# K8s+rancher+项目部署实践概要

## k8s集群+rancher

1.安装k8s需要各种yum源上的rpm软件包，也需要docker镜像源的一些镜像

2.要离线安装，只需要提前下载好所有rpm包和docker镜像文件，然后用这些离线包在目标机器上安装就可以

3.rpm包比较简单，用yum install/yumdownloader --downloadonly --downloaddir=./ <rpm包名>命令下载好所有rpm文件到本地目录，然后复制到要安装的机器上，用yum install <rpm文件>进行安装即可。

4.docker镜像需要预先用docker pull <xxxx>命令下载好镜像，然后用docker save xxx:版本|gzip > 文件 命令把镜像保存成本地文件，然后复制到要安装的机器上，用docker load -i <保存的镜像文件>命令恢复镜像就可以。

5.以上离线包安装好之后，解决一些k8s所需操作系统内核相关配置的先决条件后，就可以用kubeadm初始化k8s集群，加入各个node，最后安装rancher

6.这种离线方式需要注意和操作系统版本有比较强的关联关系：最好在确定的干净操作系统上(比如centos7.6)制作离线包，在目标机器也是确定的操作系统上（也是centos7.6）安装才可靠，否则有可能由于依赖的rpm版本不匹配、缺少依赖包等问题，而导致安装失败或安装后整个系统不稳定问题。

#### 1.1 安装K8S环境

假设需要安装的机器为5台虚拟机，前3台当master，后2台为worknode

准备相应的5个ip地址

另外多准备一个虚拟ip地址，作用暂时不明

##### 1.1.1 下载离线安装工具包

基于centos7.6制作好了一个离线包安装包，并编写了ansible自动化安装剧本，可以自动化安装，安装包：<https://pan.baidu.com/s/1-yHxTU6sbDWlGWQHGaIh2w>，提取码：o8op

注：其他linux版本，解压后替换掉离线包中的rpms目录就可以。

下载2个压缩包后后，复制到目标机器并解压到/home目录下，解压后如下：

[root@k8s-m1 home]*# pwd*

/home

[root@k8s-m1 home]*# ls -l*

总用量 0

drwxr-xr-x. 5 root root 163 10月 12 00:04 k8s-ansible

drwxr-xr-x. 4 root root 32 10月 12 00:06 k8s-offline-files

[root@k8s-m1 home]*# tree -v*

.

