

拱石云平台架构

目录

- 拱石云平台架构 1
 - 云平台概述 2
 - Jenkins 配置管理 3
 - Keystone-web 4
 - SaltStack 配置管理 6
 - Salt-Master 安装配置 6
 - Salt-Minion 安装配置 6
 - Kubernetes 配置管理 7
 - Kube-Master 安装配置 7
 - Kube-Minion 安装配置 7
 - 云平台监控 13
 - Docker-Registry 配置管理 14
- 附录 14
 - ETCD 安装截图 14
 - Jenkins 安装截图 15

云平台概述

拱石云平台目前设计主要服务于拱石官网以及内部数据库服务,云平台设计之初以高可用,高扩展性,自动化管理为基本准则,做到应用发布全自动,配置集中化管理,服务器扩展自动化。技术选型和使用上,采用了最新最有潜力的技术。

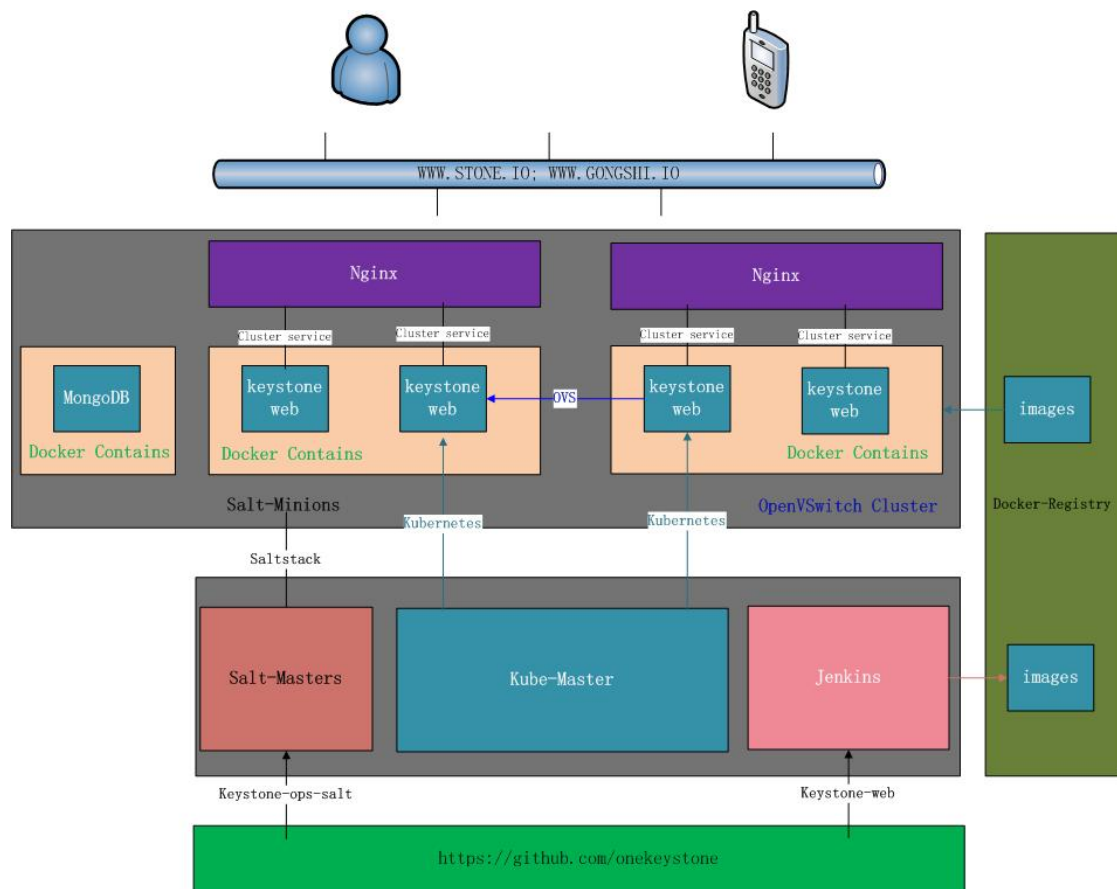
应用部署环境以及管理：CentOS7, Docker, Kubernetes, OpenVSwitch

