**普元信息技术股份有限公司**

**数字化企业云平台**

**部署指南**

[一、 文档目的 5](#_Toc461958382)

[二、 阅读对象 5](#_Toc461958383)

[三、 文档概述 5](#_Toc461958384)

[3.1 术语说明 5](#_Toc461958385)

[四、 基础环境搭建 6](#_Toc461958386)

[4.1服务器准备 6](#_Toc461958387)

[4.2服务器操作系统升级 6](#_Toc461958388)

[4.2.1 CoreOS系统升级 6](#_Toc461958389)

[4.2.2 CentOS系统升级 8](#_Toc461958390)

[4.3 CoreOS初始化 8](#_Toc461958391)

[4.3.1 创建user\_data文件 8](#_Toc461958392)

[4.3.2 初始化 12](#_Toc461958393)

[4.4 数据盘准备 12](#_Toc461958394)

[4.4.1 数据盘分区 12](#_Toc461958395)

[4.4.2 格式化 14](#_Toc461958396)

[4.4.3 挂载 14](#_Toc461958397)

[4.5 NFS搭建 14](#_Toc461958398)

[4.5.1 搭建NFS Server 14](#_Toc461958399)

[4.5.2 搭建NFS Client 15](#_Toc461958400)

[4.6 准备Docker仓库 16](#_Toc461958401)

[4.6.1 搭建Docker仓库 16](#_Toc461958402)

[4.6.2 镜像和容器的导入导出 17](#_Toc461958403)

[4.6.3 设置仓库访问安全认证 17](#_Toc461958404)

[4.7 Kubernetes集群安装 18](#_Toc461958405)

[4.7.1 安装flannel组件 18](#_Toc461958406)

[4.7.2 安装Kubernetes管理节点 20](#_Toc461958407)

[4.7.3 安装Kubernetes工作节点 24](#_Toc461958408)

[4.7.4 验证Kubernetes集群 26](#_Toc461958409)

[4.8 泛域名配置 26](#_Toc461958410)

[4.8.1 阿里云环境配置 26](#_Toc461958411)

[4.8.2 私有云环境 27](#_Toc461958412)

[4.9 OpenResty安装 27](#_Toc461958413)

[4.9.1 复制目录 27](#_Toc461958414)

[4.9.2 编译并上传镜像 28](#_Toc461958415)

[4.9.3 创建相关服务 28](#_Toc461958416)

[4.9.4 查看服务启动 28](#_Toc461958417)

[4.9.5 修改配置 28](#_Toc461958418)

[五、 第三方依赖产品安装 29](#_Toc461958419)

[5.1 Etcd集群搭建 29](#_Toc461958420)

[5.1.1 创建目录 29](#_Toc461958421)

[5.1.2 添加yml文件 29](#_Toc461958422)

[5.1.3 创建k8s namespace 35](#_Toc461958423)

[5.1.4 启动etcd 35](#_Toc461958424)

[5.1.5 验证etcd集群 36](#_Toc461958425)

[5.2 安装jenkins 36](#_Toc461958426)

[5.2.1 添加yml文件 36](#_Toc461958427)

[5.2.2 创建k8s namespace 38](#_Toc461958428)

[5.2.3 启动jenkins 39](#_Toc461958429)

[5.2.4 验证jenkins 39](#_Toc461958430)

[5.3 安装Nexus 40](#_Toc461958431)

[5.3.1 添加yml文件 40](#_Toc461958432)

[5.3.2 创建nexus-data目录 42](#_Toc461958433)

[5.3.3 启动nexus 42](#_Toc461958434)

[5.3.4 验证nuxus 42](#_Toc461958435)

[5.4 安装GitLab 43](#_Toc461958436)

[5.4.1 拉取Gitlab镜像 43](#_Toc461958437)

[5.4.2 添加Gitlab启动脚本 43](#_Toc461958438)

[5.4.3 关闭防火墙 44](#_Toc461958439)

[5.4.4 启动Gitlab 44](#_Toc461958440)

[5.4.5 验证Gitlab 44](#_Toc461958441)

[5.4.6 创建管理员 45](#_Toc461958442)

[5.5 ElasticSearch集群安装 45](#_Toc461958443)

[5.5.1 安装ElasticSearch 45](#_Toc461958444)

[5.5.2 安装jdk1.8 45](#_Toc461958445)

[5.5.3 添加配置文件 45](#_Toc461958446)

[5.5.4 创建es数据路径 47](#_Toc461958447)

[5.5.5 设置环境变量 47](#_Toc461958448)

[5.5.6 添加service文件 48](#_Toc461958449)

[5.5.7 启动服务 50](#_Toc461958450)

[5.5.8 设置分词索引 50](#_Toc461958451)

[5.6 Influxdb安装 50](#_Toc461958452)

[5.6.1 安装Influxdb 50](#_Toc461958453)

[5.6.2 添加配置文件 51](#_Toc461958454)

[5.6.3 优化udp os buffer size 51](#_Toc461958455)

[5.6.4 赋予influxdb用户权限 51](#_Toc461958456)

[5.6.5 添加service文件 51](#_Toc461958457)

[5.6.6 启动服务 52](#_Toc461958458)

[5.7 Fluentd安装 52](#_Toc461958459)

[5.7.1 安装fluentd 52](#_Toc461958460)

[5.7.2 安装相关插件 53](#_Toc461958461)

[5.7.3 修改配置文件 53](#_Toc461958462)

[5.7.4 启动服务 53](#_Toc461958463)

[5.8 Heapster安装 54](#_Toc461958464)

[5.8.1 拉取镜像 54](#_Toc461958465)

[5.8.2 添加配置文件 54](#_Toc461958466)

[5.8.3 启动Heapster 55](#_Toc461958467)

[5.9 日志采集器 55](#_Toc461958468)

[5.9.1 创建服务文件 56](#_Toc461958469)

[5.9.2 启动服务 57](#_Toc461958470)

[5.10 安装Mysql数据库 57](#_Toc461958471)

[5.10.1 添加yml文件 57](#_Toc461958472)

[5.10.2 启动Mysql 59](#_Toc461958473)

[5.10.3 初始化数据库 60](#_Toc461958474)

[5.10.4 修改数据 61](#_Toc461958475)

[六、 平台领域系统部署 61](#_Toc461958476)

[6.1 后端领域系统部署 61](#_Toc461958477)

[6.2 平台前端应用部署 66](#_Toc461958478)

[6.2.1 添加启动脚本 66](#_Toc461958479)

[6.2.2 修改启动脚本权限 67](#_Toc461958480)

[6.2.3 拉取应用镜像 67](#_Toc461958481)

[6.2.4 启动应用 67](#_Toc461958482)

1. 文档目的

为实施人员进行普元数字化企业云平台部署工作提供参考。

1. 阅读对象

希望进行普元数字化企业云平台安装部署，或了解整个平台部署过程的实施人员。

1. 文档概述

普元数字化企业云平台（The Platform）是普元信息技术股份有限公司研发的Devops平台，能为用户提供产品从开发、测试到生产、运维的全生命周期管理，和一键编译打包的自动化能力支撑，实现应用与基础设施的解耦，为应用向云平台迁移打下基础。

数字化企业云平台基于Kubernetes集群（k8s集群）和Docker技术实现，从物理视角来看，整个平台由三部分构成：平台k8s集群、用户k8s集群、独立节点。

平台k8s集群：用于管理、部署平台的所有领域系统。

用户k8s集群：用于部署用户的微应用、微服务，用户在平台上进行为应用或微服务的管理，通过打包部署功能，将产品构建成Docker镜像，然后部署到用户k8s集群上运行。

独立节点：平台除了领域系统之外，还需要依赖一些第三方产品，例如GitLab、Etcd、InfluxDb等。这些第三方依赖产品将会部署到k8s集群之外的独立节点上。

本文描述了普元数字化企业云平台的安装部署过程，后文中将整个过程分为以下三个部分进行描述：基础环境搭建、第三方依赖产品安装、平台领域系统部署。

3.1 术语说明

The Platform平台：普元数字化企业云平台的简称。

k8s集群：Kubernetes集群简称。

k8s Master节点：Kubernetes集群中的管理节点。

k8s WorkNode节点：Kubernetes集群中的计算节点。

WorkNodeIP：pod所在的Kubernetes集群计算节点的实际IP。

NodePort：启动pod时，xxx-svc.yml文件中所配置的NodePort参数

1. 基础环境搭建

进行The Platform平台的安装部署工作之前，需要在机器节点上先搭建好Kubernetes集群等基础环境。基础环境的搭建步骤如下所述。

4.1服务器准备

下表是The Platform平台的阿里云环境配置信息，用户可根据自己的需要，结合下表中的信息进行服务器准备。

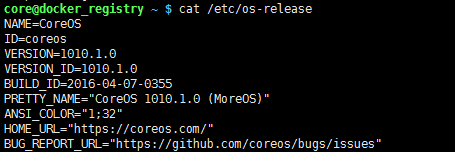
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **节点分类** | **节点用途** | **节点数量** | **操作系统** | **配置** | **数据盘** | **数据盘说明** | **数据磁盘类型** | **公网IP** |
| 平台k8s集群 | k8s master | 1 | CoreOS | 4C/8G/40G | 2T | NFS | SSD | 100M |
| k8s work node | 3 | CoreOS | 16C/64G/100G | 无 | 共享master |  |  |
| 平台独立节点 | Portal Client | 1 | CoreOS | 2C/4G/40G | 500G |  | 高效云盘 | 100M |
| Git | 1 | CoreOS | 4C/8G/40G | 500G |  | 高效云盘 | 100M |
| Docker Registry | 1 | CoreOS | 4C/8G/40G | 500G |  | 高效云盘 |  |
| InfluxDB | 1 | CentOS7 | 8C/16G/40G | 1T |  | 高效云盘 |  |
| Elastic Search | 2 | CentOS7 | 8C/16G/40G | 1T |  | 高效云盘 |  |
| 微服务k8s集群 | k8s master | 1 | CoreOS | 4C/8G/40G | 2T | NFS | SSD |  |
| k8s work node | 5 | CoreOS | 16C/64G/100G | 无 | 共享master |  |  |
| 微服务独立节点 | Redis | 3 | CentOS7 | 4C/8G/200G | 无 |  |  |  |
| 跳板机 | linux跳板机 | 1 | CentOS7 | 4C/8G/200G | 无 |  |  |  |
| windows跳板机 | 1 | windows7 | 4C/8G/200G | 无 |  |  |  |

4.2服务器操作系统升级

The Platform平台对服务器节点的操作系统版本有一定要求，所以准备好服务器节点之后，需要对各服务器节点的操作系统进行升级。进行升级时，需要保证服务器节点能连接到互联网。

4.2.1 CoreOS系统升级

使用**cat /etc/os-release**命令能查看到当前CoreOS的版本信息。



CoreOS操作系统需要升级到官方最新稳定版本，升级步骤如下。

1. 使用**vim /etc/coreos/update.conf**命令，修改ect/coreos/update.conf文件中的Group属性，让操作系统升级的是稳定版。

|  |
| --- |
| GROUP=stable |

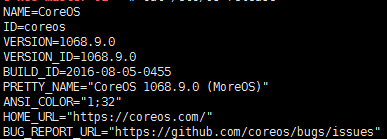
1. 执行更新命令

|  |
| --- |
| **update\_engine\_client --update** |

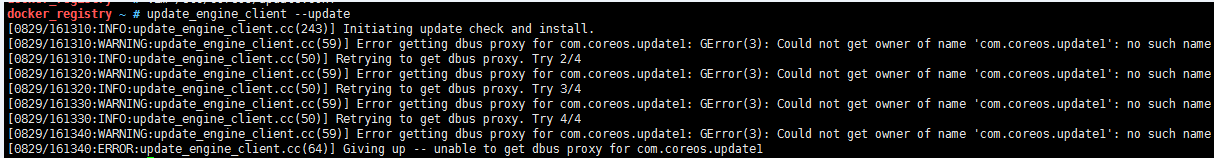
1. 完成升级后可通过下列命令查看操作系统版本

|  |
| --- |
| **cat /etc/os-release** |

版本信息如下。



CoreOS执行更新命令时，可能会出现以下异常信息。

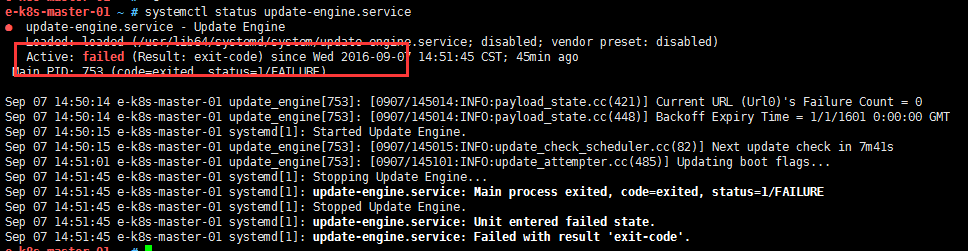


如果出现该异常，可通过以下步骤进行解决。

1. 执行**systemctl status xxxxx.service**命令查看以下两个服务是否是停止状态。

update-engine.service

locksmithd.service



1. 如果上面两个服务是停止状态，使用**systemctl start xxxxx.service**命令，将两个服务启动
2. 执行**update\_engine\_client --update**命令进行过系统升级
3. 完成升级后，执行**systemctl stop update-engin.service**命令，停止上文中的两个服务，然后重启机器