├── k8s-ansible

│ ├── ansible.cfg

│ ├── group\_vars

│ │ └── all.yml

│ ├── inventory

│ │ ├── hosts

│ │ └── singleMaster.example

│ ├── kube-flannel.yml

│ ├── roles

│ │ ├── HA

│ │ │ ├── tasks

│ │ │ │ ├── haproxy.yml

│ │ │ │ ├── keepalived.yml

│ │ │ │ └── main.yml

│ │ │ └── templates

│ │ │ ├── check\_haproxy.sh.j2

│ │ │ ├── haproxy.cfg.j2

│ │ │ └── keepalived.conf.j2

│ │ ├── docker

│ │ │ ├── files

│ │ │ │ └── docker-ce.repo

│ │ │ ├── tasks

│ │ │ │ ├── centos.yml

│ │ │ │ └── main.yml

│ │ │ └── templates

│ │ │ └── daemon.json.j2

│ │ ├── images

│ │ │ ├── tasks

│ │ │ │ └── main.yml

│ │ │ └── templates

│ │ ├── kubeadm\_config

│ │ │ ├── tasks

│ │ │ │ └── main.yml

│ │ │ └── templates

│ │ │ ├── calico.yaml.j2

│ │ │ ├── kube-flannel.yml.j2

│ │ │ └── kubeadm-config.yaml.j2

│ │ ├── rpms

│ │ │ ├── tasks

│ │ │ │ └── main.yml

│ │ │ └── templates

│ │ └── setup

│ │ ├── files

│ │ │ ├── k8s-ipvs.conf

│ │ │ └── kubernetes.conf

│ │ ├── tasks

│ │ │ ├── centos.yml

│ │ │ ├── chrony.yml

│ │ │ ├── main.yml

│ │ │ └── ntp.yml

│ │ └── templates

│ │ ├── chrony.conf.j2

│ │ ├── hosts.j2

│ │ ├── k8s-sysctl.conf.j2

│ │ └── ntp.conf.j2

│ └── setup.yml

└── k8s-offline-files

├── images

│ ├── coredns-1.3.1.tar.gz

│ ├── etcd-3.3.10.tar.gz

│ ├── kube-apiserver-v1.15.4.tar.gz

│ ├── kube-controller-manager-v1.15.4.tar.gz

│ ├── kube-proxy-v1.15.4.tar.gz

│ ├── kube-scheduler-v1.15.4.tar.gz

│ ├── pause.tar.gz

│ ├── quay.io-coreos-flannel-v0.11.0-amd64.tar.gz

│ ├── rancher-agent-2.3.0.tar.gz

│ ├── rancher-coreos-configmap-reload-v0.0.1.tar.gz

│ ├── rancher-coreos-kube-**state**-metrics-1.5.0.tar.gz

│ ├── rancher-coreos-kube-**state**-metrics-v0.0.1.tar.gz

│ ├── rancher-coreos-prometheus-config-reloader-0.29.0.tar.gz

│ ├── rancher-grafana-grafana-6.3.2.tar.gz

│ ├── rancher-istio-kubectl-1.1.5.tar.gz

│ ├── rancher-nginx-1.15.8-alpine.tar.gz

│ ├── rancher-prom-node-exporter-v0.17.0.tar.gz

│ ├── rancher-prom-prometheus-2.7.1.tar.gz

│ ├── rancher-prometheus-auth-0.2.0.tar.gz

│ └── rancher-v2.3.0.tar.gz

└── rpms

├── ansible

│ ├── PyYAML-3.10-11.el7.x86\_64.rpm

│ ├── ansible-2.8.5-1.el7.noarch.rpm

│ ├── libyaml-0.1.4-11.el7\_0.x86\_64.rpm

│ ├── python-babel-0.9.6-8.el7.noarch.rpm

│ ├── python-backports-1.0-8.el7.x86\_64.rpm

│ ├── python-backports-ssl\_match\_hostname-3.5.0.1-1.el7.noarch.rpm

│ ├── python-cffi-1.6.0-5.el7.x86\_64.rpm

│ ├── python-enum34-1.0.4-1.el7.noarch.rpm

│ ├── python-httplib2-0.9.2-1.el7.noarch.rpm

│ ├── python-idna-2.4-1.el7.noarch.rpm

│ ├── python-ipaddress-1.0.16-2.el7.noarch.rpm

│ ├── python-jinja2-2.7.2-4.el7.noarch.rpm

│ ├── python-markupsafe-0.11-10.el7.x86\_64.rpm

│ ├── python-paramiko-2.1.1-9.el7.noarch.rpm

│ ├── python-ply-3.4-11.el7.noarch.rpm

│ ├── python-pycparser-2.14-1.el7.noarch.rpm

│ ├── python-setuptools-0.9.8-7.el7.noarch.rpm

│ ├── python-six-1.9.0-2.el7.noarch.rpm

│ ├── python2-cryptography-1.7.2-2.el7.x86\_64.rpm

│ ├── python2-jmespath-0.9.0-3.el7.noarch.rpm

│ ├── python2-pyasn1-0.1.9-7.el7.noarch.rpm

│ └── sshpass-1.06-2.el7.x86\_64.rpm

├── base

│ ├── bash-completion-2.1-6.el7.noarch.rpm

│ ├── **bind**-export-libs-9.11.4-9.P2.el7.x86\_64.rpm

│ ├── **bind**-libs-9.11.4-9.P2.el7.x86\_64.rpm

│ ├── **bind**-libs-lite-9.11.4-9.P2.el7.x86\_64.rpm

│ ├── **bind**-license-9.11.4-9.P2.el7.noarch.rpm

│ ├── **bind**-utils-9.11.4-9.P2.el7.x86\_64.rpm

│ ├── conntrack-tools-1.4.4-5.el7.x86\_64.rpm

│ ├── crontabs-1.11-6.20121102git.el7.noarch.rpm

│ ├── dhclient-4.2.5-77.el7.centos.x86\_64.rpm

│ ├── dhcp-common-4.2.5-77.el7.centos.x86\_64.rpm

│ ├── dhcp-libs-4.2.5-77.el7.centos.x86\_64.rpm

│ ├── gssproxy-0.7.0-26.el7.x86\_64.rpm

│ ├── haproxy-1.5.18-9.el7.x86\_64.rpm

│ ├── htop-2.2.0-3.el7.x86\_64.rpm

│ ├── ipset-7.1-1.el7.x86\_64.rpm

│ ├── ipset-libs-7.1-1.el7.x86\_64.rpm

│ ├── ipvsadm-1.27-7.el7.x86\_64.rpm

│ ├── jq-1.5-1.el7.x86\_64.rpm

│ ├── keepalived-1.3.5-16.el7.x86\_64.rpm

│ ├── keyutils-1.5.8-3.el7.x86\_64.rpm

│ ├── libbasicobjects-0.1.1-32.el7.x86\_64.rpm

│ ├── libcollection-0.7.0-32.el7.x86\_64.rpm

│ ├── libevent-2.0.21-4.el7.x86\_64.rpm

│ ├── libini\_config-1.3.1-32.el7.x86\_64.rpm