服务器及配置管理：SaltStack

发布及流程管理：Jenkins

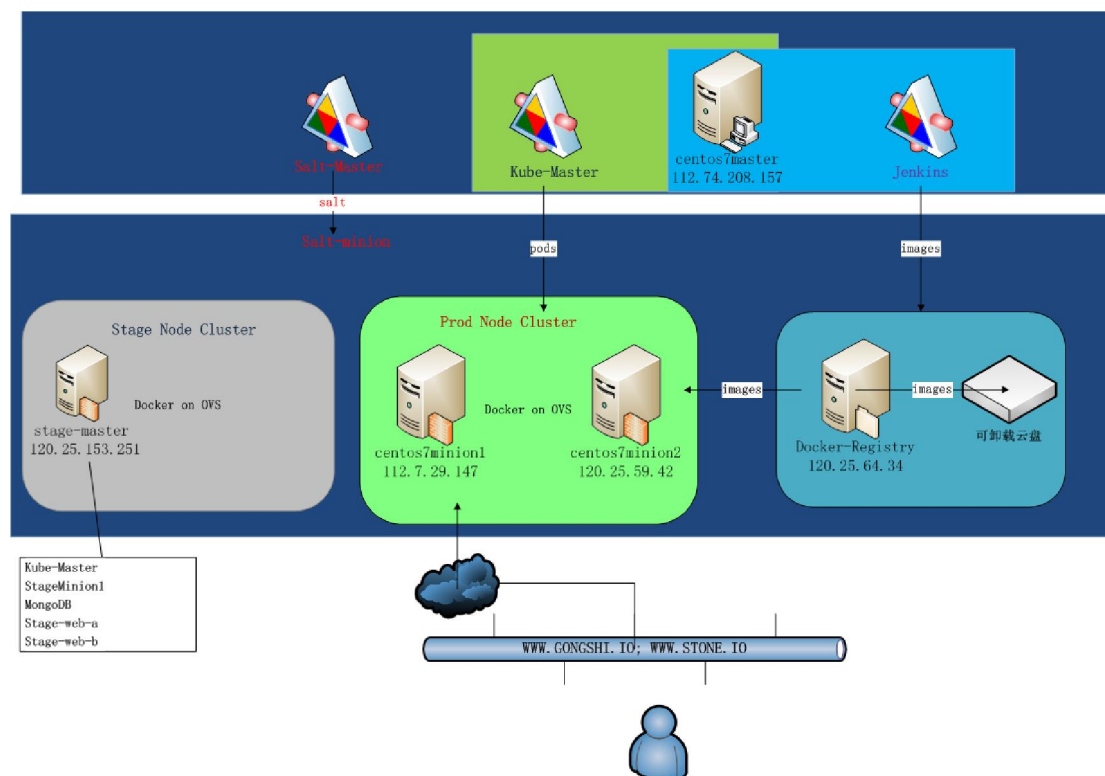
数据库: MongoDB

域名: WWW.STONE.IO



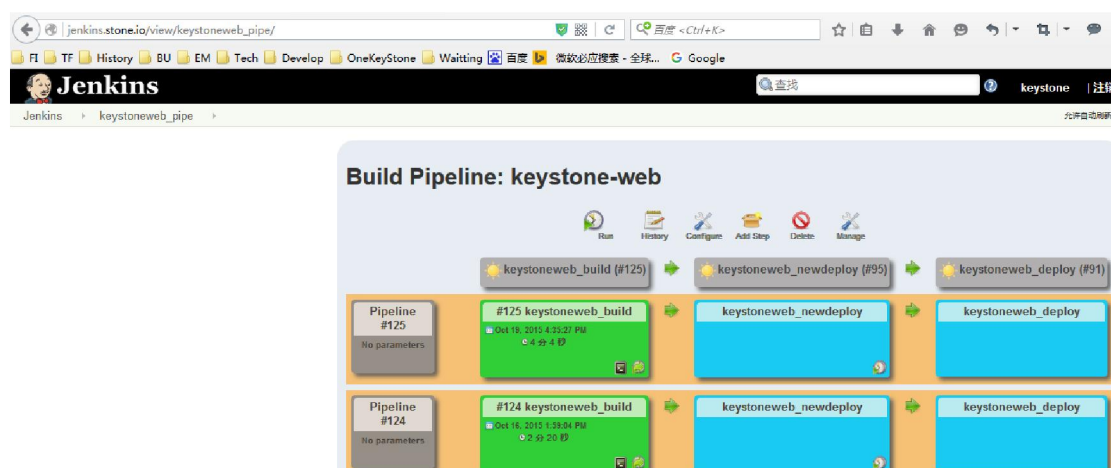
这是目前拱石所采用的技术的应用基本架构图，服务器管理通过 SaltStack, docker 容器管理使用 Kubernetes, docker 之间的虚拟网络使用 OpenVSwitch，容器对外服务的暴露通过 Nginx 反向代理。

拱石服务器选择阿里云环境，下图是初版部署服务器结构图，同时部署了 Stage 环境和 Product 环境，两套环境公用一套 Docker-Registry 环境。



Jenkins 配置管理

流程管理主要是使用 Jenkins 来管理，自动化部署，任务管理，以及驱动服务器配置都会呈现在这里。下面是生成环境的 Pipeline，生成环境目前部署了 Keystone-web 应用，应用分编译加镜像推送，发布两大步骤，发布分两次完成。



Keystone-web

下面分别就单个任务配置过程进行详解。Keystone-web 分

Allkeystonewebkeystoneweb_pipestage_web_pip+

S	W	Name ↓	上次成功	上次失败	上次持续时间	
		keystoneweb_build	2 月 18 days - #125	3 月 21 days - #44	4 分 4 秒	
		keystoneweb_create	2 月 8 days - #2	3 月 28 days - #1	0.34 秒	
		keystoneweb_deploy	2 月 18 days - #91	3 月 12 days - #68	2.2 秒	
		keystoneweb_newdeploy	2 月 18 days - #95	3 月 26 days - #5	7.4 秒	
		keystoneweb_resetdeploy	2 月 7 days - #8	无	16 秒	

图标: SML

图例RSS 全部RSS 失败RSS 最新的构建

keystoneweb-build: 自动取 Github 最新代码，编译，打包，推送 image 到 Docker-Registry.