4.2.2 CentOS系统升级

CentOS可以通过执行**cat /etc/centos-release**命令来产看当前系统版本。



CentOS只需要执行下面一行命令即可完成升级操作。该命令将会升级系统的所有包，但不升级操作系统内核。

|  |
| --- |
| **yum -y upgrade** |

完成升级后执行下列命令，进行Docker的安装

|  |
| --- |
| **yum install docker -y** |

4.3 CoreOS初始化

k8s集群中的CoreOS系统需要进行初始化操作，安装一些基础服务。

4.3.1 创建user\_data文件

在k8s集群的每一个节点中都创建user\_data文件。

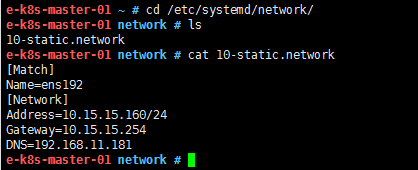
|  |
| --- |
| **mkdir /var/lib/coreos-install**  **vim /var/lib/coreos-install/user\_data** |

Master节点的文件内容如下：

|  |
| --- |
| #cloud-config  write\_files:    - path: "/etc/hosts"      permissions: "0644"      owner: "root"      content: |        127.0.0.1        localhost        10.15.15.160   e-k8s-master-01        10.15.15.161   e-k8s-work-01        10.15.15.162   e-k8s-work-02  #docker 仓库地址        10.15.15.164 euler-registry.primeton.com  hostname: e-k8s-master-01  coreos:    flannel:      interface: eth0    etcd2:      name: "%H"      heartbeat-interval: 1000      election-timeout: 5000      advertise-client-urls: http://10.15.15.160:2379      initial-advertise-peer-urls: http://10.15.15.160:2380      listen-client-urls: http://0.0.0.0:2379      listen-peer-urls: http://0.0.0.0:2380      initial-cluster-token: euler-02-cluster  initial-cluster: e-k8s-master-01=http://10.15.15.160:2380,e-k8s-work-01=http://10.15.15.161:2380,e-k8s-work-02=http://10.15.15.162:2380      initial-cluster-state: new    units:    - name: etcd2.service      command: start    - name: fleet.service      command: start    - name: flanneld.service      command: start    - name: docker.service      command: start    - name: timezone.service      command: start      content: |        [Unit]        Description=timezone        [Service]        Type=oneshot        RemainAfterExit=yes        ExecStart=/usr/bin/ln -sf ../usr/share/zoneinfo/Asia/Shanghai /etc/localtime      - name: 10-static.network      runtime: false      content: |        [Match]        Name=ens192        [Network]        Address=10.15.15.160/24        Gateway=10.15.15.254        DNS=192.168.11.181    - name: update-engine.service      command: stop    - name: locksmithd.service      command: stop  users:  - name: core    passwd: $1$IPTUqn3M$SCVKS0jdbpmdoKbawV7ut/    groups:    - sudo    - docker |

注：

a、文件中的Network信息可通过查看/etc/systemd/network下面的文件得到，如果有多个.neetwork文件，就配置多个。



b、users项在阿里云环境中不需要进行配置，因为阿里云直接使用的是root用户。

工作节点的文件内容如下：

|  |
| --- |
| #cloud-config  write\_files:    - path: "/etc/hosts"      permissions: "0644"      owner: "root"      content: |        127.0.0.1        localhost        10.15.15.160   e-k8s-master-01        10.15.15.161   e-k8s-work-01        10.15.15.162  e-k8s-work-02        10.15.15.164 euler-registry.primeton.com  hostname: e-k8s-work-01  coreos:    flannel:      interface: eth0    etcd2:      name: "%H"      heartbeat-interval: 1000      election-timeout: 5000      advertise-client-urls: http://10.15.15.161:2379      initial-advertise-peer-urls: http://10.15.15.161:2380      listen-client-urls: http://0.0.0.0:2379      listen-peer-urls: http://0.0.0.0:2380      initial-cluster-token: euler-02-cluster  initial-cluster: e-k8s-master-01=http://10.15.15.160:2380,e-k8s-work-01=http://10.15.15.161:2380,e-k8s-work-02=http://10.15.15.162:2380      initial-cluster-state: new    units:    - name: etcd2.service      command: start    - name: fleet.service      command: start    - name: flanneld.service      command: start    - name: docker.service      command: start    - name: timezone.service      command: start      content: |        [Unit]        Description=timezone        [Service]        Type=oneshot        RemainAfterExit=yes        ExecStart=/usr/bin/ln -sf ../usr/share/zoneinfo/Asia/Shanghai /etc/localtime    - name: 10-static.network      runtime: false      content: |        [Match]        Name=ens192        [Network]        Address=10.15.15.161/24        Gateway=10.15.15.254        DNS=192.168.11.181    - name: update-engine.service      command: stop    - name: locksmithd.service      command: stop |

注：文件中的Network信息可通过查看/etc/systemd/network下面的文件得到，如果有多个.neetwork文件，就配置多个。

4.3.2 初始化

|  |
| --- |
| **执行初始化服务：**  **coreos-cloudinit --from-file /var/lib/coreos-install/user\_data** |

4.3.3 重启机器

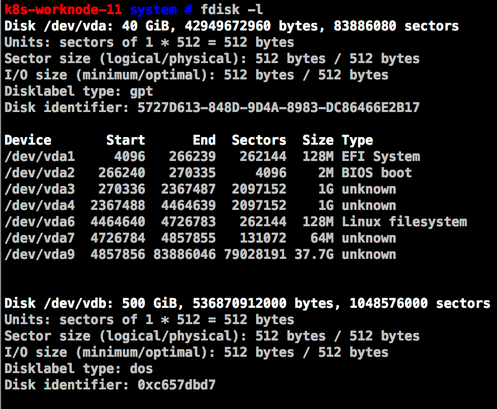
|  |
| --- |
| **重启机器**  **reboot** |

4.4 数据盘准备

完成CoreOS初始化后，需要对节点的数据盘进行分区、格式化和挂载操作。

4.4.1 数据盘分区

1. 执行**fdisk -l**命令，查看当前系统所有硬盘信息。



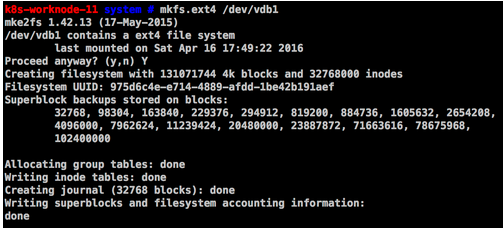
上图中可以看到有一个名为/dev/vdb的磁盘

1. 执行硬盘分区命令

|  |
| --- |
| [root@node ~]#**fdisk /dev/vdb**  Command (m for help): **n** #新建分区，然后一路回车  Command action     e   extended     p   primary partition (1-4)  **p**  #分区类型为主分区  Partition number (1-4, default 1):  #分区号，直接按**回车**  First sector (2048-496127, default 2048):   #指定分区的起始扇区，一般默认，按**回车**键即可。  Last sector, +sectors or +size{K,M,G} (2048-496127, default 496127):  Command (m for help): **p**  #用p打印出已建好的分区列表  Disk /dev/vdb: 2 TiB, 2199023255552 bytes, 4294967296 sectors Units: sectors of 1 \* 512 = 512 bytes Sector size (logical/physical): 512 bytes / 512 bytes I/O size (minimum/optimal): 512 bytes / 512 bytes Disklabel type: dos Disk identifier: 0x1b16f096  Device Boot Start End Sectors Size Id Type /dev/vdb1 2048 4294967294 4294965247 2T 83 Linux  Command (m for help): **w**  #保存  The partition table has been altered!  Calling ioctl() to re-read partition table.  Syncing disks. |

4.4.2 格式化

执行**mkfs.ext4 /dev/vdb1命令，对上文中进行分区的磁盘进行格式化。**



4.4.3 挂载

按照以下命令进行数据盘挂载。

|  |
| --- |
| [root@node ~]# **vim /etc/systemd/system/mnt-data.mount**  [Unit] Description=Mount Extra Data  [Mount] What=/dev/vdb1 Where=/mnt/data Type=ext4  [Install] WantedBy=multi-user.target  保存退出  进行目录挂载  **systemctl start mnt-data.mount**  添加开机自动挂载目录  **systemctl enable mnt-data.mount** |

4.5 NFS搭建

如4.1节中的服务器配置表所示，整个平台共有两个k8s集群，每个k8s集群使用一个NFS，NFS配置在k8s master节点上。完成磁盘挂载后，即可进行NFS的搭建。

4.5.1 搭建NFS Server

使用k8s master节点作为NFS Server，登录到该节点上，然后修改/etc/exports文件。

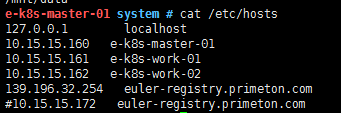
|  |
| --- |
| ＃**vim /etc/exports**  添加：  /mnt/data     \*(rw,async,no\_subtree\_check,no\_root\_squash)  # **systemctl enable nfs-server.service**  # **systemctl start nfs-server.service**  查看挂载目录是否可以正常访问：  # **showmount -e 192.168.20.1** #该IP为当前节点的IP  Export list for 192.168.20.1:  /mnt/data 192.168.20.0/24 |

4.5.2 搭建NFS Client

将k8s集群中的每一个work node作为NFS Client。

|  |
| --- |
| # **mkdir -p /mnt/euler\_storage/pod\_data\_volume/** # **vim /etc/systemd/system/mnt-euler\_storage-pod\_data\_volume.mount**  [Unit] Description=Mount nfs data  [Mount] What=[e-k8s-master-01:/mnt/data](http://e-k8s-master-01/mnt/data) Where=/mnt/euler\_storage/pod\_data\_volume Type=nfs  [Install] WantedBy=multi-user.target  保存退出  设置开机自启动 # **systemctl enable mnt-euler\_storage-pod\_data\_volume.mount**  启动nfs 客户端服务并 # **systemctl start mnt-euler\_storage-pod\_data\_volume.mount** |

注：红字部分为master节点的IP或者master节点IP的解析名称，可在master节点上通过cat /etc/hosts命令查看。



查看是否挂载成功

|  |
| --- |
| # **df -h**  Filesystem                            Size  Used Avail Use% Mounted on  devtmpfs                              2.0G     0  2.0G   0% /dev  tmpfs                                 2.0G     0  2.0G   0% /dev/shm  tmpfs                                 2.0G  668K  2.0G   1% /run  tmpfs                                 2.0G     0  2.0G   0% /sys/fs/cgroup  /dev/vda9                              95G  7.5G   84G   9% /  /dev/vda4                             985M  611M  324M  66% /usr  /dev/vda1                             128M   67M   62M  52% /boot  tmpfs                                 2.0G     0  2.0G   0% /media  tmpfs                                 2.0G     0  2.0G   0% /tmp  /dev/vda6                             108M   80K   99M   1% /usr/share/oem  192.168.20.1:/mnt/data          493G   19G  449G   5% /home/nfs |

