数量，这个实现感觉有点粗矿。。。没有看到我想要的严谨判定。

这里获取deployment的pods列表，并不是调用的k8s api：

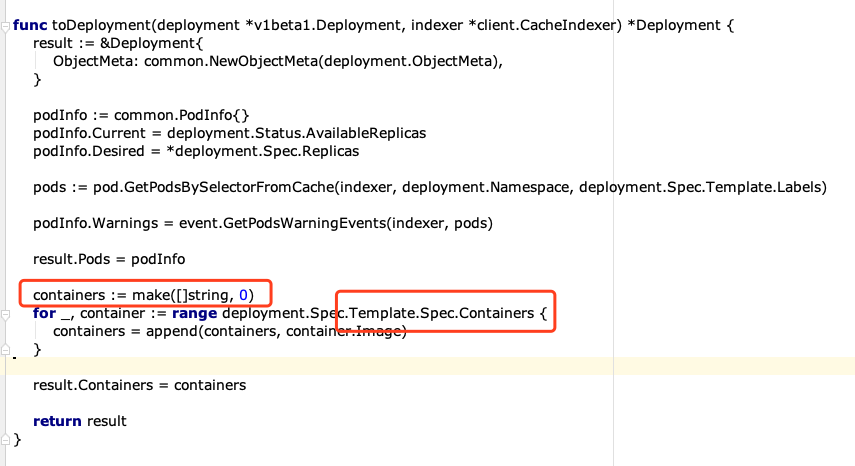


Wayne貌似是监听了k8s的所有event，在内存构建了一份pod列表数据，indexer是这样初始化来的：



为啥不直接像kubectl这样取apiserver请求一下呢？还不知道。

kubectl get pods -l app=api-deployment -o yaml



同样的，POD内容器的name都会收集起来。

接口响应值：

{

"data": {

"objectMeta": {

"name": "api-deployment",

"namespace": "default",

"labels": {

"app": "api-deployment",

"wayne-app": "api",

"wayne-ns": "php-team"

},

"annotations": {

"deployment.kubernetes.io/revision": "3"

},

"creationTimestamp": "2018-12-11T03:26:03Z"