Execute shell

Command

```
app=keystone-web
dockerdr=10.170.130.148:5000/keystone
echo $WORKSPACE

docker run -u root --rm -v $WORKSPACE:/opt/keystone 10.170.130.148:5000/keystone/web-build-env sh -c "cd /opt/keystone: ./build.sh"

docker build -t="$dockerdr/$app:v$BUILD_NUMBER" $WORKSPACE
docker push $dockerdr/$app:v$BUILD_NUMBER

docker tag -f $dockerdr/$app:v$BUILD_NUMBER $dockerdr/$app:latest
docker push $dockerdr/$app:latest

docker rmi $dockerdr/$app:v$BUILD_NUMBER
docker rmi $dockerdr/$app:latest
```

See the list of available environment variables

keystoneweb-newdeploy: 首次部分发布更新 docker 节点运行 image

Execute shell

Command

```
#Part Deploy
sh /srv/keystone-ops-salt/kube-master/scrpit/RCNewDeploy.sh "keystone-web-a" "keystone-web-b" "/srv/keystone-ops-salt/kube-master/scrpit"
```

See the list of available environment variables

keystoneweb-deploy: 第二次完成剩下部分 docker 节点更新 image

Execute shell

Command

```
#Part Deploy
sh /srv/keystone-ops-salt/kube-master/scrpit/RCDeploy.sh "keystone-web-a" "keystone-web-b" "/srv/keystone-ops-salt/kube-master/scrpit"
```

See the list of available environment variables

keystoneweb-reset: 重置更新所有 docker 节点运行 image

Execute shell

Command

```
#Part Deploy
#salt 'kube-master' state.highstate
#salt 'kube-master' cmd.run "sh /srv/salt-data/kube-master/scrpit/RCReset.sh"
sh /srv/keystone-ops-salt/kube-master/scrpit/RCReset.sh "keystone-web-a" "keystone-web-b" "/srv/keystone-ops-salt/kube-master/scrpit"
```

See the list of available environment variables

kestoneweb-create: 首次初始化 kestoneweb 的应用节点信息到 kubernetes, 这个要首先做好任务文件的配置, 代码在 keystone-ops-salt 里

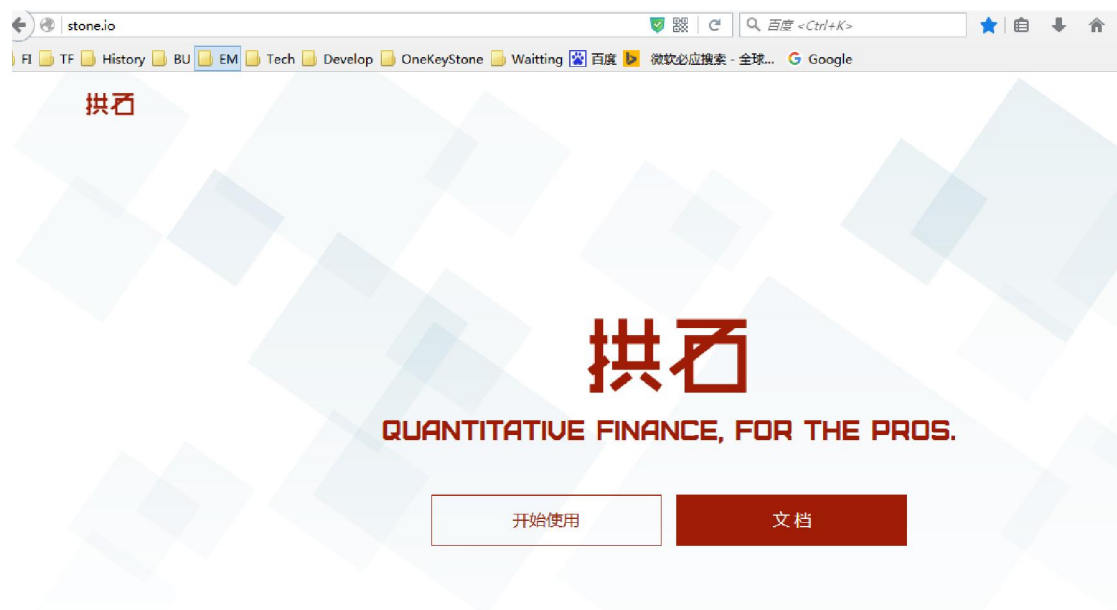
Execute shell

```
Command
groupa=keystone-web-a.json
groupb=keystone-web-b.json
svcname=keystone-web-svc.json
taskpath=/srv/keystone-ops-salt/kube-master/task

kubectl create -f $taskpath/$groupa
kubectl create -f $taskpath/$groupb
kubectl create -f $taskpath/$svcname
```