看到上面红字部分即挂载成功。

4.6 准备Docker仓库

4.6.1 搭建Docker仓库

1. 选择一台操作系统为CoreOS的独立节点进行Docker仓库搭建。如果机器节点不能联网，可通过**docker load -i registry-2.tar**命令，将tar镜像归档导入到节点的Docker中。
2. 编辑Docker仓库启动脚本

|  |
| --- |
| #  **mkdir -p /mnt/euler\_storage/euler-registry**  #  **vim /mnt/euler\_storage/euler-registry/start.sh**  docker run -d -p 80:5000 --restart always --name registry -v /mnt/euler\_storage/[euler-registry:/var/lib/registry](http://euler-registry/var/lib/registry) registry:2  保存并退出 |

（3）启动Docker仓库镜像

|  |
| --- |
| #  **chmod 744 /mnt/euler\_storage/euler-registry/start.sh**  #  **/mnt/euler\_storage/euler-registry/start.sh** |

（4）验证Docker仓库是否启动成功

在服务器节点上执行curl http://euler-registry.primeton.com/v2/\_catalog命令，能够查看仓库中有哪些镜像。

例如：



4.6.2 镜像和容器的导入导出

Docker容器在启动的时候，如果本地没有相应的镜像，将会从Docker仓库进行拉取，所以需要将机器节点连接到互联网。如果不能联网，可通过docker命令进行镜像和容器的导入导出。

1. 导出

用户可通过**docker export**命令将容器导出为文件，也可使用**docker save**命令，将本地镜像导出为文件。

1. 导入

用户可以通过docker import命令将文件导入为docker容器，也可使用docker load将文件导入为镜像

1. 平台需要上传的Docker镜像列表（双击可打开）



4.6.3 设置仓库访问安全认证

因为Docker仓库没有使用HTTPS协议，所以Docker认为访问该仓库是不安全的，Docker如果需要从非SSL源管理镜像，需要配置Docker的insecury-registry参数。

k8s集群中，我们的初始化文件都已经加上了该参数，下面是针对非k8s集群机器的参数配置进行说明。

|  |  |
| --- | --- |
| |  | | --- | | **/etc/hosts**  # Append，在没有DNS服务的情况下进行配置  11.11.11.11 euler-registry.primeton.com | |

|  |
| --- |
| **CoreOS**  **cp -f /usr/lib/systemd/system/docker.service /etc/systemd/system/**    **vim /etc/systemd/system/docker.service**    # 在ExecStart后面追加 --insecure-registry euler-registry.primeton.com    # restart docker.service  **systemctl enable docker.service**  **systemctl daemon-reload**  **systemctl restart docker.service** |
| **Ubuntu**  **vim /etc/default/docker**    # DOCKER\_OPTS 追加配置参数 --insecure-registry euler-registry.primeton.com    **service docker restart** |

|  |  |
| --- | --- |
| |  | | --- | | **CentOS**  **vim /etc/sysconfig/docker**    # OPTIONS 追加配置参数 --insecure-registry euler-registry.primeton.com      # restart docker.service  **systemctl enable docker.service**  **systemctl daemon-reload**  **systemctl restart docker.service** | |

4.7 Kubernetes集群安装

4.7.1 安装flannel组件

1. 在Master节点和WorkNode节点上增加以下配置文件。

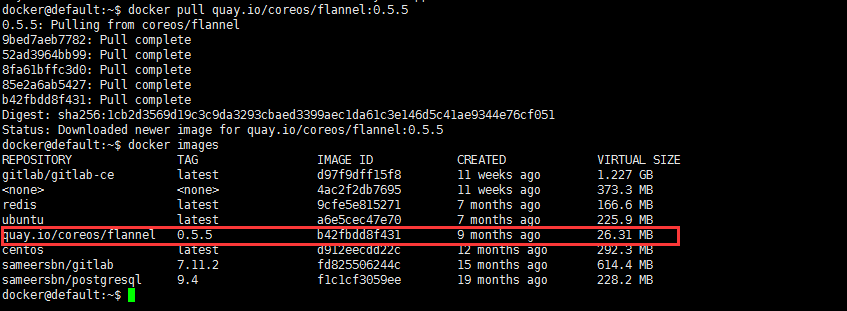
|  |
| --- |
| **/etc/systemd/system/early-docker.service.d/10-opts.conf**  [Service]  Environment="DOCKER\_OPTS=--insecure-registry euler-registry.primeton.com" |

|  |
| --- |
| **/etc/systemd/system/docker.service.d/10-opts.conf**   [Service]  Environment="DOCKER\_OPTS=--insecure-registry euler-registry.primeton.com" |

|  |
| --- |
| **/etc/systemd/system/flanneld.service.d/10-opts.conf**  [Service]  Environment="FLANNEL\_VER=latest"  Environment="FLANNEL\_IMG=euler-registry.primeton.com/quay.io/coreos/flannel" |

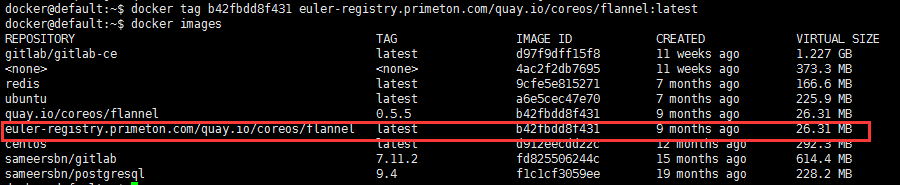
1. 下载flannel的Docker镜像

执行**docker pull quay.io/coreos/flannel:0.5.5**命令，将镜像下载到本地。



1. 上传flannel镜像到搭建好的Docker仓库中

执行**docker tag b42fbdd8f431 quay.io/coreos/flannel:latest**命令，对flannel重命名并且打上latest的tag。



执行**docker push euler-registry.primeton.com/quay.io/coreos/flannel**命令，将打好tag的flannel镜像上传到搭建好的Docker仓库中。

完成上传后，执行**curl** [**http://euler-registry.primeton.com/v2/\_catalog**命令，可看到Docker仓库中的flannel镜像。](http://euler-registry.primeton.com/v2/_catalog命令，可看到Docker仓库中的flannel镜像。)



1. 停止docker服务

执行 **systemctl stop docker.service**命令停止docker服务

1. 在Master节点上将flannel服务注册到etcd

在Master节点上执行以下命令，将flannel服务注册到etcd

|  |
| --- |
| **etcdctl set /coreos.com/network/config '{"Network":"210.100.0.0/8", "Backend": {"Type":"udp", "Port":7890}}'** |

1. 启动所有节点上的flannel服务

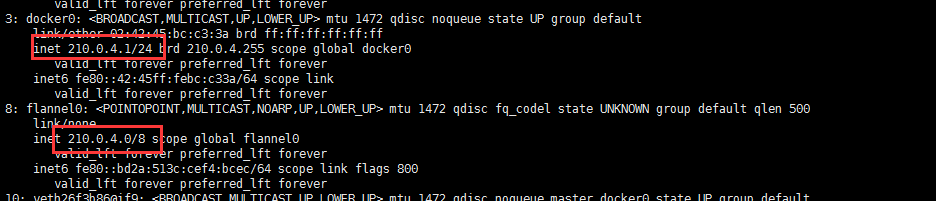
执行**systemctl start flanneld.service**命令启动flannel服务

1. 将所有节点的Docker网络服务修改为flannel

执行以下命令将节点上的Docker网络服务修改为flannel服务

|  |
| --- |
| **rm /var/run/docker.pid**  **source /run/flannel/subnet.env**  **ifconfig docker0 ${FLANNEL\_SUBNET}** |

执行上述命令后，通过**ip a**命令可以看到docker0和flannel0的ip如下图所示。



1. 重新启动Docker和early-docker服务

|  |
| --- |
| **systemctl restart docker**  **systemctl restart early-docker** |

4.7.2 安装Kubernetes管理节点

1. 下载Kubernetes，进行解压安装

|  |
| --- |
| **ssh**[**root@10.15.15.160**](mailto:root@192.168.20.1) **mkdir -p /opt/bin            #/opt/bin这个目录已经存在PATH环境变量中 cd /opt/bin curl -OL**[**https://github.com/kubernetes/kubernetes/releases/download/v1.2.6/kubernetes.tar.gz**](https://github.com/kubernetes/kubernetes/releases/download/v1.2.0/kubernetes.tar.gz)**tar xvf kubernetes.tar.gz tar xvf kubernetes/server/kubernetes-server-linux-amd64.tar.gz cp -rvf kubernetes/server/bin/{kubelet,kube-apiserver,kube-proxy,kubectl,kube-scheduler,kube-controller-manager} /opt/bin chmod a+x /opt/bin/kube\*** |

1. 下载安装kubu-dns

kube-dns是Kubernetes1.3版本中的组件，所以除了1.2.6版本之外，还需要下载1.3.0版本。然后从1.3.0版本中把kube-dns复制到步骤（1）的/opr/bin目录下。

|  |
| --- |
| **ssh**[**root@192.168.20.1**](mailto:root@192.168.20.1) **cd /opt curl -OL**[**https://github.com/kubernetes/kubernetes/releases/download/v1.3.0/kubernetes.tar.gz**](https://github.com/kubernetes/kubernetes/releases/download/v1.2.0/kubernetes.tar.gz)**tar xvf kubernetes.tar.gz tar xvf kubernetes/server/kubernetes-server-linux-amd64.tar.gz cp -rvf kubernetes/server/bin/kube-dns /opt/bin chmod a+x /opt/bin/kube\*** |

1. 配置systemd服务

|  |
| --- |
| **kubernete apiservere**  **vim /etc/systemd/system/kube-apiserver.service**  [Unit]  Description=Kubernetes API Server  Documentation=https://github.com/GoogleCloudPlatform/kubernetes  Requires=etcd2.service  After=etcd2.service  [Service]  ExecStart=/opt/bin/kube-apiserver \          --service-account-key-file=/opt/bin/kube-serviceaccount.key \          --service-account-lookup=false \          --admission-control=NamespaceLifecycle,NamespaceAutoProvision,LimitRanger,SecurityContextDeny,ServiceAccount,ResourceQuota \          --runtime-config=api/v1,extensions/v1beta1=true,extensions/v1beta1/thirdpartyresources=true \          --allow-privileged=true \          --insecure-bind-address=0.0.0.0 \          --insecure-port=8080 \          --kubelet-https=true \          --secure-port=6443 \          --service-cluster-ip-range=200.110.0.0/16 \          --etcd-servers=http://127.0.0.1:2379 \          --bind-address=0.0.0.0 \          --logtostderr=true \          --cors-allowed-origins='.\*' \          --service-node-port-range='30002-39998' \          --v=0 \          --allow-privileged=true  Restart=always  RestartSec=10  [Install]  WantedBy=multi-user.target |

|  |  |
| --- | --- |
| |  | | --- | | **kubernete scheduler**  **vim  /etc/systemd/system/kube-scheduler.service**  [Unit]  Description=Kubernetes Scheduler  Documentation=https://github.com/kubernetes/kubernetes  After=kube-apiserver.service  Requires=kube-apiserver.service  [Service]  ExecStart=/opt/bin/kube-scheduler \       --master=127.0.0.1:8080 \       --v=0  Restart=always  RestartSec=10  [Install]  WantedBy=multi-user.target | |

|  |  |
| --- | --- |
| |  | | --- | | **kubernete controller manager**  **vim  /etc/systemd/system/kube-controller-manager.service**  [Unit]  Description=Kubernetes Controller Manager  Documentation=https://github.com/kubernetes/kubernetes  After=kube-apiserver.service  Requires=kube-apiserver.service  [Service]  ExecStart=/opt/bin/kube-controller-manager \       --service-account-private-key-file=/opt/bin/kube-serviceaccount.key \       --master=127.0.0.1:8080 \       --logtostderr=true \       --v=0  Restart=always  RestartSec=10  [Install]  WantedBy=multi-user.target | |

|  |
| --- |
| **Kubernetes dns**  **vim  /etc/systemd/system/kube-dns.service**  [Unit]  Description=Kubernetes dns  Documentation=https://github.com/kubernetes/kubernetes  [Service]  ExecStart=/opt/bin/kube-dns \       --kube-master-url=http://127.0.0.1:8080 \       --domain=clusteruser \       --healthz-port=8081 \       --logtostderr=true  Restart=always  RestartSec=10  [Install]  WantedBy=multi-user.target |

1. 生成kube-apiserver需要使用的验证秘钥

执行以下命令，生成kube-apiserver文件中红字部分的验证秘钥。

|  |
| --- |
| **openssl genrsa -out /opt/bin/kube-serviceaccount.key 2048 2>/dev/null** |