},

"pods": {

"current": 1,

"desired": 1,

"running": 0,

"pending": 0,

"failed": 0,

"succeeded": 0

},

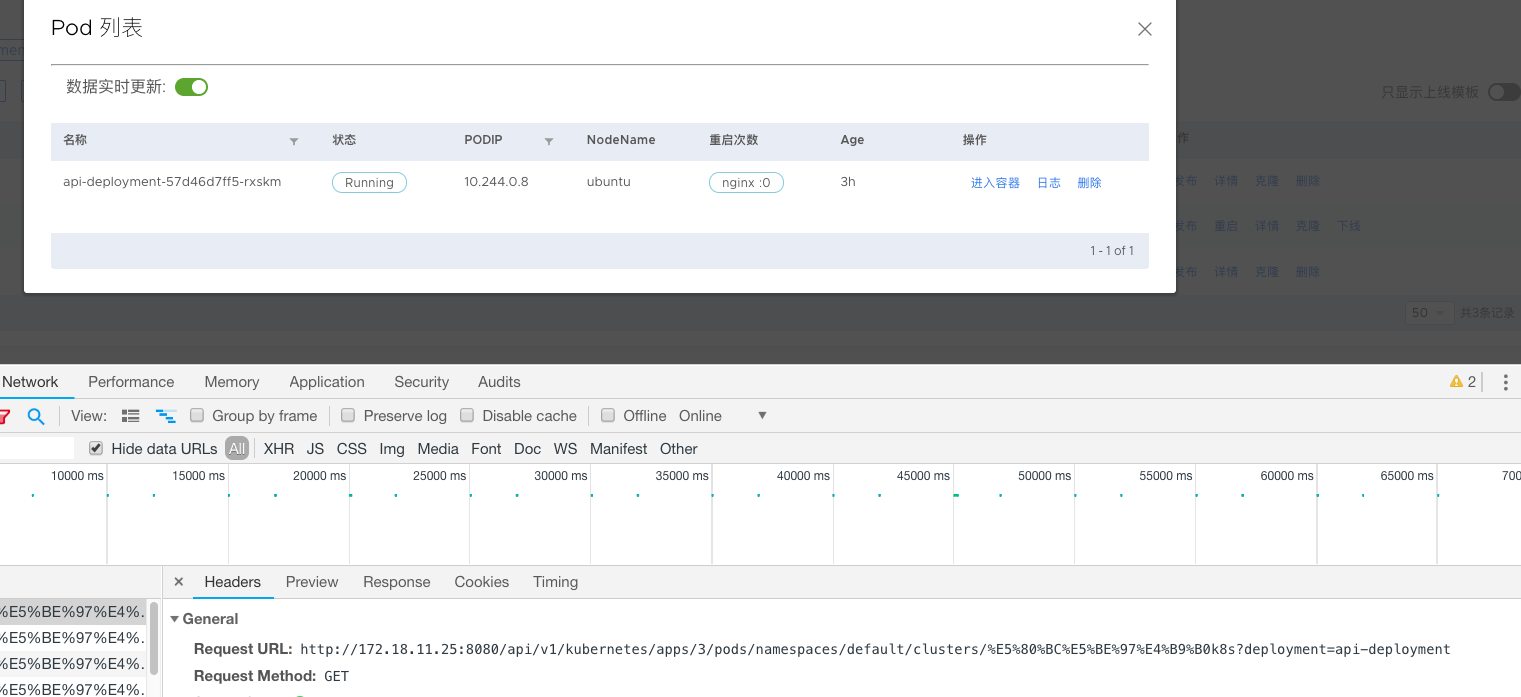
"containers": ["nginx:1.7.9"]

}

}

## Deployment pod列表获取

界面与接口如下：



对应源码：

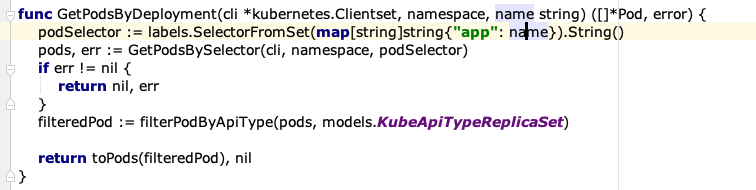
/Users/liangdong/Documents/github/golang/src/github.com/Qihoo360/wayne/src/backend/controllers/kubernetes/pod/pod.go

函数：

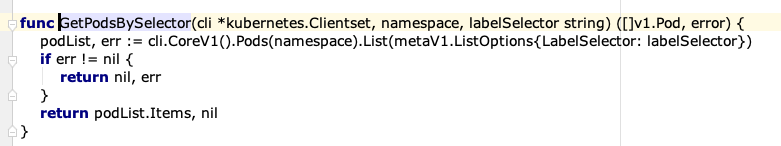


会进入第一个分支，调用GetPodsByDeployment：

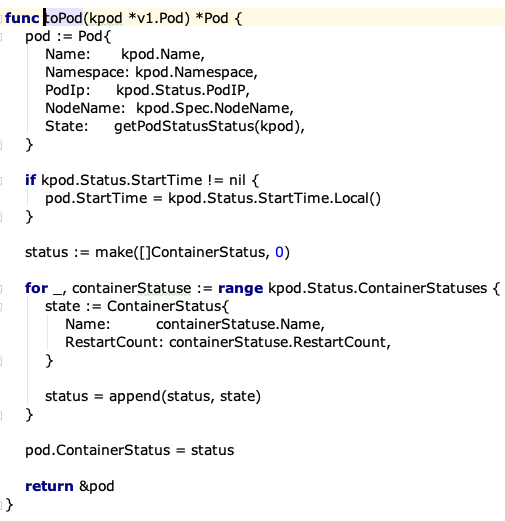
通过label筛选出POD，这里就是要筛选包含标签app: api-deployment的POD：