│ ├── libnetfilter\_cthelper-1.0.0-10.el7.x86\_64.rpm

│ ├── libnetfilter\_cttimeout-1.0.0-6.el7.x86\_64.rpm

│ ├── libnetfilter\_queue-1.0.2-2.el7\_2.x86\_64.rpm

│ ├── libnfsidmap-0.25-19.el7.x86\_64.rpm

│ ├── libpath\_utils-0.2.1-32.el7.x86\_64.rpm

│ ├── libpcap-1.5.3-11.el7.x86\_64.rpm

│ ├── libref\_array-0.1.5-32.el7.x86\_64.rpm

│ ├── libseccomp-2.3.1-3.el7.x86\_64.rpm

│ ├── libtirpc-0.2.4-0.16.el7.x86\_64.rpm

│ ├── libverto-libevent-0.2.5-4.el7.x86\_64.rpm

│ ├── libxml2-python-2.9.1-6.el7\_2.3.x86\_64.rpm

│ ├── lm\_sensors-libs-3.4.0-8.20160601gitf9185e5.el7.x86\_64.rpm

│ ├── lsof-4.87-6.el7.x86\_64.rpm

│ ├── net-snmp-agent-libs-5.7.2-43.el7.x86\_64.rpm

│ ├── net-snmp-libs-5.7.2-43.el7.x86\_64.rpm

│ ├── net-tools-2.0-0.25.20131004git.el7.x86\_64.rpm

│ ├── nfs-utils-1.3.0-0.65.el7.x86\_64.rpm

│ ├── oniguruma-5.9.5-3.el7.x86\_64.rpm

│ ├── perl-5.16.3-294.el7\_6.x86\_64.rpm

│ ├── psmisc-22.20-16.el7.x86\_64.rpm

│ ├── python-chardet-2.2.1-3.el7.noarch.rpm

│ ├── python-kitchen-1.1.1-5.el7.noarch.rpm

│ ├── quota-4.01-19.el7.x86\_64.rpm

│ ├── quota-nls-4.01-19.el7.noarch.rpm

│ ├── rpcbind-0.2.0-48.el7.x86\_64.rpm

│ ├── socat-1.7.3.2-2.el7.x86\_64.rpm

│ ├── sysstat-10.1.5-18.el7.x86\_64.rpm

│ ├── tcp\_wrappers-7.6-77.el7.x86\_64.rpm

│ ├── tcpdump-4.9.2-4.el7.x86\_64.rpm

│ ├── telnet-0.17-64.el7.x86\_64.rpm

│ ├── unzip-6.0-20.el7.x86\_64.rpm

│ ├── wget-1.14-18.el7\_6.1.x86\_64.rpm

│ └── yum-utils-1.1.31-52.el7.noarch.rpm

├── docker

│ ├── audit-2.8.5-4.el7.x86\_64.rpm

│ ├── audit-libs-2.8.5-4.el7.x86\_64.rpm

│ ├── audit-libs-python-2.8.5-4.el7.x86\_64.rpm

│ ├── checkpolicy-2.5-8.el7.x86\_64.rpm

│ ├── container-selinux-2.107-3.el7.noarch.rpm

│ ├── containerd.io-1.2.6-3.3.el7.x86\_64.rpm

│ ├── device-mapper-1.02.158-2.el7.x86\_64.rpm

│ ├── device-mapper-event-1.02.158-2.el7.x86\_64.rpm

│ ├── device-mapper-event-libs-1.02.158-2.el7.x86\_64.rpm

│ ├── device-mapper-libs-1.02.158-2.el7.x86\_64.rpm

│ ├── device-mapper-persistent-data-0.8.5-1.el7.x86\_64.rpm

│ ├── docker-ce-18.09.9-3.el7.x86\_64.rpm

│ ├── docker-ce-cli-19.03.2-3.el7.x86\_64.rpm

│ ├── libcgroup-0.41-21.el7.x86\_64.rpm

│ ├── libsemanage-python-2.5-14.el7.x86\_64.rpm

│ ├── lvm2-2.02.185-2.el7.x86\_64.rpm

│ ├── lvm2-libs-2.02.185-2.el7.x86\_64.rpm

│ ├── policycoreutils-2.5-33.el7.x86\_64.rpm

│ ├── policycoreutils-python-2.5-33.el7.x86\_64.rpm

│ ├── python-IPy-0.75-6.el7.noarch.rpm

│ └── setools-libs-3.3.8-4.el7.x86\_64.rpm

├── kernel

│ ├── kernel-ml-5.3.4-1.el7.elrepo.x86\_64.rpm

│ └── kernel-ml-devel-5.3.4-1.el7.elrepo.x86\_64.rpm

└── kubernetes1.15.4

├── 14bfe6e75a9efc8eca3f638eb22c7e2ce759c67f95b43b16fae4ebabde1549f3-cri-tools-1.13.0-0.x86\_64.rpm

├── 548a0dcd865c16a50980420ddfa5fbccb8b59621179798e6dc905c9bf8af3b34-kubernetes-cni-0.7.5-0.x86\_64.rpm

├── 2813cf105c52ad1240f2cc6cba9a3f779bf2f5c4940c731a27df6e5d9557a5b1-kubeadm-1.15.4-0.x86\_64.rpm

├── dd6f87ffb5e04121b39c9b8301573225aff0a135ebf73046dcd3e21c4ab7a6cb-kubelet-1.15.4-0.x86\_64.rpm

└── de6039d16a6e77e0f38ce47cfaff9d450545757b3d09d34a10daf5667cd95ef6-kubectl-1.15.4-0.x86\_64.rpm

32 directories, 159 files

[root@k8s-m1 home]*#*

##### 1.1.2 安装ansible

没啥说的，先用yum安装ansible的离线rpm

cd /home/k8s-offline-files/rpms/ansible/

yum install -y $(ls)

##### 1.1.3 执行ansible自动化化剧本（20-50分钟）

cd /home/k8s-ansible

用vi inventory/hosts编辑主机ip配置文件，内容如下：

*# kubelet's node name must be `[a-z0-9]([-a-z0-9]\*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]\*[a-z0-9])?)\*`*

*# nodename used by k8s cluster, you also could use the ip address*

[firstmaster] *#堡垒机，是第一台master机器*

172.16.206.125 hostname=k8s-m1 nodename=172.16.206.125

[othermaster]

172.16.206.126 hostname=k8s-m2 nodename=172.16.206.126

172.16.206.127 hostname=k8s-m3 nodename=172.16.206.127

[node]

172.16.206.128 hostname=k8s-n1 nodename=172.16.206.128

172.16.206.129 hostname=k8s-n2 nodename=172.16.206.129

[master:children]

firstmaster

othermaster

[Master:children]

firstmaster

othermaster

[Allnode:children]

Master

node

用vi group\_vars/all.yml编辑变量文件的内容：修改ssh密码，修改虚拟IP地址、掩码和网卡设备名，其他保持原样即可：

ansible\_ssh\_port: 22

ansible\_ssh\_pass: '【密码】'

ntp\_type: chrony **# or ntp**

ntp\_servers:

- 'cn.pool.ntp.org'

- 's1b.time.edu.cn'

TimeZone: 'Asia/Shanghai'

kernel: !!**bool** true **# or change true to false**

VIP: '【准备的虚拟ip】'

VIP\_NETMASK: 24

INTERFACE\_NAME: eth0

PodSubnet: '10.68.0.0/16'

docker:

version: '18.09'

dataDir: /**var**/lib/docker

execOpts:

- 'native.cgroupdriver=systemd'

registryMirrors:

- https:*//fz5yth0r.mirror.aliyuncs.com*

**# insecureRegistries:**

**# - 100.64.2.52:9999**

**# - 100.64.1.31:9999**

kubelet:

swap: false **# 如果不希望关闭swap设置成true**

然后就执行剧本：

**ansible-playbook** **setup**.yml

剧本将在5台机器上自动化完成以下工作：

* 分发rpm包到各个机器
* 用yum安装所有的包：base、docker、kernel(升级linux内核)、kubernetes1.15.4基础工具(kubectl kubeadm kubelet)
* 优化docker的配置参数，设置为服务自启动
* 配置基于keepalive+Haproxy的高可用环境
* 分发docker镜像文件，并加载到每台机器的docker环境中
* 生成并分发kubeadm的安装配置文件
* 关闭firewalld NetworkManager dnsmasq
* 禁止rsyslog获取journald日志
* 设置系统时钟同步服务
* 修改默认启动内核到新内核
* 设置linux启动内核参数适应docker
* 设置系统时区
* 修改fstab文件关闭swap
* 生成并分发hosts文件
* 分发linux内核运行参数优化文件，并配置生效
* 关闭sshd反向解析（提高ssh连接速度）
* 设置每台机器的hostname
* 重启机器

期间可以喝个茶，等所有5台机器自动安装配置基础环境，需要半个小时到1小时左右时间，成功完成后机器会自动重启

##### 1.1.4 初始化k8s集群（5-10分钟）

##### 1.1.5 kubeadm的初始准备（只需要在k8s-m1上操作）

重启后用ssh登录到第一台master机器上，  
确认kubeadm配置文件已经自动生成 vi /etc/kuber/kubeadm/kubeadm-config.yaml，内容如下：

apiVersion: kubeadm.k8s.io/v1beta1

kind: ClusterConfiguration

imageRepository: registry.aliyuncs.com/google\_containers

**networking:**

serviceSubnet: 10.96.0.0/12

podSubnet: 10.68.0.0/16

kubernetesVersion: v1.15.4

controlPlaneEndpoint: api.me:6443

**apiServer:**

certSANs:

- api.me

---

apiVersion: kubelet.config.k8s.io/v1beta1

kind: KubeletConfiguration

**systemReserved:**

cpu: "0.25"

memory: 128Mi

imageGCHighThresholdPercent: 85

imageGCLowThresholdPercent: 80

imageMinimumGCAge: 2m0s

---

apiVersion: kubeproxy.config.k8s.io/v1alpha1

kind: KubeProxyConfiguration

**ipvs:**

minSyncPeriod: 1s

scheduler: rr

syncPeriod: 10s

mode: ipvs

##### 1.1.6 使用kubeadm，用upload-certs和config参数初始化集群

[root@k8s-m1 k8s-ansible]*# kubeadm init --config=/etc/kuber/kubeadm/kubeadm-config.yaml --upload-certs*

[init] Using Kubernetes version: v1.15.4

[preflight] Running pre-flight checks

[preflight] Pulling images required **for** setting up a Kubernetes cluster

[preflight] This might take a minute **or** two, depending on the speed **of** your internet connection

[preflight] You can also perform **this** action **in** beforehand using 'kubeadm config images pull'

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Activating the kubelet service

[certs] Using certificateDir folder "/etc/kubernetes/pki"

[certs] Generating "etcd/ca" certificate **and** key

[certs] Generating "etcd/healthcheck-client" certificate **and** key

[certs] Generating "apiserver-etcd-client" certificate **and** key

[certs] Generating "etcd/server" certificate **and** key

[certs] etcd/server serving cert **is** signed **for** DNS names [k8s1 localhost] **and** IPs [172.16.206.125 127.0.0.1 ::1]

[certs] Generating "etcd/peer" certificate **and** key

[certs] etcd/peer serving cert **is** signed **for** DNS names [k8s1 localhost] **and** IPs [172.16.206.125 127.0.0.1 ::1]

[certs] Generating "ca" certificate **and** key

[certs] Generating "apiserver" certificate **and** key

[certs] apiserver serving cert **is** signed **for** DNS names [k8s1 kubernetes kubernetes.**default** kubernetes.**default**.svc kubernetes.**default**.svc.cluster.local api.me api.me] **and** IPs [10.96.0.1 172.16.206.125 172.16.206.251]

[certs] Generating "apiserver-kubelet-client" certificate **and** key

[certs] Generating "front-proxy-ca" certificate **and** key

[certs] Generating "front-proxy-client" certificate **and** key

[certs] Generating "sa" key **and** public key

[kubeconfig] Using kubeconfig folder "/etc/kubernetes"

[kubeconfig] Writing "admin.conf" kubeconfig file

[kubeconfig] Writing "kubelet.conf" kubeconfig file

[kubeconfig] Writing "controller-manager.conf" kubeconfig file

[kubeconfig] Writing "scheduler.conf" kubeconfig file

[control-plane] Using manifest folder "/etc/kubernetes/manifests"