执行完命令之后将会生成/opt/bin/kube-serviceaccount.key文件，内容如下图所示。



1. 启动Kubernetes服务

|  |  |
| --- | --- |
| |  | | --- | | 开机启动  **systemctl enable kube-controller-manager.service**  **systemctl enable kube-scheduler.service**  **systemctl enable kube-apiserver.service**  启动服务  **systemctl start kube-controller-manager.service**  **systemctl start kube-scheduler.service**  **systemctl start kube-apiserver.service** | |

4.7.3 安装Kubernetes工作节点

1. 下载Kubernetes，进行解压安装

|  |
| --- |
| **ssh**[**root@192.168.20.2**](mailto:root@192.168.20.1) **mkdir -p /opt/bin            #/opt/bin这个目录已经存在PATH环境变量中 cd /opt/bin curl -OL**[**https://github.com/kubernetes/kubernetes/releases/download/v1.2.6/kubernetes.tar.gz**](https://github.com/kubernetes/kubernetes/releases/download/v1.2.0/kubernetes.tar.gz)**tar xvf kubernetes.tar.gz tar xvf kubernetes/server/kubernetes-server-linux-amd64.tar.gz cp -rvf kubernetes/server/bin/{kubelet,kube-apiserver,kube-proxy,kubectl,kube-scheduler,kube-controller-manager} /opt/bin chmod a+x /opt/bin/kube\*** |

1. 配置systemd服务

|  |  |
| --- | --- |
| |  | | --- | | **kuberneter proxy**  **vim /etc/systemd/system/kube-proxy.service**  [Unit]  Description=Kubernetes Proxy  Documentation=https://github.com/kubernetes/kubernetes  [Service]  ExecStart=/opt/bin/kube-proxy  \       --master=http://e-k8s-master-01:8080 \       --logtostderr=true \       --v=0  Restart=always  RestartSec=10  [Install]  WantedBy=multi-user.target | |

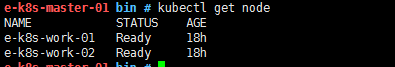
|  |  |
| --- | --- |
| |  | | --- | | **kuberneter kubelet**  **vim /etc/systemd/system/kube-kubelet.service**  [Unit]  Description=Kubernetes Kubelet  Documentation=https://github.com/kubernetes/kubernetes  [Service]  ExecStart=/opt/bin/kubelet \          --address=0.0.0.0 \          --port=10250 \          --api-servers=http://u-k8s-master-01:8080 \          --pod-infra-container-image=docker.io/kubernetes/pause:latest \          --logtostderr=true \          --cadvisor-port=4194 \          --healthz-bind-address=0.0.0.0 \          --healthz-port=10248 \          --max-pods=60 \          --cluster\_dns=210.0.90.1 \ #与Master的docker0 IP相同          --cluster\_domain=clusteruser \          --v=0 \          --allow-privileged=true  Restart=always  RestartSec=10  [Install]  WantedBy=multi-user.target | |

1. 启动Kubernetes服务

|  |  |
| --- | --- |
| |  | | --- | | 开机启动  **systemctl enable kube-proxyservice**  **systemctl enable kube-kubelet.service**  启动服务  **systemctl start kube-proxy.service**  **systemctl start kube-kubelet.service** | |

4.7.4 验证Kubernetes集群

在Master节点上执行**kubectl get node**命令，能显示出当前k8s集群中有多少个WorkNode。



4.8 泛域名配置

泛域名配置，可以让我们访问k8s部署好的资源时，不论底层有多少个k8s计算节点，都不需要关心具体的IP，而是直接可以使用域名+端口的方式进行访问。

4.8.1 阿里云环境配置

在阿里云环境上进行泛域名配置可以参考以下两个网页。

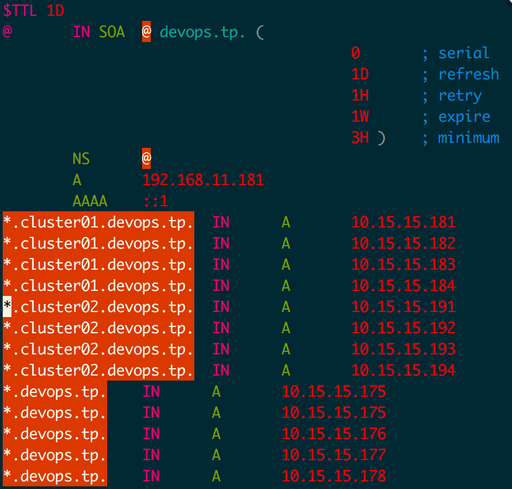
（1）阿里云域名申请：<https://wanwang.aliyun.com/domain/>

（2）阿里云的云解析DNS能力：<https://wanwang.aliyun.com/domain/dns/>

4.8.2 私有云环境

（1）在私有云环境中使用DNS泛解析能力，需要在DNS服务器上添加如下配置。

**vim /var/named/chroot/var/named/devops.tp**



注：\*.devops.tp是制定泛域名，后面对应的机器是泛域名映射到的机器

1. 完成上述配置后，重启dns服务。

4.9 OpenResty安装

Openresty实现了微服务的动态路由，对外屏蔽k8s server的nodeport，即可以让我们访问部署好的资源时，不再需要输入端口。

4.9.1 复制目录

将安装目录复制到K8s master结点。因为CoreOS不支持Xshell的上传工具，建议将安装文件上传至CentOS机器，然后使用scp或者搭建ftp，http服务器。

拷贝文件到~/euler-sem-openresty目录

4.9.2 编译并上传镜像

在master结点上执行以下命令，进行镜像编译

|  |
| --- |
| **cd ~/euler-sem-openresty/image**  **./build-push-images.sh** |

完成编译后将编译的镜像上传到docker中。

4.9.3 创建相关服务

在K8s集群中创建openresty及其相关服务。

在master结点上执行以下命令

|  |
| --- |
| **cd ~/euler-sem-openresty/yaml**  **kubectl create namespace euler-system**    **kubectl --namespace=euler-system create configmap sem-redis-configmap --from-file redis/configmap**  **kubectl --namespace=euler-system create -f redis/master-svc.yaml**  **kubectl --namespace=euler-system create -f redis/master-deployment.yaml**  **kubectl --namespace=euler-system create -f redis/slave-svc.yaml**  **kubectl --namespace=euler-system create -f redis/slave-deployment.yaml**    **kubectl --namespace=euler-system create configmap sem-openresty-configmap --from-file openresty/configmap**  **kubectl --namespace=euler-system create -f openresty/ds.yaml** |

4.9.4 查看服务启动

通过以下命令可产看服务是否已经正常启动。

|  |
| --- |
| **kubectl --namespace=euler-system get ds**  **kubectl --namespace=euler-system get deployment**  **kubectl --namespace=euler-system get rs**  **kubectl --namespace=euler-system get pod** |

4.9.5 修改配置

服务启动后需要配置用户k8s集群相关信息及反向代理服务。

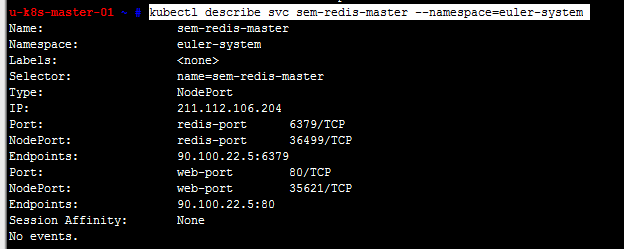
查看数据库sem\_cluster表中的CLUSTER\_CONFIG 字段，如下。

|  |
| --- |
| { "kubernetes.master":"http://10.25.105.109:8080"  , "public.domain.suffix":"cluster01.hzqzinfo.com"  , "sem.openresty":{"enabled":true, "port.names":"web", "redis.master.host":"139.196.11.126", "redis.master.port":36499, "jedis.connect.timeout":30000, "jedis.so.timeout":30000}  , "service.create.timeout":"300"  } |

Kubernetes.master：用户k8s集群的master节点的服务地址。

Redis.master.host：用户k8s其中一个workNode的ip地址,redis.master.port的端口可以在用户k8s集群的master节点使用查询获取。

|  |
| --- |
| **kubectl describe svc sem-redis-master --namespace=euler-system** |



如图redis-port的nodeport端口号为36499

1. 第三方依赖产品安装

5.1 Etcd集群搭建

Etcd集群注册中心是所有领域系统依赖的服务注册中心，我们采用一主两从的方案搭建etcd集群。在新一代平台中我们通过容器搭建Etcd集群，并用k8s进行管理。

5.1.1 创建目录

登录到k8s master节点上，创建etcd配置文件目录

|  |
| --- |
| **mkdir -p /mnt/data/products/etcd** |

5.1.2 添加yml文件

|  |
| --- |
| **etcd0-pod.yml**  **vim /mnt/data/products/etcd/etcd0-pod.yml**  apiVersion: v1  kind: Pod  metadata:    name: etcd0    labels:      product: "etcd0"    namespace: "kube-system"  spec:      containers:        - name: "etcd0"          image: "euler-registry.primeton.com/etcd3:1.0.0"          imagePullPolicy: Always          args:            - "--name=etcd0"            - "--advertise-client-urls=http://200.11.98.19:2379"            - "--listen-client-urls=http://0.0.0.0:2379"            - "--listen-peer-urls=http://0.0.0.0:2380"            - "--initial-advertise-peer-urls=http://200.110.98.19:2380"            - "--initial-cluster=etcd0=http://200.110.98.19:2380,etcd1=http://200.110.98.20:2380,etcd2=http://200.110.98.21:2380"            - "--initial-cluster-state=new"            - "--initial-cluster-token=cluster0"            - "--data-dir=/var/lib/etcd/data"          ports:            - containerPort: 2379              protocol: "TCP"            - containerPort: 2380              protocol: "TCP"          resources:            limits:              cpu: "1000m"              memory: "1Gi"          volumeMounts:            - name: "etcd-data"              mountPath: /var/lib/etcd/data      volumes:        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/etcd/data0"          name: "etcd-data" |

|  |
| --- |
| **etcd0-svc.yml**  **vim /mnt/data/products/etcd/etcd0-svc.yml**  apiVersion: "v1"  kind: "Service"  metadata:    annotations: {}    labels:      product: "etcd0"    name: "etcd0"    namespace: "kube-system"  spec:    deprecatedPublicIPs: []    externalIPs: []    clusterIP: 200.110.98.19    ports:    - name: port1      port: 2379      nodePort: 32379      targetPort: 2379    - name: port2      port: 2380      nodePort: 32380      targetPort: 2380    selector:      product: "etcd0"    type: "NodePort" |

|  |
| --- |
| **etcd1-pod.yml**  **vim /mnt/data/products/etcd/etcd1-pod.yml**  apiVersion: v1  kind: Pod  metadata:    name: etcd1    labels:      product: "etcd1"    namespace: "kube-system"  spec:      containers:        - name: "etcd1"          image: "euler-registry.primeton.com/etcd3:1.0.0"          imagePullPolicy: Always          args:            - "--name=etcd1"            - "--advertise-client-urls=http://200.110.98.20:2379"            - "--listen-client-urls=http://0.0.0.0:2379"            - "--listen-peer-urls=http://0.0.0.0:2380"            - "--initial-advertise-peer-urls=http://200.110.98.19:2380"            - "--initial-cluster=etcd0=http://200.110.98.19:2380,etcd1=http://200.110.98.20:2380,etcd2=http://200.110.98.21:2380"            - "--initial-cluster-state=new"            - "--initial-cluster-token=cluster0"            - "--data-dir=/var/lib/etcd/data"          ports:            - containerPort: 2379              protocol: "TCP"            - containerPort: 2380              protocol: "TCP"          resources:            limits:              cpu: "1000m"              memory: "1Gi"          volumeMounts:            - name: "etcd-data"              mountPath: /var/lib/etcd/data      volumes:        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/etcd/data1"          name: "etcd-data" |

|  |
| --- |
| **etcd1-svc.yml**  **vim /mnt/data/products/etcd/etcd1-svc.yml**  apiVersion: "v1"  kind: "Service"  metadata:    annotations: {}    labels:      product: "etcd1"    name: "etcd1"    namespace: "kube-system"  spec:    deprecatedPublicIPs: []    externalIPs: []    clusterIP: 200.110.98.20    ports:    - name: port1      port: 2379      nodePort: 33379      targetPort: 2379    - name: port2      port: 2380      nodePort: 33380      targetPort: 2380    selector:      product: "etcd1"    type: "NodePort" |