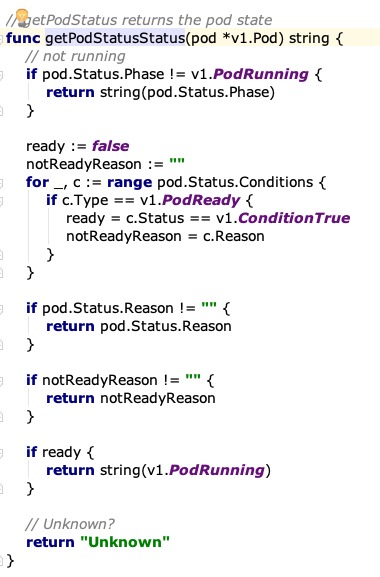
直接调用k8s api获取：

并且对pod的ownerReferences.kind字段做了一波过滤，只返回那些通过replicaset创建的pod，没啥用处。

最后调用的toPods转换成应答的结构：



取了POD基本信息，以及POD内每个container的重启次数，其中getPodStatusStatus计算POD的状态：



各种字段判定，恐怕也是经验积累下来的，各种试出来的吧，有点参考价值。

最后就是应答：

{

"data": [{

"name": "api-deployment-57d46d7ff5-rxskm",

"namespace": "default",

"containerStatus": [{

"name": "nginx",

"restartCount": 0

}],

"state": "Running",

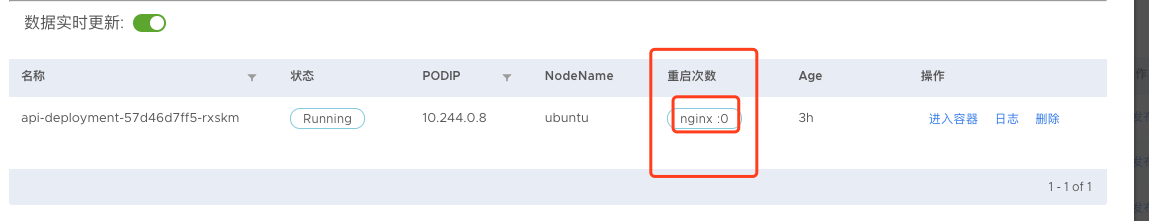
"podIp": "10.244.0.8",

"nodeName": "ubuntu",

"startTime": "2018-12-11T11:26:03+08:00"

}]

}

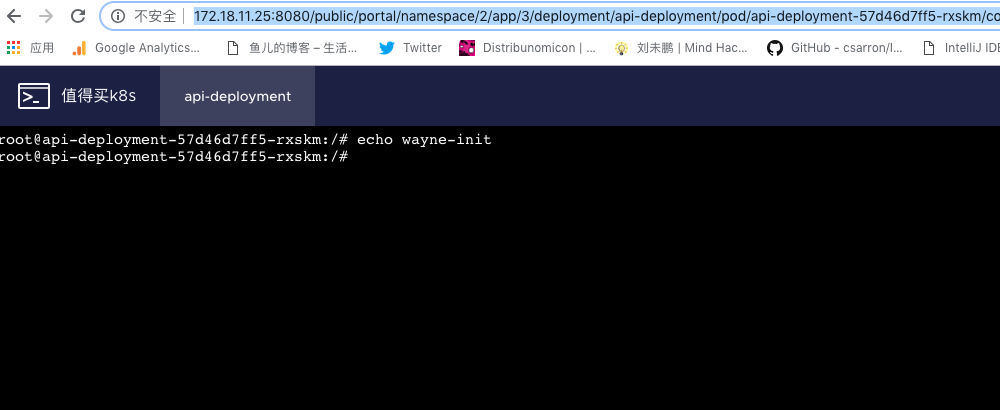
每个pod的状态、IP、节点名，以及POD内每个容器的重启次数。

## web ssh访问pod内容器

访问的URL中，体现了访问api-deployment-57d46d7ff5-rxskm这个POD的nginx容器：

http://172.18.11.25:8080/public/portal/namespace/2/app/3/deployment/api

-deployment/pod/api-deployment-57d46d7ff5-rxskm/container/nginx/terminal/%E5%80%BC%E5%BE%97%E4%B9%B0k8s/default