[control-plane] Creating static Pod manifest **for** "kube-apiserver"

[control-plane] Creating static Pod manifest **for** "kube-controller-manager"

[control-plane] Creating static Pod manifest **for** "kube-scheduler"

[etcd] Creating static Pod manifest **for** local etcd **in** "/etc/kubernetes/manifests"

[wait-control-plane] Waiting **for** the kubelet to boot up the control plane **as** static Pods **from** directory "/etc/kubernetes/manifests". This can take up to 4m0s

[apiclient] All control plane components are healthy after 20.002362 seconds

[upload-config] Storing the configuration used **in** ConfigMap "kubeadm-config" **in** the "kube-system" Namespace

[kubelet] Creating a ConfigMap "kubelet-config-1.15" **in** namespace kube-system with the configuration **for** the kubelets **in** the cluster

[upload-certs] Storing the certificates **in** Secret "kubeadm-certs" **in** the "kube-system" Namespace

[upload-certs] Using certificate key:

0e4ebf7815a375aac2b61cfa62c8b3305c358ed0f060bf0be284acb788ef6f80

[mark-control-plane] Marking the node k8s1 **as** control-plane **by** adding the label "node-role.kubernetes.io/master=''"

[mark-control-plane] Marking the node k8s1 **as** control-plane **by** adding the taints [node-role.kubernetes.io/master:NoSchedule]

[bootstrap-token] Using token: viv09y.lsn40nn5o8hixy7g

[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles

[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs **in** order **for** nodes to get long term certificate credentials

[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs **from** a Node Bootstrap Token

[bootstrap-token] configured RBAC rules to allow certificate rotation **for** all node client certificates **in** the cluster

[bootstrap-token] Creating the "cluster-info" ConfigMap **in** the "kube-public" namespace

[addons] Applied essential addon: CoreDNS

[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following **as** a regular user:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one **of** the options listed at:

https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number **of** the control-plane node running the following command on each **as** root:

kubeadm join api.me:6443 --token viv09y.lsn40nn5o8hixy7g \

--discovery-token-ca-cert-hash sha256:27cd74e9d1624fde9738405c29bb17e33ad6e5950f6ad2126936c9e9b7187c5a \

--control-plane --certificate-key 0e4ebf7815a375aac2b61cfa62c8b3305c358ed0f060bf0be284acb788ef6f80

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!

As a safeguard, uploaded-certs will be deleted **in** two hours; If necessary, you can use

"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

Then you can join any number **of** worker nodes **by** running the following on each **as** root:

kubeadm join api.me:6443 --token viv09y.lsn40nn5o8hixy7g \

--discovery-token-ca-cert-hash sha256:27cd74e9d1624fde9738405c29bb17e33ad6e5950f6ad2126936c9e9b7187c5a

[root@k8s-m1 k8s-ansible]*#*

##### 1.1.7 然后执行：

**mkdir** -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo **chown** $(id -u):$(id -g) $HOME/.kube/config

##### 1.1.8 部署flannel网络(网络组件只要且只能安装1个)

注意要使用与podSubnet上面配置匹配的pod CIDR参数来安装CNI插件,要按照实际情况修改文件中的值。

cd /home/k8s-ansible

kubectl apply -f kube-flannel.yml

验证1个master节点是Ready：用命令 kubectl get ....命令

[root@k8s-m1 Kubernetes-ansible]*# kubectl get po --all-namespaces*

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-**system** coredns-bccdc95cf-m2q44 1/1 Running 0 2m50s

kube-**system** coredns-bccdc95cf-rh879 1/1 Running 0 2m50s

kube-**system** etcd-k8s1 1/1 Running 0 114**s**

kube-**system** kube-apiserver-k8s1 1/1 Running 0 118**s**

kube-**system** kube-controller-manager-k8s1 1/1 Running 0 2m7s

kube-**system** kube-flannel-ds-amd64-5hw2b 1/1 Running 0 46**s**

kube-**system** kube-proxy-6rdc7 1/1 Running 0 2m49s

kube-**system** kube-scheduler-k8s1 1/1 Running 0 2m3s

[root@k8s-m1 k8s-ansible]*# kubectl get node*

NAME STATUS ROLES AGE VERSION

k8s1 Ready master 3m12s v1.15.4

[root@k8s1 Kubernetes-ansible]*# kubectl get cs*

NAME STATUS MESSAGE ERROR

controller-manager Healthy ok

scheduler Healthy ok

etcd-0 Healthy {"health":"true"}

[root@k8s-m1 k8s-ansible]*#*

这样，第一台master就完成了

##### 1.1.9 加入其它master和worknode

然后，  
在另外第2，3台master机器上执行（要从之前kubeadm安装成功后的提示信息中复制要执行的内容）：

**kubeadm** **join** **api**.me:6443 **--token** **【生成的token】**

为了能执行kubectl命令，需要执行以下命令：

**mkdir** -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo **chown** $(id -u):$(id -g) $HOME/.kube/config

在最后2台worknode上执行：

**kubeadm** **join** api.me:6443 **--token** **【节点token】**

##### 1.1.10 确认安装成功

安装成功后，是这样：

[root@k8s-m1 k8s-ansible]*# kubectl get po --all-namespaces*

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-**system** coredns-bccdc95cf-m2q44 1/1 Running 0 9m35s