|  |
| --- |
| **etcd2-pod.yml**  **vim /mnt/data/products/etcd/etcd2-pod.yml**  apiVersion: v1  kind: Pod  metadata:    name: etcd2    labels:      product: "etcd2"    namespace: "kube-system"  spec:      containers:        - name: "etcd2"          image: "euler-registry.primeton.com/etcd3:1.0.0"          imagePullPolicy: Always          args:            - "--name=etcd2"            - "--advertise-client-urls=http://200.110.98.21:2379"            - "--listen-client-urls=http://0.0.0.0:2379"            - "--listen-peer-urls=http://0.0.0.0:2380"            - "--initial-advertise-peer-urls=http://200.110.98.19:2380"            - "--initial-cluster=etcd0=http://200.110.98.19:2380,etcd1=http://200.110.98.20:2380,etcd2=http://200.110.98.21:2380"            - "--initial-cluster-state=new"            - "--initial-cluster-token=cluster0"            - "--data-dir=/var/lib/etcd/data"          ports:            - containerPort: 2379              protocol: "TCP"            - containerPort: 2380              protocol: "TCP"          resources:            limits:              cpu: "1000m"              memory: "1Gi"          volumeMounts:            - name: "etcd-data"              mountPath: /var/lib/etcd/data      volumes:        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/etcd/data2"          name: "etcd-data" |

|  |
| --- |
| **etcd2-svc.yml**  **vim /mnt/data/products/etcd/etcd2-svc.yml**  apiVersion: "v1"  kind: "Service"  metadata:    annotations: {}    labels:      product: "etcd2"    name: "etcd2"    namespace: "kube-system"  spec:    deprecatedPublicIPs: []    externalIPs: []    clusterIP: 200.110.98.21    ports:    - name: port1      port: 2379      nodePort: 34379      targetPort: 2379    - name: port2      port: 2380      nodePort: 34380      targetPort: 2380    selector:      product: "etcd2"    type: "NodePort" |

注：yml文件中的clusterIP，前两个数字需要参照k8s集群的clusterIP进行修改。

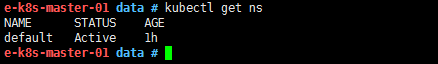
k8s集群的clusterIP可以通过**kubectl get svc --all-namespaces**命令进行查看。



5.1.3 创建k8s namespace

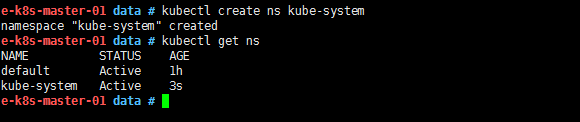
从上文中可以看到，部署etcd使用的k8s namespace是kube-system，在启动etcd之前要保证需要使用的namespace已经创建。通过以下命令能查看目前k8s集群中的namespace。

|  |
| --- |
| **kubectl get ns** |



如果需要使用的namespace还不存在，可通过以下命令进行创建。

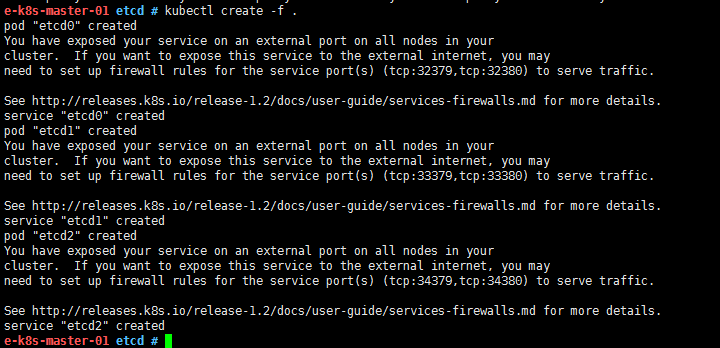
|  |
| --- |
| **kubectl create ns kube-system** |



5.1.4 启动etcd

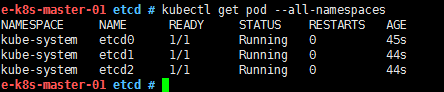
在/mnt/data/products/etcd/目录下执行以下命令，k8s集群将会在worknode节点上创建pod，并且启动pod中的etcd

|  |
| --- |
| **kubectl create -f .** |



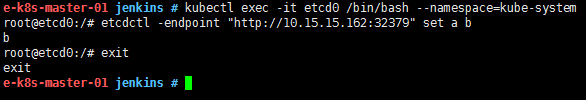
执行完上述命令后，可通过以下命令查看pod是否已经成功创建并且运行。

|  |
| --- |
| **kubectl get pod --all-namespaces** |



5.1.5 验证etcd集群

|  |
| --- |
| #进入etcdo的容器中  **kubectl exec -it etcd0 /bin/bash --namespace=kube-system**  #执行以下命令验证etcd是否搭建成功，访问的url为worknodeIP：nodeport  **etcdctl -endpoints “http://10.15.15.162:32379” set a b** |



5.2 安装jenkins

5.2.1 添加yml文件

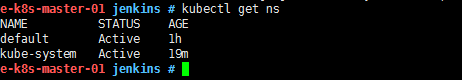
在/mnt/data/products目录下创建名为jenkins的文件夹，然后在文件夹中新增以下yml文件。

|  |
| --- |
| **jenkins-pod.yml**  **vim /mnt/data/products/jenkins/jenkins-pod.yml**  apiVersion: "v1"  kind: "ReplicationController"  #kind: "pod"  metadata:    annotations: {}    labels:      product: "jenkins"    name: "jenkins"    namespace: "default-euler"  spec:    replicas: 1    selector:      product: "jenkins"    template:      metadata:        annotations: {}        labels:          product: "jenkins"      spec:        containers:        - args: []          command: []          image: "euler-registry.primeton.com/jenkins:1.1.0"          name: "jenkins"          ports:          - containerPort: 8080          env:            - name: "REGISTRY\_URL"              value: "euler-registry.primeton.com"            - name: "REGISTRY\_HOST"              value: "10.15.15.163"          securityContext:            privileged: true          resources:            limits:              cpu: "1500m"              memory: "2Gi"            requests:              cpu: "1000m"              memory: "1Gi"          volumeMounts:          - mountPath: "/jenkins"            name: "data-volume"          - mountPath: "/root/.m2"            name: "maven-volume"        imagePullSecrets: []        nodeSelector: {}        volumes:        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/default-euler/jenkins/jenkins\_home"          name: "data-volume"        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/default-euler/jenkins/maven"          name: "maven-volume" |
| **jenkins-svc.yml**  **vim /mnt/data/products/jenkins/jenkins-svc.yml**  apiVersion: "v1"  kind: "Service"  metadata:    annotations: {}    labels:      product: "jenkins"    name: "jenkins"    namespace: "default-euler"  spec:    deprecatedPublicIPs: []    externalIPs: []    clusterIP: 200.110.98.25    ports:    - port: 8080      nodePort: 38080      targetPort: 8080    selector:      product: "jenkins"    type: "NodePort" |

5.2.2 创建k8s namespace

从上文中可以看到，部署jenkins使用的k8s namespace是default-euler，在启动jenkins之前要保证需要使用的namespace已经创建。通过以下命令能查看目前k8s集群中的namespace。

|  |
| --- |
| **kubectl get ns** |



如果需要使用的namespace还不存在，可通过以下命令进行创建。

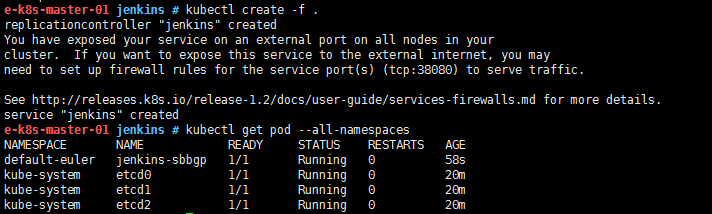
|  |
| --- |
| **kubectl create ns default-euler** |

5.2.3 启动jenkins

在启动jenkins之前需要保证在搭建的Docker仓库中已经有jenkins-pod.yml文件中配置的jenkins镜像。

在/mnt/data/products/jenkins目录下执行以下命令，创建pod，并且启动jenkins。

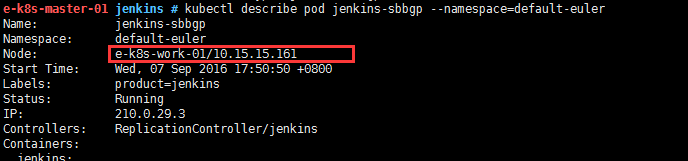
|  |
| --- |
| **kubectl create -f .** |



5.2.4 验证jenkins

通过下列命令，查看jenkins被创建到了哪一个工作节点上。

|  |
| --- |
| **kubectl describe pod jenkins-sbbgp --namespace=default-euler** |



然后访问http://10.15.15.161:38080，将会跳转到jenkins的登录页面。



Jenkins默认用户名密码为admin/admin

5.3 安装Nexus

5.3.1 添加yml文件

在/mnt/data/products目录下创建名为nexus的文件夹，然后在文件夹中新增以下yml文件。

|  |
| --- |
| **nexus-pod.yml**  **vim /mnt/data/products/nexus/nexus-pod.yml**  apiVersion: "v1"  kind: "ReplicationController"  metadata:    annotations: {}    labels:      product: "nexus"    name: "nexus"    namespace: "default-euler"  spec:    replicas: 1    selector:      product: "nexus"    template:      metadata:        annotations: {}        labels:          product: "nexus"      spec:        containers:        - args: []          command: []          image: "euler-registry.primeton.com/sonatype/nexus3:latest"          name: "nexus"          ports:          - containerPort: 8081          resources:            limits:              cpu: "1500m"              memory: "2Gi"            requests:              cpu: "1000m"              memory: "1Gi"          volumeMounts:          - mountPath: "/nexus-data"            name: "data-volume"        imagePullSecrets: []        nodeSelector: {}        volumes:        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/default-euler/nexus/nexus-data"          name: "data-volume" |
| **nexus-svc.yml**  **vim /mnt/data/products/nexus/nexus-svc.yml**  apiVersion: "v1"  kind: "Service"  metadata:    annotations: {}    labels:      product: "nexus"    name: "nexus"    namespace: "default-euler"  spec:    deprecatedPublicIPs: []    externalIPs: []    clusterIP: 200.110.98.26    ports:    - port: 8081      nodePort: 38081      targetPort: 8081    selector:      product: "nexus"    type: "NodePort"    selector:      product: "nexus"    type: "NodePort" |

5.3.2 创建nexus-data目录

创建路径为/mnt/data/default-euler/nexus/nuxus-data的目录，然后进入到nexus目录之下，执行下列命令，修改nuxus-data目录的权限。

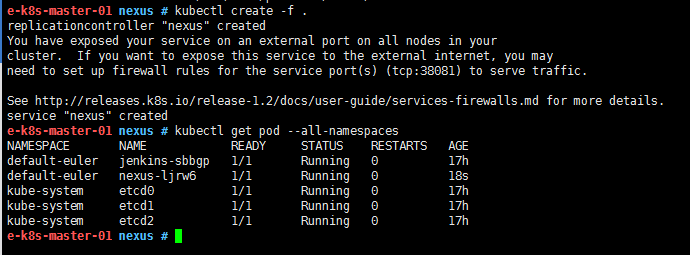
|  |
| --- |
| **chowm 200 -R nexus-data/** |

5.3.3 启动nexus

在启动nexus之前需要保证在搭建的Docker仓库中已经有nexus-pod.yml文件中配置的nexus镜像。

在/mnt/data/products/nexus目录下执行以下命令，创建pod，并且启动nexus。

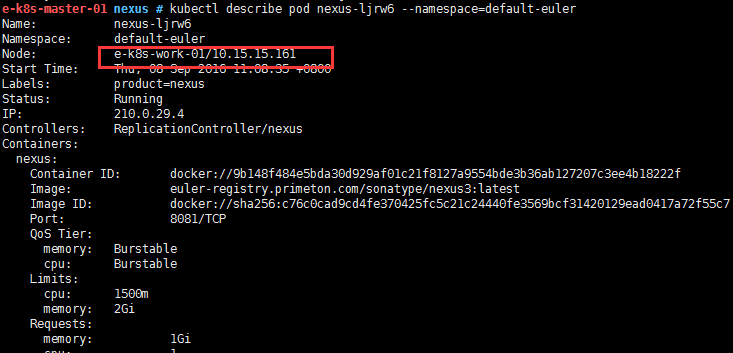
|  |
| --- |
| **kubectl create -f .** |