打开了一个web ssh终端。

实现原理从源码可以看懂：

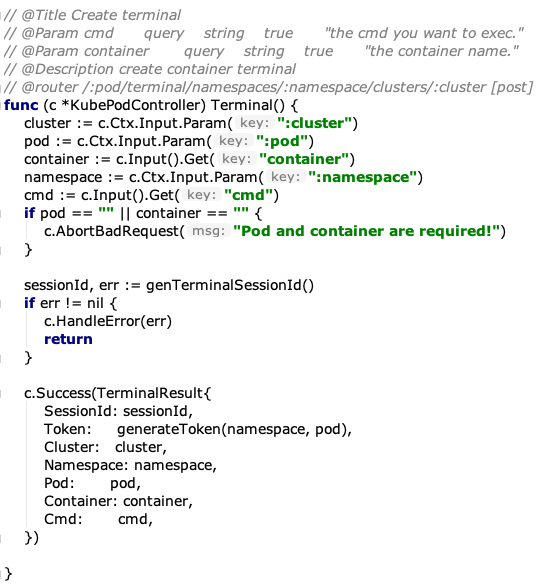
/Users/liangdong/Documents/github/golang/src/github.com/Qihoo360/wayne/src/backend/controllers/kubernetes/pod/terminal.go

Web端可以用开源xterm.js库搞定，wayne就这么搞的。

Xterm.js会通过weboscket连到wayne，wayne调用k8s client的remotecommand包可以和apiserver之间启动一个SPDY协议的长连接。

Wayne需要提供给remotecommand包3个回调函数，分别是Read,Write,Next，remotecommand会持续的调用read来获取webshell的输入，write是令wayne把终端输出通过websocket写到web端，next是remotecommand持续调用来获取是否web端改变了终端大小。