kube-**system** coredns-bccdc95cf-rh879 1/1 Running 0 9m35s

kube-**system** etcd-k8s1 1/1 Running 0 8m39s

kube-**system** etcd-k8s2 1/1 Running 0 82**s**

kube-**system** etcd-k8s3 1/1 Running 0 2m10s

kube-**system** kube-apiserver-k8s1 1/1 Running 0 8m43s

kube-**system** kube-apiserver-k8s3 1/1 Running 0 2m10s

kube-**system** kube-controller-manager-k8s1 1/1 Running 1 8m52s

kube-**system** kube-controller-manager-k8s2 1/1 Running 0 29**s**

kube-**system** kube-controller-manager-k8s3 1/1 Running 0 2m10s

kube-**system** kube-flannel-ds-amd64-4mpqv 1/1 Running 0 72**s**

kube-**system** kube-flannel-ds-amd64-5hw2b 1/1 Running 0 7m31s

kube-**system** kube-flannel-ds-amd64-rdlbj 1/1 Running 0 68**s**

kube-**system** kube-flannel-ds-amd64-tsktn 1/1 Running 0 84**s**

kube-**system** kube-flannel-ds-amd64-v228z 1/1 Running 0 2m10s

kube-**system** kube-proxy-6hkzd 1/1 Running 0 2m10s

kube-**system** kube-proxy-6rdc7 1/1 Running 0 9m34s

kube-**system** kube-proxy-9c8x4 1/1 Running 0 72**s**

kube-**system** kube-proxy-fwmb9 1/1 Running 0 84**s**

kube-**system** kube-proxy-pnjns 1/1 Running 0 68**s**

kube-**system** kube-scheduler-k8s1 1/1 Running 1 8m48s

kube-**system** kube-scheduler-k8s2 1/1 Running 0 30**s**

kube-**system** kube-scheduler-k8s3 1/1 Running 0 2m10s

[root@k8s-m1 k8s-ansible]*# kubectl get node*

NAME STATUS ROLES AGE VERSION

k8s1 Ready master 10**m** v1.15.4

k8s2 Ready master 2m14s v1.15.4

k8s3 Ready master 3**m** v1.15.4

k8s4 Ready <none> 2m2s v1.15.4

k8s5 Ready <none> 118**s** v1.15.4

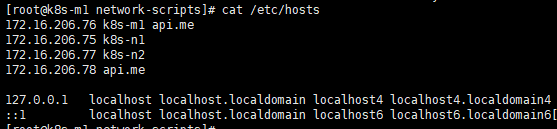
[root@k8s-m1 k8s-ansible]*#*

##### 1.1.11. k8s集群新增节点

参考：<https://www.cnblogs.com/chuangcc/p/10697394.html>

补充：1.配置主机名：为了集群各节点主机名统一，命令：hostnamectl set-hostname [hostname]