See [the list of available environment variables](#)

这样就完成了整个 keystone-web 的自动化发布过程。



keystone-web 的代码托管在 github 里, 这里 Build 任务里使用了代码钩子勾住 github, 详细过程可以查阅相关 Jenkins 的 GitHub Hook 功能。

构建触发器

- ☐ 触发远程构建 (例如, 使用脚本)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☒ Build when a change is pushed to GitHub
- ☐ Poll SCM

这样拱石的应用发布在秒级就可以完成, 整个发布过程包括编译也控制在三分钟左右。

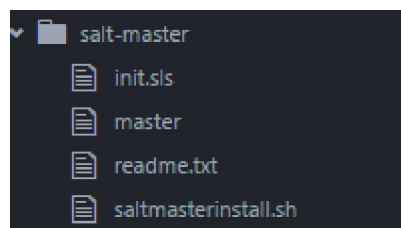
SaltStack 配置管理

服务器配置管理我们使用的是 SaltStack，SaltStack 除了能做配置管理，还能管理远程服务器，拥有丰富的 modules 以及扩展。SaltStack 分 Master 控制端和 minion 端。

Salt-Master 安装配置

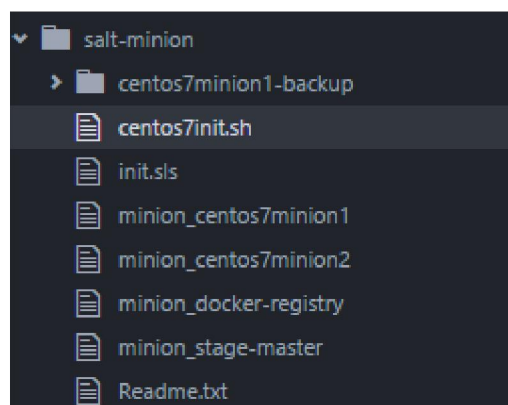
安装 salt-master 请参考 keystone-ops-salt/salt-master 下的 saltmasterinstall.sh，或者你可以直接在 salt-master 的安装服务器上运行这个 sh 文件完成安装。

配置 salt-master 则请参考 keystone-ops-salt/salt-master 下的 master 配置文件，或者直接把这个文件替换你的 master 配置文件，在 centos7 系统上这个配置文件位于/etc/salt 目录下。



Salt-Minion 安装配置

同样参照 salt-master 的安装过程，所有 salt-minion 的配置文件都存在 keystone-ops-salt/salt-minion 目录下，如果是新增机器，请参考配置文件，安装 sh 文件是 centos7init.sh .