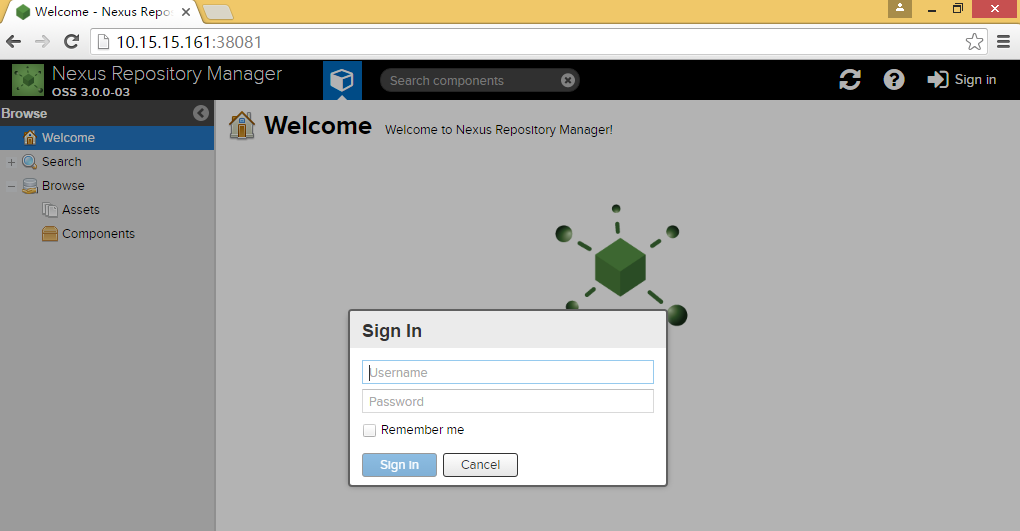
5.3.4 验证nuxus

通过下列命令，查看nuxus被创建到了哪一个工作节点上。

|  |
| --- |
| Kubectl describe pod nuxus-ljrw6 --namespace=default-euler |



然后访问http://10.15.15.161:38081，将会跳转到nuxus的登录页面。



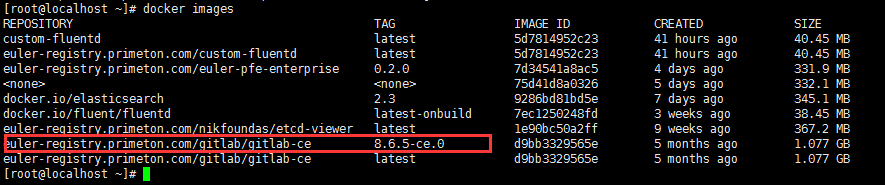
Jenkins默认用户名密码为admin/admin123

5.4 安装GitLab

我们将把GitLab安装在操作系统为CentOS的独立节点之上，用来给用户提供代码管理和版本控制的能力。

5.4.1 拉取Gitlab镜像

在CentOS机器节点上执行**docker pull euler-registry.primeton.com/gitlab-ce:8.6.5-ce.0**命令，进行镜像拉取。



5.4.2 添加Gitlab启动脚本

|  |
| --- |
| **vim /opt/gitlab/git-start.sh**  #!/bin/bash  docker run -d --hostname 10.15.15.164 -p 80:80 --name gitlab --restart always -v /opt/gitlab/gitlab:/etc/gitlab -v /opt/gitlab/logs:/var/log/g  itlab -v /opt/gitlab/data:/var/opt/gitlab euler-registry.primeton.com/gitlab/gitlab-ce:8.6.5-ce.0 |

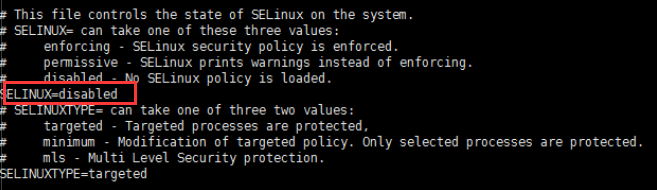
文件中蓝色部分为Gitlab所在机器节点的IP，如果Gitlab需要提供给外网访问，可将IP改为域名。

然后在gitlab目录下执行**chomd a+x git-start.sh**命令，修改git-start.sh权限，让启动脚本成为可执行文件。

5.4.3 关闭防火墙

为了能访问Gitlab，需要把机器节点的防火墙关闭，关闭操作如下。

执行**vim /etc/selinux/config**命令，修改config文件中的SELINUX参数。

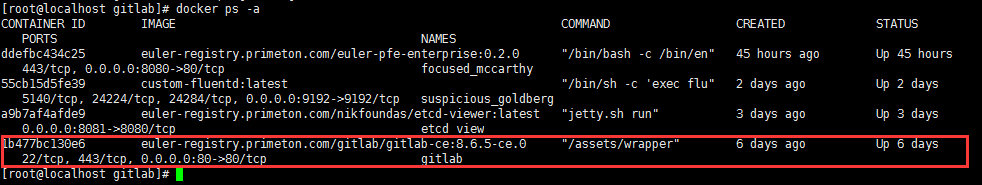


然后执行以下命令停止系统防火墙。

|  |
| --- |
| **systemctl stop firewalld**  **systemctl disale firewalld** |

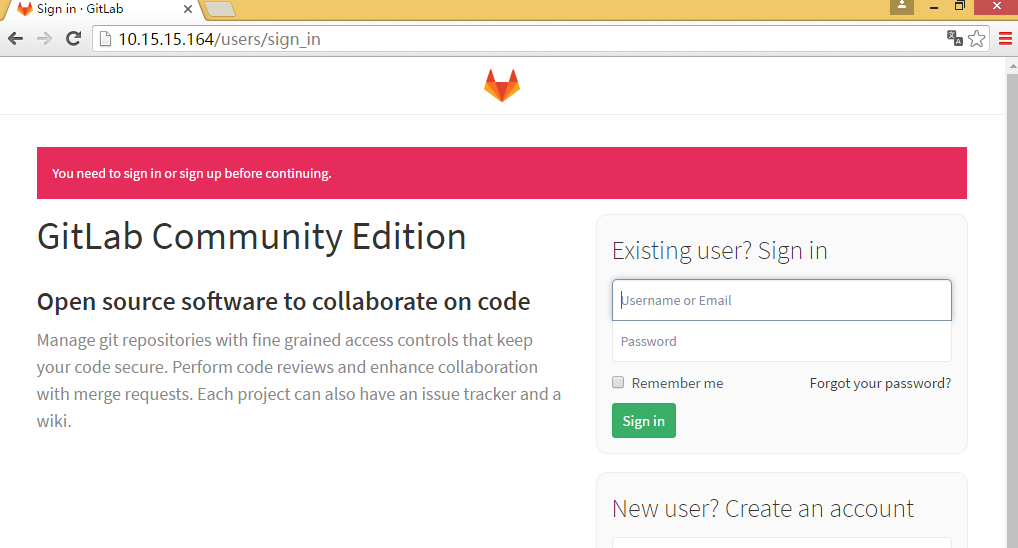
5.4.4 启动Gitlab

执行git-start.sh启动脚本，启动Gitlab。



5.4.5 验证Gitlab

在浏览器访问git-start.sh启动脚本中的设定的IP或域名，将会跳转到Gitlab的页面。



5.4.6 创建管理员

创建一个名为sysadmin的管理员账号，提供给The Platform平台的VCS领域系统使用。

5.5 ElasticSearch集群安装

在统一监控平台中，每个系统的日志最终会存储在ElasticSearch中。为了ElasticSearch的高可用，我们搭建的ElasticSearch集群共3个节点，使用版本为最新的es2.3.5。

5.5.1 安装ElasticSearch

在CentOS机器节点上，使用一下命令进行ElasticSearch的安装。

|  |
| --- |
| **yum -y install** [**https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/rpm/elasticsearch/2.3.5/elasticsearch-2.3.5.rpm**](https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/rpm/elasticsearch/2.3.5/elasticsearch-2.3.5.rpm) |

5.5.2 安装jdk1.8

ElasticSearch2.3.5需要使用jdk1.8，所以用户需要在CentOS机器节点上下载安装jdk1.8.

下载地址：<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

完成下载后使用下列命令进行jdk1.8的安装。

|  |
| --- |
| rpm -ivh jdk-8u101-linux-x64.rpm |

5.5.3 添加配置文件

在节点上添加如下配置文件。

|  |
| --- |
| 节点1  **vim /etc/elasticsearch/elasticsearch.yml**  #  # ---------------------------------- Cluster -----------------------------------  cluster.name: elasticsearch  #  # ------------------------------------ Node ------------------------------------    # Use a descriptive name for the node:  node.name: node-1  #  # ----------------------------------- Memory -----------------------------------  # Lock the memory on startup:  bootstrap.mlockall: true  #  # ---------------------------------- Network -----------------------------------  network.host: 0.0.0.0  # --------------------------------- Discovery ----------------------------------  discovery.zen.ping.multicast.enabled: false  discovery.zen.fd.ping\_timeout: 60s  discovery.zen.ping.timeout: 60s  discovery.zen.minimum\_master\_nodes: 2  discovery.zen.ping.unicast.hosts: ["192.168.20.38","192.168.20.39","192.168.20.40"] |

|  |
| --- |
| 节点2  **vim /etc/elasticsearch/elasticsearch.yml**  #  # ---------------------------------- Cluster -----------------------------------  cluster.name: elasticsearch  #  # ------------------------------------ Node ------------------------------------    # Use a descriptive name for the node:  node.name: node-2  #  # ----------------------------------- Memory -----------------------------------  # Lock the memory on startup:  bootstrap.mlockall: true  #  # ---------------------------------- Network -----------------------------------  network.host: 0.0.0.0  # --------------------------------- Discovery ----------------------------------  discovery.zen.ping.multicast.enabled: false  discovery.zen.fd.ping\_timeout: 60s  discovery.zen.ping.timeout: 60s  discovery.zen.minimum\_master\_nodes: 2  discovery.zen.ping.unicast.hosts: ["192.168.20.38","192.168.20.39","192.168.20.40"] |

|  |
| --- |
| 节点3  **vim /etc/elasticsearch/elasticsearch.yml**  #  # ---------------------------------- Cluster -----------------------------------  cluster.name: elasticsearch  #  # ------------------------------------ Node ------------------------------------    # Use a descriptive name for the node:  node.name: node-3  #  # ----------------------------------- Memory -----------------------------------  # Lock the memory on startup:  bootstrap.mlockall: true  #  # ---------------------------------- Network -----------------------------------  network.host: 0.0.0.0  # --------------------------------- Discovery ----------------------------------  discovery.zen.ping.multicast.enabled: false  discovery.zen.fd.ping\_timeout: 60s  discovery.zen.ping.timeout: 60s  discovery.zen.minimum\_master\_nodes: 2  discovery.zen.ping.unicast.hosts: ["192.168.20.38","192.168.20.39","192.168.20.40"] |

5.5.4 创建es数据路径

使用以下命令创建ElasticSearch的数据令，并且进行权限修改。

|  |
| --- |
| **mkdir /mnt/data/elasticsearch**  **chown elasticsearch:elasticsearch /mnt/data/elasticsearch** |

5.5.5 设置环境变量

为了优化性能，需要设置一下环境变量。

|  |
| --- |
| **vim /etc/sysconfig/elasticsearch**  ES\_STARTUP\_SLEEP\_TIME=5  ES\_HEAP\_SIZE=8g  MAX\_OPEN\_FILES=65535  MAX\_LOCKED\_MEMORY=unlimited |