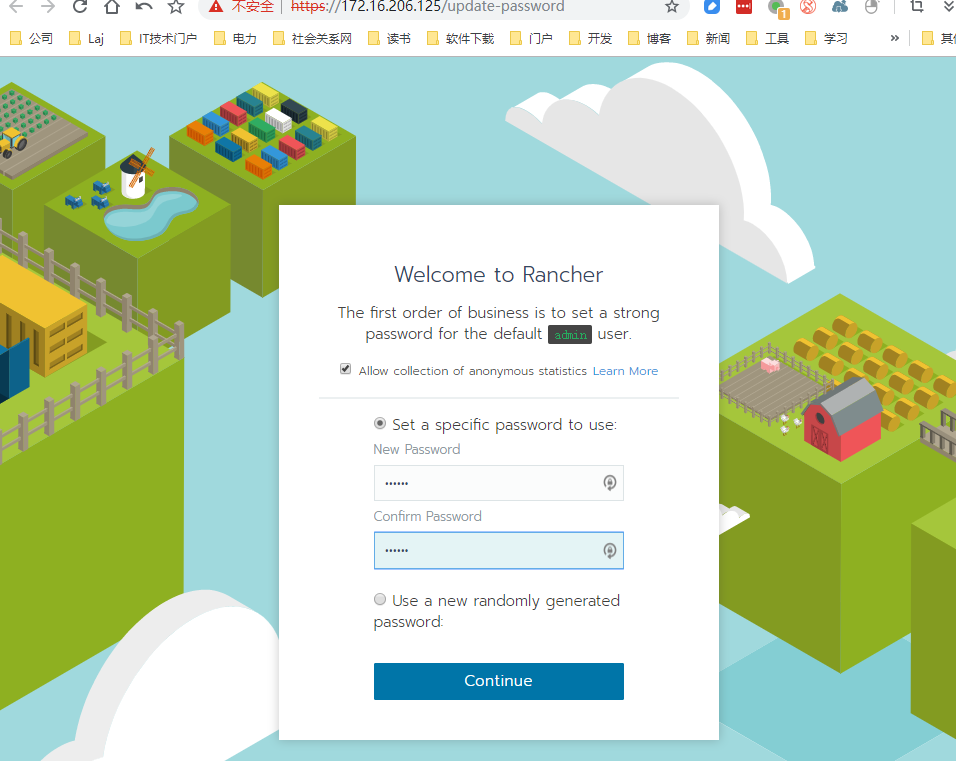
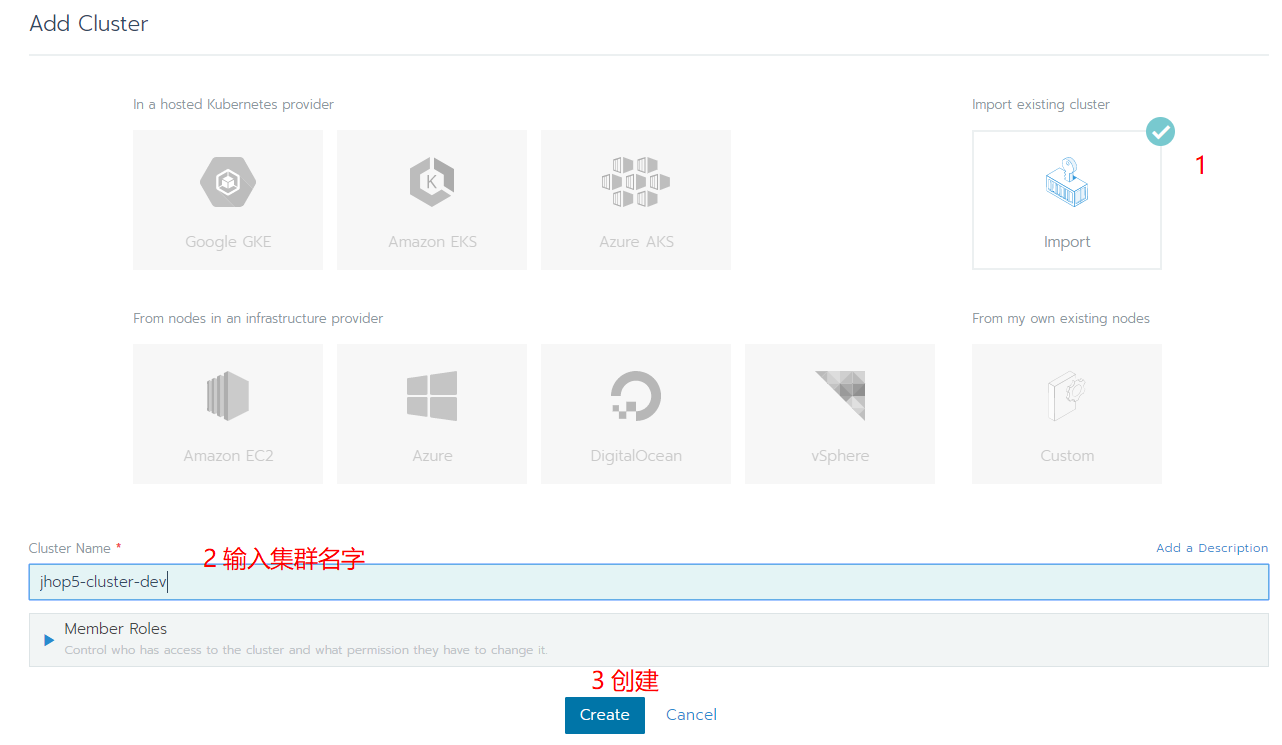
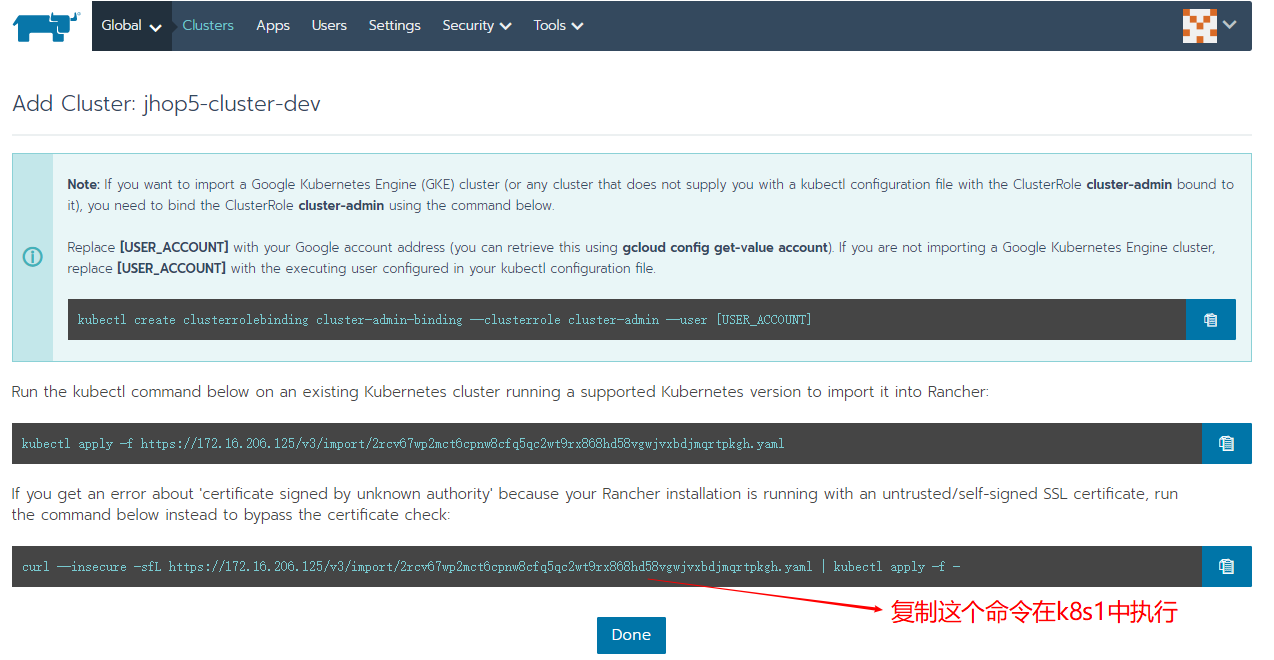
2.配置hosts文件,例：