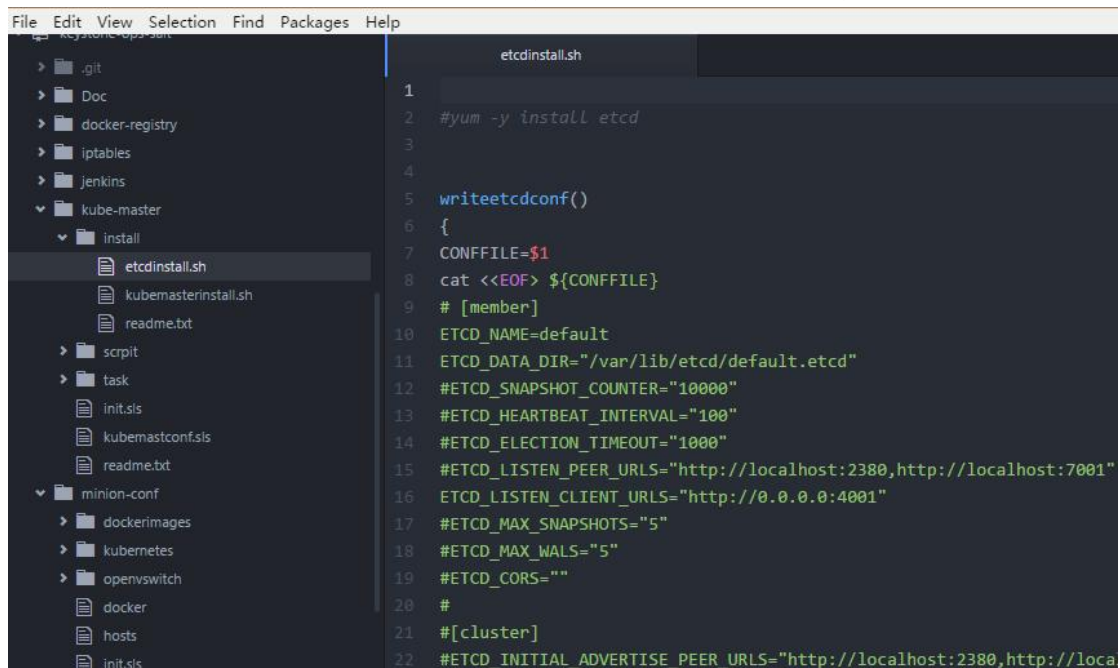
Kubernetes 配置管理

拱石的应用最终都运行在 docker 容器里，我们使用 google 的 docker 编排管理器 kubernetes 来管理我们的 docker 容器。

实际上我们通过 saltstack 来管理服务器后，所有的安装和配置都集中到 salt 代码里了，具体情况可以参考对应的代码，不过有点特殊的是，我们的 kube-master 是安装在 salt-master 的机器上，并没有完全实现自动化，如果要看完整自动化，可以看 stage_kube-master 部分。

Kube-Master 安装配置

安装过程也基本上是脚本化，k8s 的 master 安装主要分 etcd 数据库的安装，以及 api-server 的安装，具体过程请看 sh 文件，位于 keystone-ops-salt/kube-master/install 下，etcdinstall.sh 和 kubemasterinstall.sh，直接在服务器上运行对应脚本即可完成对应的安装，脚本，安装脚本被我隐掉了，因为脚本同时包含安装和配置过程，你可以手动复制安装脚本完成安装，确认安装成功之后再运行脚本完成配置。在自动化安装过程里，会分别执行软件安装的 salt 文件和配置的 salt 文件，是两个步骤，这里我集成在一个步骤里。



```
1
2 #yum -y install etcd
3
4
5 writeetcdconf()
6 {
7   CONFFILE=$1
8   cat <<EOF > ${CONFFILE}
9   # [member]
10  ETCD_NAME=default
11  ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
12  #ETCD_SNAPSHOT_COUNTER="10000"
13  #ETCD_HEARTBEAT_INTERVAL="100"
14  #ETCD_ELECTION_TIMEOUT="1000"
15  #ETCD_LISTEN_PEER_URLS="http://localhost:2380,http://localhost:7001"
16  ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:4001"
17  #ETCD_MAX_SNAPSHOTS="5"
18  #ETCD_MAX_WALS="5"
19  #ETCD_CORS=""
20  #
21  #[cluster]
22  #ETCD_INITIAL_ADVERTISE_PEER_URLS="http://localhost:2380,http://loca
```

Kube-Minion 安装配置

实际上所有 minion 服务器的安装配置都是通过 salt 集中配置，自动驱动的，但是这里要介绍一下原理和基本过程。

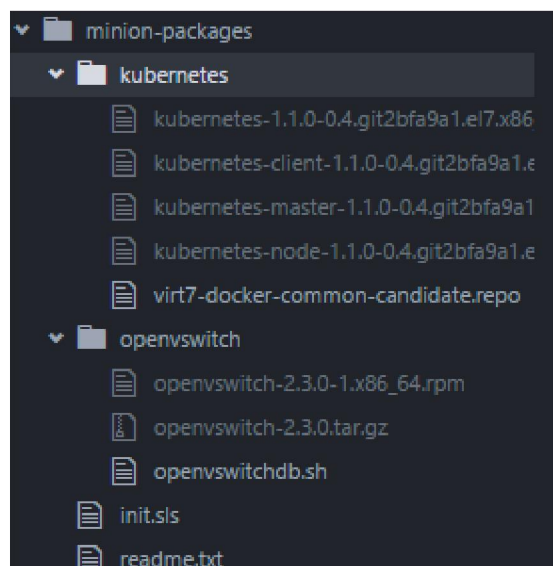
下面这个文件是 salt 的 top.sls 驱动文件，对应服务器驱动过程在各个节点下面，centos7minion*是标准的 kube-minion 的应用节点配置，主要包含两个部分，package 和 conf 部分，具体过程可以深入两个部分的文件细节去查看，这里总体介绍一下。安装过程主要是完成软件环境安装，以及软件配置。软件环境安装包含 kube-minion 环境，软件包，docker 环境，openvswitch 环境，以及 nginx 环境。软件配置主要包括，kube 的配置，openvswitch

网络配置，路由规则配置，docker 网桥配置，nginx 配置，等。

```
base:
  # salt-master
  'centos7master':
    - kube-master
    - kube-master.task
  'docker-registry':
    - docker-registry
    - iptables.docker-registry
  'centos7minion*':
    - minion-packages
    - minion-conf
  'stage-master*':
    - minion-packages
    - kube-master
    - stage_kube-master
    - stage_minion-conf
    - stage_kube-master.task
```