5.5.6 添加service文件

|  |
| --- |
| **vim /usr/lib/systemd/system/elasticsearch.service**  [Unit]  Description=Elasticsearch  Documentation=http://www.elastic.co  Wants=network-online.target  After=network-online.target      [Service]  Environment=ES\_HOME=/usr/share/elasticsearch  Environment=CONF\_DIR=/etc/elasticsearch  Environment=DATA\_DIR=/mnt/data/elasticsearch/data  Environment=LOG\_DIR=/mnt/data/elasticsearch/log  Environment=PID\_DIR=/var/run/elasticsearch  EnvironmentFile=-/etc/sysconfig/elasticsearch      WorkingDirectory=/usr/share/elasticsearch      User=elasticsearch  Group=elasticsearch      ExecStartPre=/usr/share/elasticsearch/bin/elasticsearch-systemd-pre-exec      ExecStart=/usr/share/elasticsearch/bin/elasticsearch \                                                  -Des.pidfile=${PID\_DIR}/elasticsearch.pid \                                                  -Des.**default**.path.home=${ES\_HOME} \                                                  -Des.**default**.path.logs=${LOG\_DIR} \                                                  -Des.**default**.path.data=${DATA\_DIR} \                                                  -Des.**default**.path.conf=${CONF\_DIR}      StandardOutput=journal  StandardError=inherit  Restart=always  RestartSec=10      # Specifies the maximum file descriptor number that can be opened by **this** process  LimitNOFILE=65535      # Specifies the maximum number of bytes of memory that may be locked into RAM  # Set to "infinity" **if** you use the 'bootstrap.mlockall: true' option  # in elasticsearch.yml and 'MAX\_LOCKED\_MEMORY=unlimited' in /etc/sysconfig/elasticsearch  LimitMEMLOCK=infinity      # Disable timeout logic and wait until process is stopped  TimeoutStopSec=0      # SIGTERM signal is used to stop the Java process  KillSignal=SIGTERM      # Java process is never killed  SendSIGKILL=no      # When a JVM receives a SIGTERM signal it exits with code 143  SuccessExitStatus=143      [Install]  WantedBy=multi-user.target      # Built **for** Distribution: RPM-2.3.5 (rpm) |

5.5.7 启动服务

通过以下命令进行elasticsearch服务的启动和设置自启动。

|  |
| --- |
| **systemctl start elasticsearch**  #设置服务自启动  **systemctl enable elasticsearch** |

5.5.8 设置分词索引

不分词—CONTAINER\_NAME字段。

|  |
| --- |
| curl -XPUT '<http://10.15.15.112:9200/euler'>  -d '  {      "mappings" : {          "products" : {              "properties" : {                  "CONTAINER\_NAME" : {                      "type" : "string",                      "index" : "not\_analyzed"                  }              }          }      }  }' |

5.6 Influxdb安装

Influxdb用于存储采集到的性能数据，包括主机和容器的CPU、内存、网络、磁盘等数据。平台所使用的Influxcb版本号为0.13.0

5.6.1 安装Influxdb

在CentOS机器节点上使用以下命令进行Influxdb的安装。

|  |
| --- |
| **yum -y install**[**https://repos.influxdata.com/rhel/7/x86\_64/stable/influxdb-0.13.0.x86\_64.rpm**](https://repos.influxdata.com/rhel/7/x86_64/stable/influxdb-0.13.0.x86_64.rpm) |

5.6.2 添加配置文件

|  |
| --- |
| **vim /etc/influxdb.conf**  [meta]  dir="/mnt/data/influxdb/meta"      [data]  dir="/mnt/data/influxdb/data"  wal-dir="/mnt/data/influxdb/wal"  cache-max-memory-size=4194304000  cache-snapshot-memory-size=2097152000      [[udp]]  enabled = **true**  bind-address = ":8089"  database = ""  batch-size = 100000  batch-pending = 50  batch-timeout = "2s"  read-buffer = 8388608 |

5.6.3 优化udp os buffer size

使用以下命令查看udp os buffer size。

|  |
| --- |
| **sysctl net.core.rmem\_max** |

如果值小于8388308，可以修改sysctl.conf文件。

|  |
| --- |
| **vim /etc/sysctl.conf**  net.core.rmem\_max=8388608 |

也可通过以下命令进行修改。

|  |
| --- |
| **sysctl -w net.core.rmem\_max=8388608** |

5.6.4 赋予influxdb用户权限

|  |
| --- |
| **chown influxdb:influxdb /mnt/data/influxdb** |

5.6.5 添加service文件

|  |
| --- |
| **vim /etc/systemd/system/influxdb.service**  # If you modify **this**, please also make sure to edit init.sh      [Unit]  Description=InfluxDB is an open-source, distributed, time series database  Documentation=https://docs.influxdata.com/influxdb/  After=network.target      [Service]  User=influxdb  Group=influxdb  LimitNOFILE=65536  Environment='STDOUT=/dev/null'  Environment='STDERR=/var/log/influxdb/influxd.log'  EnvironmentFile=-/etc/**default**/influxdb  ExecStart=/bin/sh -c "/usr/bin/influxd -config /etc/influxdb/influxdb.conf ${INFLUXD\_OPTS} >>${STDOUT} 2>>${STDERR}"  #KillMode=control-group  #Restart=on-failure  Restart=always  RestartSec=10      [Install]  WantedBy=multi-user.target  Alias=influxd.service |

5.6.6 启动服务

|  |
| --- |
| **systemctl daemon-reload**  **systemctl start influxdb** |

5.7 Fluentd安装

5.7.1 安装fluentd

执行以下命令进行fluentd的安装。

|  |
| --- |
| **curl –L https://toolbelt.treasuredata.com/sh/install-redhat-td-agent2.sh | sh** |

5.7.2 安装相关插件

|  |
| --- |
| 安装fluent-plugin-elasticsearch  **/usr/sbin/td-agent-gem install fluent-plugin-elasticsearch**    安装fluentd type  **/usr/sbin/td-agent-gem install fluent-plugin-typecast**    安装secure-forward  **/usr/sbin/td-agent-gem install fluent-plugin-secure-forward** |

5.7.3 修改配置文件

在td-agent.conf文件中添加如下信息配置。

|  |
| --- |
| **vim /etc/td-agent/td-agent.conf**  <source>    type tcp    port 9192               //接收端口    tag theplatform.logs      //tag标签    format json  </source>    <match theplatform.logs>    type elasticsearch        //存储方式    host localhost           //es主机IP    port 9200    index\_name euler    type\_name products  </match> |

5.7.4 启动服务

|  |
| --- |
| 启动服务  **systemctl start td-agent**  设置开机启动  **systemctl enable td-agent**  查看服务状态  **systemctl status td-agent** |

5.8 Heapster安装

Heapster用作对Docker集群的监控，Heapster原生支持k8s和CoreOS中容器的性能数据采集，收集所有Kubernetes Node的数据，然后进行汇总，导入到第三方工具中（如Influxdb）。

Heapster可以以pod的方式运行在Kubernetes平台里，也可以单独以standalone的方式运行。

在The Platform平台中，因为有两个k8s集群，所以在每个k8s集群中通过pod的方式启动一个Heapster，并指向同一个Influxdb。

5.8.1 拉取镜像

|  |
| --- |
| **docker pull kubernetes/heapster:canary** |

5.8.2 添加配置文件

|  |
| --- |
| **vim /mnt/data/products/heapster/heapster-controller.yml**  apiVersion: v1  kind: ReplicationController  metadata:    labels:      k8s-app: heapster      name: heapster      version: v6    name: heapster    namespace: kube-system  spec:    replicas: 1    selector:      k8s-app: heapster      version: v6    template:      metadata:        labels:          k8s-app: heapster          version: v6      spec:        containers:        - name: heapster          image: euler-registry.primeton.com/kubernetes/heapster:canary          command:          - /heapster          - -metric\_resolution=20s          - --source=kubernetes:http://192.168.20.21:8080  #kubernetes apiserver的url          - --sink=influxdb:http://192.168.20.38:8086?db=theplatform #influxdb的地址。 |

|  |
| --- |
| **vim /mnt/data/products/heapster/heapster-service.yml**  apiVersion: v1  kind: Service  metadata:    labels:      kubernetes.io/cluster-service: 'true'      kubernetes.io/name: Heapster    name: heapster    namespace: kube-system  spec:    ports:    - port: 80      targetPort: 8082    selector:      k8s-app: heapster |

5.8.3 启动Heapster

|  |
| --- |
| **kubectl create -f /mnt/data/products/heapster/heapster-controller.yml**  **kubectl create -f /mnt/data/products/heapster/heapster-service.yml** |

5.9 日志采集器

日志采集器其实是一个进程，用来从journald中读取日志并发送到fluentd，然后fluentd再统一将日志发给ElasticSearch。所以日志采集器的本质是一个agent，需要在2个k8s集群中的每个计算节点上都安装该服务。

5.9.1 创建服务文件

|  |
| --- |
| **vim /etc/systemd/system/log\_collector.service**  [Unit]  Description=Send journalD logs to fluentD1  [Service]  Restart=always  RestartSec=30s  Environment="FLUENTD\_HOSTIP=10.165.110.88" "FLUENTD\_PORT=9192" "JOURNALD\_POS=/tmp/log\_collector.pos"  #lets setup the position, and backfill the logs to fluentd if we need to  ExecStartPre=/bin/bash -c ' \    touch $JOURNALD\_POS; \    last\_pos=`cat $JOURNALD\_POS`; \    if [ -z  $last\_pos ]; then \      logger -t "fluentd" "journalD position not found; pre-loading all historical entries"; \      last\_line=`journalctl -u docker.service -o json  | tee ncat $FLUENTD\_HOSTIP $FLUENTD\_PORT | tail -1`; \      logger -t "fluentd" "journalD loaded all historical entries"; \      echo $last\_line | awk \'{print $4}\' | cut -d\\" -f2 > $JOURNALD\_POS; \      last\_pos=`cat $JOURNALD\_POS`; \    fi; '  # lets start streaming the logs to fluentd  ExecStart=/bin/bash -c '\    logger -t "fluentd" "streaming journalD to fluentd"; \    last\_pos=`cat $JOURNALD\_POS`; \    journalctl -u docker.service -o json -f --after-cursor=$last\_pos | ncat $FLUENTD\_HOSTIP $FLUENTD\_PORT; '  [X-Fleet]  Global=true |

5.9.2 启动服务

|  |
| --- |
| **systemctl start log\_collector**  **systemctl enable log\_collector** |

5.10 安装Mysql数据库

The Platform平台需要依赖Mysql数据库，用于保存平台数据，所以在部署平台领域系统之前，需要在平台k8s集群中部署一个Mysql数据库，并且进行数据库初始化。

5.10.1 添加yml文件

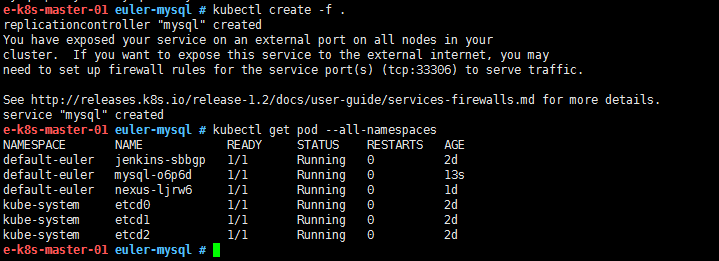
|  |
| --- |
| **mysql-pod.yml**  **vim /mnt/data/products/euler-mysql/mysql-pod.yml**  apiVersion: "v1"  kind: "ReplicationController"  #kind: "pod"  metadata:    annotations: {}    labels:      product: "mysql"    name: "mysql"    namespace: "default-euler"  spec:    replicas: 1    selector:      product: "mysql"    template:      metadata:        annotations: {}        labels:          product: "mysql"      spec:        containers:        - args:            - "--character-set-server=utf8mb4"            - "--collation-server=utf8mb4\_unicode\_ci"          image: "euler-registry.primeton.com/mysql:5.7"          name: "mysql"          ports:          - containerPort: 3306          env:            - name: "MYSQL\_DATABASE"              value: "euler"            - name: "MYSQL\_ROOT\_PASSWORD"              value: "root"            - name: "MYSQL\_USER"              value: "euler"            - name: "MYSQL\_PASSWORD"              value: "euler"          securityContext:            privileged: true          resources:            limits:              cpu: "2000m"              memory: "2Gi"            requests:              cpu: "500m"              memory: "1Gi"          volumeMounts:          - mountPath: "/var/lib/mysql"            name: "data-volume"          - mountPath: "/etc/mysql/conf.d"            name: "conf-volume"        imagePullSecrets: []        nodeSelector: {}        volumes:        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/default-euler/euler-mysql/data"          name: "data-volume"        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/default-euler/euler-mysql/conf/"          name: "conf-volume" |
| **mysql-svc.yml**  **vim /mnt/data/products/euler-mysql/mysql-svc.yml**  apiVersion: "v1"  kind: "Service"  metadata:    annotations: {}    labels:      product: "mysql"    name: "mysql"    namespace: "default-euler"  spec:    deprecatedPublicIPs: []    externalIPs: []    clusterIP: 200.110.100.36    ports:    - port: 3306      nodePort: 33306      targetPort: 3306    selector:      product: "mysql"    type: "NodePort" |