### 1.2. 安装k8s机器管理工具rancher2（5-20分钟）

1. 在k8s1上执行命令：
2. docker run -d --restart=unless-stopped \
3. -p 80:80 -p 443:443 \

rancher/rancher:v2.3.0

成功后，在浏览器打开链接, 安装页面提示设置admin密码为123456，下一步保存url。  
  
然后要加我们上面已经创建好的集群：  
  
然后按界面提示在k8s1中执行命令  


点【Done】按钮，等待集群初始化完成，大功告成！

补充：k8s支持 CentOS 7.4 / 7.5 / 7.6 / 7.7版本，Linux版本不同，依赖的rpm包有所不同，需要更新相应的rpm包。

## 项目部署

### 2.1基础信息

Harbor 镜像管理

Nacos 配置中心

Postgresql 数据源数据库（项目生产数据）

Redis 缓存数据库（无基础数据）

Rabbitmq jhop-business、jhop-platform依赖（消息代理软件）

Consul 服务网格-负责服务之间的网络调用、限流、熔断和监控

Nfs 网络文件系统,rabbitmq 依赖

### 2.2依赖部署

#### 2.2.1 Harbor

参考：<https://blog.csdn.net/weixin_41465338/article/details/80146218>

#### 2.2.2 Nacos

单点部署参考：<https://www.cnblogs.com/yangxianyu/p/11159080.html>

集群部署：

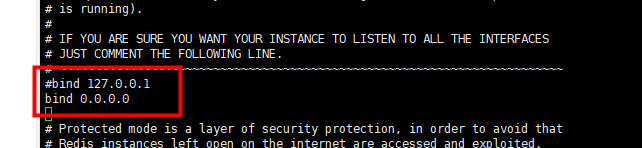
#### 2.2.3 Postgresql

参考：<https://blog.csdn.net/thinktik/article/details/81090177>

#### 2.2.4 Redis

1.单点部署参考：<https://www.jianshu.com/p/bfc283fb2388>

注意：远程访问redis服务需修改redis.conf



也可以在集群中进行部署

#### 2.2.5 Rabbitmq

<https://blog.csdn.net/hellozpc/article/details/81436980> Consul

#### Nfs

参考：<https://blog.csdn.net/aixiaoyang168/article/details/83782336>

### 2.3镜像制作

#### 2.3.2 修改源码配置文件，关联配置中心

将代码配置提取出来，通过配置中心进行管理、配置（nacos）

#### 2.3.4 使用maven打包程序

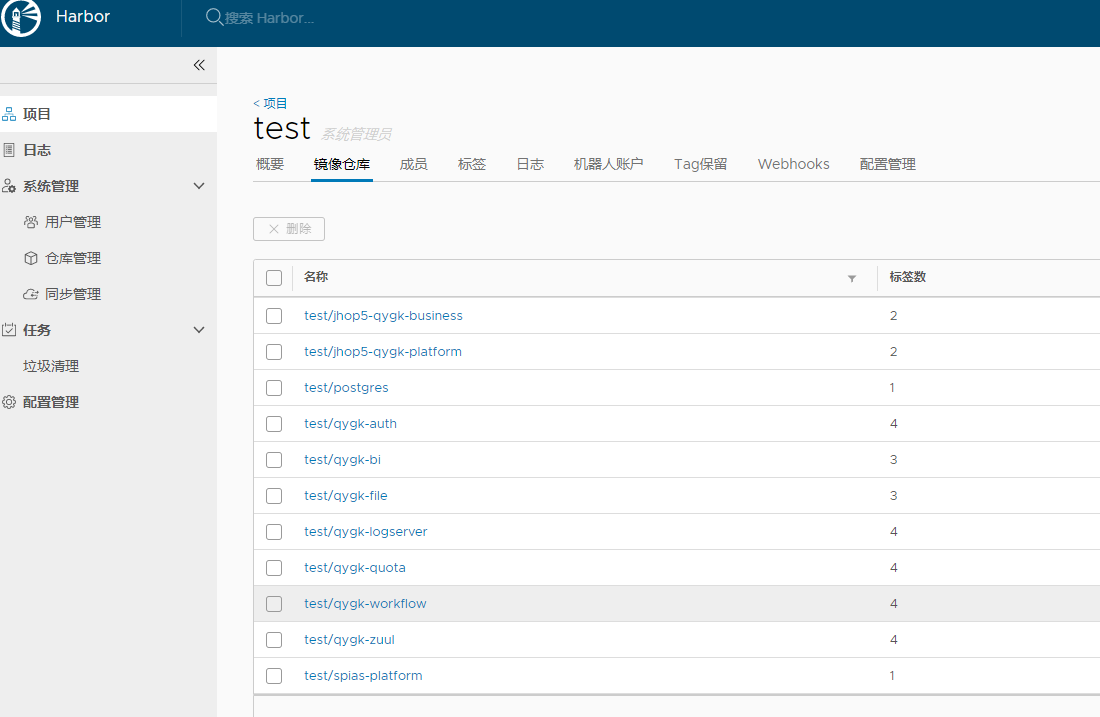
安装参考：<https://www.cnblogs.com/jimmy-muyuan/p/7895933.html>

切换到编译模块目录下执行编译打包（mvn install）

#### 2.3.6 打包成docker镜像推送到Harbor

编译服务器安装docker：<https://www.runoob.com/docker/centos-docker-install.html>

利用代码路径下的Dockerfile文件，创建docker镜像，然后给镜像打上tag推送到harbor



### 2.4后端部署

可以使用ks8命令部署程序，也可以在rancher（图形化界面）上部署程序，