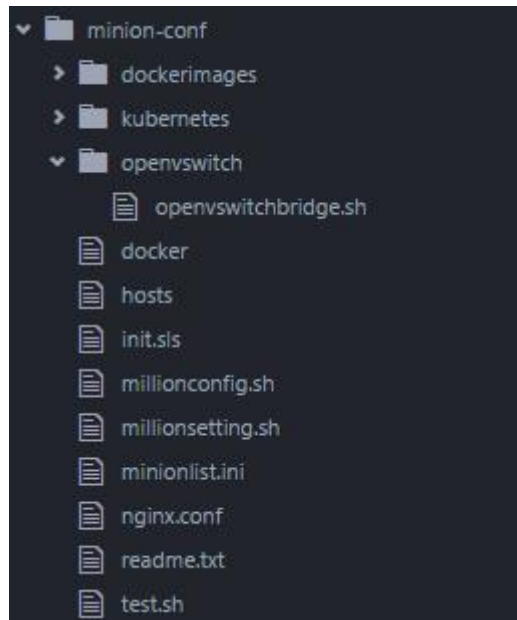
软件包安装

软件包安装详情可以在 minion-package 的 init.sls 里查看详情，默认小软件包直接使用 salt 的自动安装，大软件包为了加速自动化过程，以及减少网络问题带来的不确定性，在 salt-master 本地做了缓存，复制到目标 minion 机器上进行安装，所有过程都是自动化的。



配置管理

配置管理也是完全自动化的，在 repo 里的 minion-conf 里，由 init.sls 驱动完成所有的配置。配置分几个部分，docker 基础 image 加载，kube 基本配置，ip，路由，以及 openvswitch 配置，nginx 代理配置。



Docker base image

由于 docker 运行需要两个基础 images，默认这两个 images 可能被墙了，所以通过我们内部管理进行加载。

Kube-minion

Kube-minion 的配置主要是初始化几个配置文件，minion 端主要依赖 config, kubelet, proxy 这三个配置，所有 minion 都是一样的，这三个配置采用文件复制的形式复制过去，具体细节有 sls 文件控制。

OpenVSwitch

使用 openvswitch 的目的是打通容器之间的网络，让容器之间可以通信，这样才有可能让容器中的 web 应用访问容器中的 db 应用，构建容器之间的虚拟网络有很多手段，比如 Flannel, Weave 以及 OpenvSwitch。经过一些比较，我们选择最为专业的 OpenvSwitch 来作为我们的网桥工具，因为 OVS 的延迟最小，速度最快。

OVS 的配置融合进了 sh 文件进行初始化，主要工作是完成 IP 配置，网桥设置，GRE 网桥关联，路由规则。这些配置需要跟所有加入 OVS 网络的服务器节点进行互相配置，所以这部分过程稍微复杂一点，需要理解 OVS 的概念以及原理。所有配置细节做进了 sh 文件，

sh 文件根据基础配置文件 minionlist.ini 完成所有配置，这里简单介绍一下 minionlist.ini 配置文件。

```
[comon]
minions=centos7minion1,centos7minion2,end
minioncount=1

[centos7minion1]
remoteip=10.170.9.74
kbrip=10.244.1.1
kbrgetway=10.244.0.0
kbripfile=/etc/sysconfig/network-scripts/ifcfg-kbr0
obripfile=/etc/sysconfig/network-scripts/ifcfg-obr0
routedev=kbr0
routeipfile=/etc/sysconfig/network-scripts/route-kbr0
kubeletconf=/etc/kubernetes/kubelet
kubernetesconfig=/etc/kubernetes/config
api_server=http://centos7master:8080
etcd_server=http://centos7master:4001

[centos7minion2]
remoteip=10.170.113.203
kbrip=10.244.2.1
kbrgetway=10.244.0.0
kbripfile=/etc/sysconfig/network-scripts/ifcfg-kbr0
obripfile=/etc/sysconfig/network-scripts/ifcfg-obr0
routedev=kbr0
routeipfile=/etc/sysconfig/network-scripts/route-kbr0
kubeletconf=/etc/kubernetes/kubelet
kubernetesconfig=/etc/kubernetes/config
api_server=http://centos7master:8080
etcd_server=http://centos7master:4001

[end]
```

```
[comon]
minions=centos7minion1,centos7minion2,end
minioncount=1
```

这部分主要起作用的是 minions 节点,sh 文件会根据这个节点解析位于 end 之前的节点名称,然后去后面节点找到对应的节点进行解析。

找到了节点名称之后,去下面的节点组获取节点详细信息,如下,

remoteip: 是本地内网 IP, 设置 GRE 网桥的时候要用

kbrip: 是内部网桥 kbr0 的容器节点虚拟 ip, 需要被设置给容器

kbrgetway: 是内部网桥默认路由组

kbripfile: 是 centos7 默认 kbr0 网桥的 ip 文件

obriple: 是 centos7 默认的 obr0 网桥的 ip 文件
routedev: 路由规则设备
routefile: centos7 路由规则对应设备的存储规则文件路径
kubeletconf: kubelet 配置文件路径
kubernetesconfig: kube 的 config 文件路径
api_server: kube 的 master 服务器地址
etcd_server: etcd 数据服务器地址

```
[centos7minion1]
remoteip=10.170.9.74
kbrip=10.244.1.1
kbrgetway=10.244.0.0
kbripfile=/etc/sysconfig/network-scripts/ifcfg-kbr0
obriple=/etc/sysconfig/network-scripts/ifcfg-obr0
routedev=kbr0
routefile=/etc/sysconfig/network-scripts/route-kbr0
kubeletconf=/etc/kubernetes/kubelet
kubernetesconfig=/etc/kubernetes/config
api_server=http://centos7master:8080
etcd_server=http://centos7master:4001
```