Web连websocket之前，会先到wayne申请一个token：

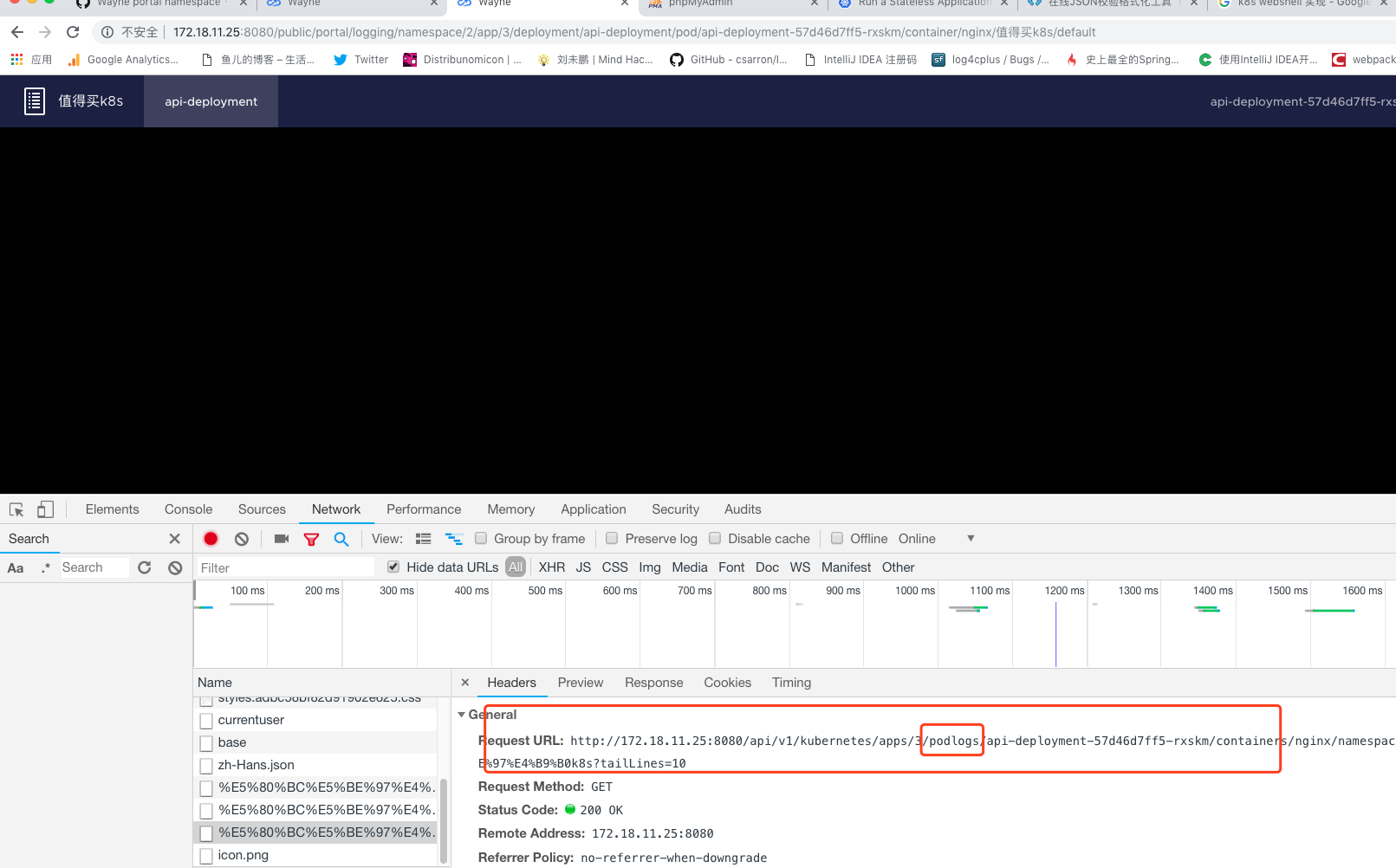


然后通过xterm.js带着token按websocket连到wayne，经过校验token后握手为websocket连接进行后续通讯。

代码不复杂，就不分析了。

## 获取pod日志

请求：



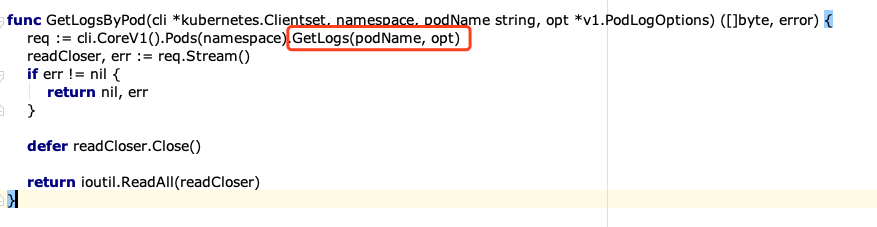
代码：

/Users/liangdong/Documents/github/golang/src/github.com/Qihoo360/wayne/src/backend/controllers/kubernetes/log/log.go

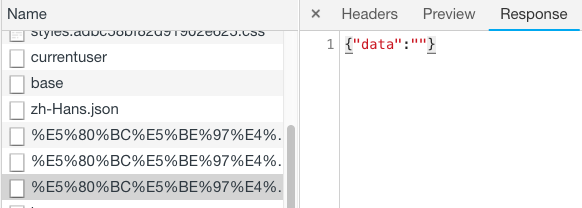
调用k8s client直接获取的：



读了末尾的N行日志：



应答：



# RBAC设计

用户user

权限permission

角色group：类型又分为app角色，namespace角色

角色权限表group\_permissions：哪个group拥有那些permission。

关联表app\_user：哪个user在哪个app赋予哪个group。

关联表namespace\_user：哪个user在哪个namespace赋予哪个group。