5.10.2 启动Mysql

在启动Mysql之前需要保证在搭建的Docker仓库中已经有mysql-pod.yml文件中配置的mysql镜像。

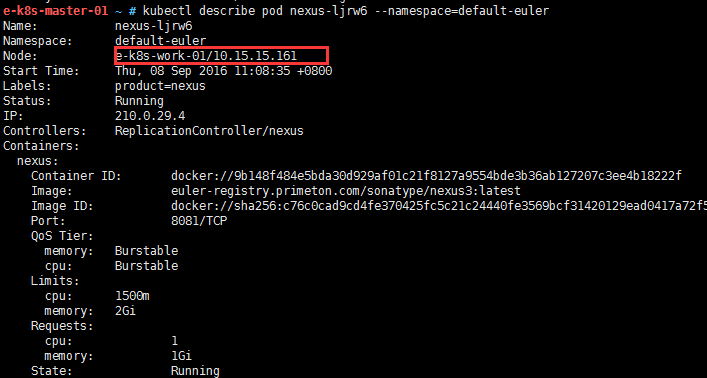
在/mnt/data/products/euler-mysql目录下执行以下命令，创建pod，并且启动mysql。

|  |
| --- |
| **kubectl create -f .** |

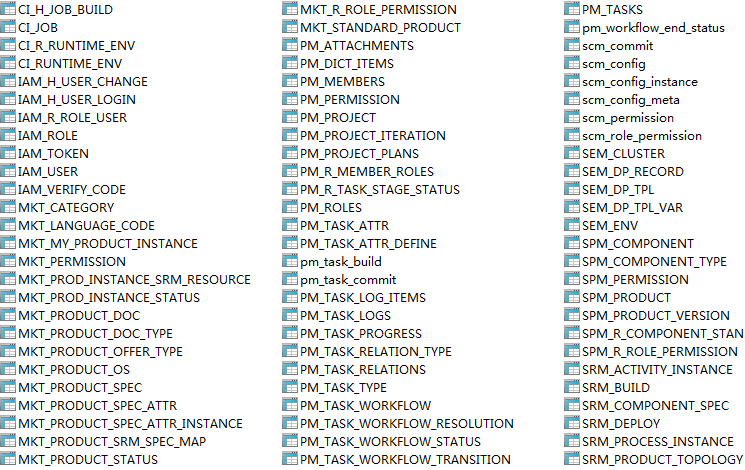


5.10.3 初始化数据库

使用**kubectl describe svc mysql-06p6d --namespace=default-euler**命令查看Mysql运行在哪一个工作节点上，让后使用WorknodeIP：NodePort作为数据库的访问地址。



通过数据库工具连接到数据库，然后使用平台的数据库初始化脚本进行平台数据库的初始化操作。



5.10.4 修改数据

完成The Platform平台的数据库初始化之后，需要对其中的sem\_cluster表的数据进行修改，修改方法请参照[4.9.5 修改配置](#修改配置)。

1. 平台领域系统部署

The Platform平台使用前后端分离的模式进行开发，由一个前端应用和十几个后端领域系统共同组成，在进行平台领域系统部署的时候，需要先完成后端领域系统的部署，再进行前端应用部署。

6.1 后端领域系统部署

目前The Platform平台的后端微服务，由以下领域系统组成。

|  |  |  |
| --- | --- | --- |
| 领域系统 | 中文名 | 功能描述 |
| Portal | 门户 | 平台提供给用户使用的统一操作门户，将各个领域系统的功能组装起来，然后提供API给前端页面进行调用 |
| IAM (Identity&AccessManagement) | 身份识别与访问管理 | 提供用户身份识别和访问控制的能力，包括用户管理、Token管理和用户授权等功能 |
| TM (Tenants Management) | 租户管理 | 提供租户管理的能力，包括租户管理和租户配额等功能 |
| SPM (Software Product Management) | 软件产品管理 | 提供产品、组件的基准定义和管理能力，包括产品类型、产品管理、组件管理等功能 |
| SCM (Software Configuration Management) | 软件配置管理 | 提供产品、组件配置管理能力，包括配置项的定义和各个环境下的配置信息管理 |
| SRM (Software Resource Management) | 软件资源管理 | 提供产品和组件自动编译、打包和部署的能力，提供部署模板管理，支持编译和部署的流程编排 |
| SEM (Software Environment Management) | 软件环境管理 | 提供租户和产品环境资源配额、负载均衡，以及运行容器的管理能力 |
| UMC (Unified Monitoring Center) | 统一监控中心 | 提供统一的监控、预警能力，包括系统日志和业务日志的监控，产品的资源使用情况和运行情况的监控 |
| VCS (Version Control System) | 版本控制系统 | 提供源代码库管理能力，包括代码库的创建、维护，分支的管理和用户权限控制 |
| CI (Continuous Integration) | 持续集成 | 提供持续集成任务调度和执行的能力，包括集成任务管理、编译、打包等功能 |
| BPR (Binary Package Repository) | 二进制包仓库 | 提供二进制包仓库的管理能力，编译产物的上传、下载和存储访问等功能 |
| PM (Project Management) | 项目管理 | 提供产品开发过程的管理和协作能力，包括任务计划、人员分工和过程跟踪等功能 |
| MKT (Market) | 市场 | 提供软件产品市场管理，包括市场类型管理、市场产品开通等功能 |

平台领域系统的部署方法基本相同，只是一些配置文件有差异，所以后文中将以一个领域系统的部署过程作为示例，然后对其他领域系统的配置文件分别进行说明。

（1）添加领域系统配置文件

|  |
| --- |
| **vcs-pod.yml**  **vim /mnt/data/euler/vcs/vcs-pod.yml**  apiVersion: "v1"  kind: "ReplicationController"  #kind: "pod"  metadata:    annotations: {}    labels:      product: "euler-vcs"    name: "euler-vcs"    namespace: "default-euler"  spec:    replicas: 1    selector:      product: "euler-vcs"    template:      metadata:        annotations: {}        labels:          product: "euler-vcs"      spec:        containers:        - args: []          image: "euler-registry.primeton.com/euler-vcs:0.2.0"          imagePullPolicy: Always          name: "euler-vcs"          ports:          - containerPort: 8080          env: []          resources:            limits:              cpu: "1000m"              memory: "2Gi"            requests:              cpu: "200m"              memory: "500m"          volumeMounts:          - mountPath: "/app/work"            name: "data-volume"          - mountPath: "/app/conf"            name: "conf-volume"        imagePullSecrets: []        nodeSelector: {}        volumes:        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/default-euler/euler-vcs/data"          name: "data-volume"        - hostPath:            path: "/mnt/euler\_storage/pod\_data\_volume/default-euler/euler-vcs/conf/"          name: "conf-volume" |

|  |
| --- |
| **vcs-svc.yml**  **vim /mnt/data/euler/vcs/vcs-svc.yml**  apiVersion: "v1"  kind: "Service"  metadata:    annotations: {}    labels:      product: "euler-vcs"    name: "euler-vcs"    namespace: "default-euler"  spec:    deprecatedPublicIPs: []    externalIPs: []    clusterIP: 200.110.100.92    ports:    - port: 8080      nodePort: 31092      targetPort: 8080    selector:      product: "euler-vcs"    type: "NodePort" |

|  |
| --- |
| **application.yml**  **vim /mnt/data/default-euler/euler-vcs/conf/application.yml**  spring.profiles.include: msf  server:      port: 8080  db:      driverClass: "com.mysql.jdbc.Driver"      user: "euler"      password: "euler"      jdbcUrl: "jdbc:mysql://200.110.100.36:3306/euler?autoReconnect=true&characterEncoding=UTF-8"      maxPoolSize: 20  gitlab:      url: http://10.174.65.57      apiToken: SPQnNj7c1zXdxfdz5\_XA  application:      name: vcs      algorithm: AES      secretKey: "jcUxQSE8\_cGY7hV73C2PeQ=="  # application server ip port  service:    registries:     - name: iam       domain: "200.110.100.81:8080"  srd:    group: euler\_group    regProtocol: etcd    address: 10.174.56.155:32379  token:    validate:      remote: true      excludeUrl: "/vcs/users/operate,/vcs/users/register,/vcs/users/changePwd"  thread:    corePoolSize: 10    maxPoolSize: 20 |

application.yml配置说明：

db：领域系统数据库配置。

gitlab：平台依赖的Gitlab配置，其中ApiToken为Gitlab中，之前预先创建的systemin账号的private token。

application：algorithm参数为领域系统对数据库中gitlab用户账号密码使用的加密方法，secretKey参数为用于加密的密钥。

service：配置IAM领域系统地址。

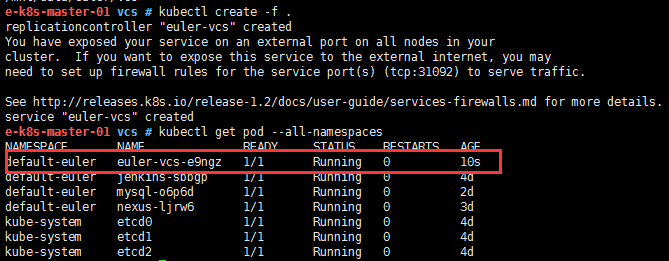
srd：配置平台ETCD（服务注册与发现中心）的地址和group属性。

Token：token校验配置，当remote参数为false时，VCS领域系统将不进行IAM Token校验；当remote参数为true时，用户需要进行IAM领域系统登录后，才能调用VCS的API。excludeUrl参数为VCS领域系统中不需要进行Token拦截校验的API restUrl。

（2）启动领域系统

进入/mnt/data/euler/vcs目录下，执行以下命令，创建pod，并且启动领域系统。

|  |
| --- |
| **kubectl create -f .** |



如上所述，部署平台领域系统只需要两步操作：（1）添加yml配置文件，（2）执行命令创建pod。

所有领域系统的yml配置文件都在下面的压缩包内，双击可打开。



6.2 平台前端应用部署

6.2.1 添加启动脚本

在用来部署平台前端应用的机器节点上添加以下启动脚本。

|  |
| --- |
| **vim /opt/portal-fe/start.sh**  DATA\_DIR=$(cd $(dirname ${0}); pwd)  IMAGE\_VERSION="0.2.0"  IMAGE\_URL="euler-registry.primeton.com/euler-pfe-enterprise"  SERVER\_PORT=31085  SERVER\_IP=10.15.15.161  docker run -d -p 8080:80 --restart=always -e P\_APP\_ENV="{'portalse\_host':'${SERVER\_IP}','portalse\_port':'${SERVER\_PORT}', 'bpr\_host':'10.15.15.162', 'bpr\_port':'31080'}" -v ${DATA\_DIR}/conf:  /etc/nginx/conf.d -v ${DATA\_DIR}/logs:/var/log/nginx ${IMAGE\_URL}:${IMAGE\_VERSION}  echo "Portal-fe start success" |

文件说明：

SERVER\_PORT：euler-pse领域系统的Nodeport

SERVER\_IP：euler-pse的pod所在的WorkNodeIP

bpr\_host：euler-bpr领域系统pod所在的WorkNodeIP

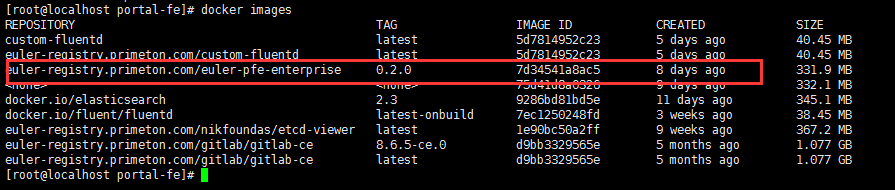
bpr\_port：euler\_bpr领域系统的Nodeport

6.2.2 修改启动脚本权限

执行**chomd a+x start.sh**命令修改启动脚本权限，将启动脚本设置成可运行文件。

6.2.3 拉取应用镜像

从docker仓库将前端应用的镜像拉去到机器节点本地。



6.2.4 启动应用

运行/opt/portal-fe/start.sh文件，然后在浏览器中访问http://10.15.15.164:8080（portal-fe所在节点IP+8080），即可访问平台。