这些规则可以随时增加，sh 文件或自动进行扩展匹配。

IP and Router

ip 和路由规则在上面，这里要理解虚拟子网和路由的对应规则。如下

```
[root@centos7minion2 ~]# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
0.0.0.0          120.25.59.247   0.0.0.0         UG    0      0          0 eth1
10.0.0.0          10.170.119.247  255.0.0.0       UG    0      0          0 eth0
10.170.112.0      0.0.0.0         255.255.248.0   U      0      0          0 eth0
10.244.0.0        0.0.0.0         255.255.0.0     U      0      0          0 kbr0
10.244.2.0        0.0.0.0         255.255.255.0   U      0      0          0 kbr0
11.0.0.0          10.170.119.247  255.0.0.0       UG    0      0          0 eth0
100.64.0.0        10.170.119.247  255.192.0.0     UG    0      0          0 eth0
120.25.56.0       0.0.0.0         255.255.252.0   U      0      0          0 eth1
169.254.0.0       0.0.0.0         255.255.0.0     U      1002   0          0 eth0
169.254.0.0       0.0.0.0         255.255.0.0     U      1003   0          0 eth1
169.254.0.0       0.0.0.0         255.255.0.0     U      1006   0          0 kbr0
172.16.0.0        10.170.119.247  255.240.0.0     UG    0      0          0 eth0
192.168.0.0       10.170.119.247  255.255.0.0     UG    0      0          0 eth0
[root@centos7minion2 ~]# ip route
default via 120.25.59.247 dev eth1
10.0.0.0/8 via 10.170.119.247 dev eth0
10.170.112.0/21 dev eth0 proto kernel scope link src 10.170.113.203
10.244.0.0/16 dev kbr0
10.244.2.0/24 dev kbr0 proto kernel scope link src 10.244.2.1
11.0.0.0/8 via 10.170.119.247 dev eth0
100.64.0.0/10 via 10.170.119.247 dev eth0
120.25.56.0/22 dev eth1 proto kernel scope link src 120.25.59.42
169.254.0.0/16 dev eth0 scope link metric 1002
169.254.0.0/16 dev eth1 scope link metric 1003
169.254.0.0/16 dev kbr0 scope link metric 1006
172.16.0.0/12 via 10.170.119.247 dev eth0
192.168.0.0/16 via 10.170.119.247 dev eth0
[root@centos7minion2 ~]#
```

Nginx

应用运行在 docker 容器内部，通过 kubernetes 的 service 方式暴露出来，为了将服务暴露到外网，每个 minion 上都安装了 nginx 做反向代理，这样就同时解决了服务暴露和负载均衡的问题了。这个配置会通过 salt 直接自动替换服务器上的 nginx 配置，配置节点主要是要跟 keystone-web 的内部服务 ip 对应。

```
location / {  
    proxy_pass http://10.254.0.244;  
    proxy_http_version 1.1;  
    proxy_set_header Upgrade $http_upgrade;  
    proxy_set_header Connection 'upgrade';  
    proxy_set_header Host $host;  
    proxy_cache_bypass $http_upgrade;  
}
```

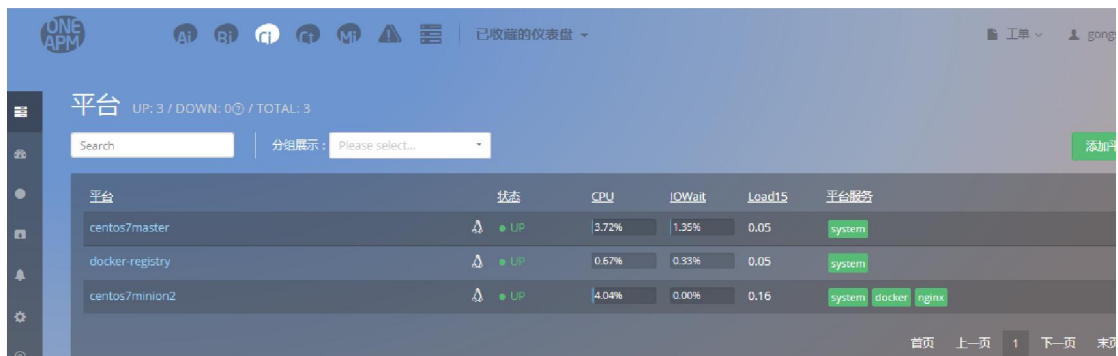
云平台监控

监控部分探索过 ELK(Elasticsearch, Logstash, Kibana)，但是初期考虑到应用复杂度不高，暂时没有部署 ELK 组合来做监控，而是使用目前免费的第三方监控平台 ONEAPM。



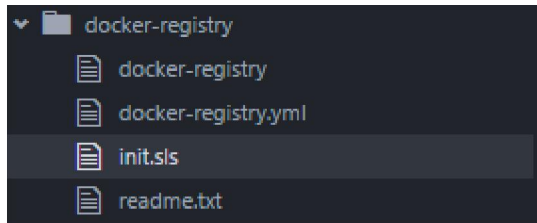
目前我们使用了 ONEAPM 的 AI 和 CI 两个功能，Keystone-web 使用了 nodejs 的 AI，服务器本身使用了 CI。

具体可登陆 ONEAPM 进行查看，目前使用的是我的账户 155291330@qq.com，密码是我们的服务器初始密码。



Docker-Registry 配置管理

我们的 image 仓库是一台单独的云服务器，挂了一个可卸载云盘，服务器上单独安装 docker-registry 组件，具体安装过程也是通过 salt 完全自动驱动的，具体细节请查看



附录

ETCD 安装截图

```
[root@CentOS7Master install]# sh etcdinstall.sh
Loaded plugins: langpacks
base
epel
extras
updates
(1/7): epel/x86_64/group_gz
(2/7): base/7/x86_64/group_gz
(3/7): epel/x86_64/updateinfo
(4/7): extras/7/x86_64/primary_db
(5/7): epel/x86_64/primary_db
(6/7): updates/7/x86_64/primary_db
(7/7): base/7/x86_64/primary_db
Resolving Dependencies
--> Running transaction check
--> Package etcd.x86_64 0:2.0.13-2.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                        Arch                Version
=====
Installing:
etcd                           x86_64              2.0.13-2.el7

Transaction Summary
=====
Install 1 Package

Total download size: 2.9 M
Installed size: 12 M
```



```

Installed:
  etcd.x86_64 0:2.0.13-2.el7

Complete!
ln -s '/usr/lib/systemd/system/etcd.service' '/etc/systemd/system/multi-user.target.wants/etcd.service' - Etcd Server
   Loaded: loaded (/usr/lib/systemd/system/etcd.service; enabled)
   Active: active (running) since Thu 2015-08-27 10:36:05 CST; 87ms ago
 Main PID: 30221 (etcd)
    CGroup: /system.slice/etcd.service
            └─30221 /usr/bin/etcd

Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 etcdserver:
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 etcdserver:
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 etcdserver:
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 etcdserver:
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 etcdserver:
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 etcdserver:
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 raft: ce2a82
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 raft: newRa
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 raft: ce2a82
Aug 27 10:36:05 CentOS7Master bash[30221]: 2015/08/27 10:36:05 etcdserver:
Hint: Some lines were ellipsized, use -l to show in full.

```

Jenkins 安装截图

```

Last login: Tue Aug 11 14:04:41 2015 from 192.168.93.1
[root@centos-master ~]# wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
--2015-08-11 17:02:10-- http://pkg.jenkins-ci.org/redhat/jenkins.repo
Resolving pkg.jenkins-ci.org (pkg.jenkins-ci.org)... 199.193.196.24
Connecting to pkg.jenkins-ci.org (pkg.jenkins-ci.org)|199.193.196.24|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 75 [text/plain]
Saving to: '/etc/yum.repos.d/jenkins.repo'

100%[*****>] 75 --.-K/s 1s

2015-08-11 17:02:11 (12.5 MB/s) - '/etc/yum.repos.d/jenkins.repo' saved [75/75]

[root@centos-master ~]# rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
[root@centos-master ~]# yum install jenkins
Loaded plugins: fastestmirror
base                                                    | 3.6 kB 00
epel/x86_64/metalink                                   | 3.3 kB 00
epel                                                    | 4.4 kB 00
extras                                                  | 3.4 kB 00
jenkins                                                 | 951 B 00
updates                                                | 3.4 kB 00
virt7-docker-common-candidate                         | 3.4 kB 00
(1/4): extras/7/x86_64/primary_db                    | 74 kB 00
(2/4): virt7-docker-common-candidate/primary_db      | 16 kB 00
(3/4): updates/7/x86_64/primary_db                  | 3.2 MB 00
(4/4): epel/x86_64/primary_db                        | 3.7 MB 00
(1/3): epel/x86_64/updateinfo                        | 454 kB 00
(2/3): epel/x86_64/packages                          | 1.6 MB 00
(3/3): jenkins/primary                               | 32 kB 00
Determining fastest mirrors
 * base: centos.ustc.edu.cn
 * epel: ftp.cuhk.edu.hk
 * extras: centos.ustc.edu.cn
 * updates: mirrors.sina.cn
jenkins
Resolving Dependencies
--> Running transaction check
--> Package jenkins.noarch 0:1.624-1.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository
=====
Installing:
jenkins noarch 1.624-1.1 jenkins
Transaction Summary
-----
Install 1 Package

Total download size: 60 M
Installed size: 60 M
Is this ok [y/d/N]: y

```