SpringCloud 微服务 RuoYi-Cloud 部署文档（DevOps 版）（2024-01-25版）

> 方式1：argo-rollouts + istio（金丝雀发布）（渐进式交付）（推荐）
>
> 方式2：k8s-deployment滚动更新（不推荐）

基础集群组件

```
0、k8s集群（k8s-1.29.1）
1、helm、kubens、kubectl补全
2、ingress-nginx
3、istio
4、Argo-Rollouts
5、nfs-subdir-external-provisioner
6、metrics-server
7、gitlab
8、harbor
9、jenkins
```

RuoYi-Cloud 业务组件

```
0、mysql-8.0.28

1、nacos-2.1.0

2、redis-7.2.4
```

一、基础集群组件


```
0、k8s集群（k8s-1.29.1）
1、helm、kubens、kubectl补全
2、ingress-nginx
3、istio
4、Argo-Rollouts
5、nfs-subdir-external-provisioner
6、metrics-server
7、gitlab
8、harbor
9、jenkins
```


0、k8s集群（k8s-1.29.1）

containerd-1.6.27 + k8s-1.29.1（最新）（kubeadm方式）（containerd容器运行时版）

> kubeadm方式安装最新版k8s-1.29.1（containerd容器运行时）
>
> containerd-1.6.27 + k8s-1.29.1（最新）（kubeadm方式）

> containerd-1.6.27
>
> k8s-1.29.1

- k8s-master（centos-7.9） （4c8g-200g）
- k8s-node1（centos-7.9） （8c16g-200g）
- k8s-node2（rocky-9.3） （8c16g-200g）
- k8s-node3（rocky-9.3） （8c16g-200g）

0、环境准备（centos-7.9、rocky-9.3 环境配置+调优）

```
# 颜色
echo "PS1='\[\033[35m\][\[\033[OOm\]\[\033[31m\]\u\[\033[33m\]\[\033[33m\]@\
[\033[03m\]\[\033[35m\]\h\[\033[OOm\] \[\033[5;32m\]\w\[\033[OOm\]\[\033[35m\]]\
[\033[OOm\]\[\033[5;31m\]\\$\[\033[OOm\] '" >> ~/.bashrc && source ~/.bashrc

echo 'PS1="[\[\e[33m\]\u\[\e[Om\]\[\e[31m\]@\[\e[Om\]\[\e[35m\]\h\[\e[Om\]:\
[\e[32m\]\w\[\e[Om\]] \[\e[33m\]\t\[\e[Om\] \[\e[31m\]Power\[\e[Om\]=\
[\e[32m\]\!\[\e[Om\] \[\e[35m\]^O^\[\e[Om\]\n\[\e[95m\]公主请输命令^O^\[\e[Om\] \
[\e[36m\]\\$\[\e[Om\] "' >> ~/.bashrc && source ~/.bashrc

# 0、centos7 环境配置
# 安装 vim
yum -y install vim wget net-tools

# 行号
echo "set nu" >> /root/.vimrc

# 搜索关键字高亮
sed -i "8calias grep='grep --color'" /root/.bashrc

# 腾讯源
cp /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo-bak
wget -O /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.cloud.tencent.com/repo/centos7_base.repo
wget -O /etc/yum.repos.d/CentOS-Epel.repo
http://mirrors.cloud.tencent.com/repo/epel-7.repo

yum clean all
yum makecache
```

```
# 1、设置主机名
hostnamectl set-hostname k8s-master && su -
hostnamectl set-hostname k8s-node1 && su -
hostnamectl set-hostname k8s-node2 && su -
hostnamectl set-hostname k8s-node3 && su -
```

```
# 2、添加hosts解析
cat >> /etc/hosts << EOF
192.168.1.200 k8s-master
192.168.1.201 k8s-node1
192.168.1.202 k8s-node2
192.168.1.203 k8s-node3
EOF
```

```
# 3、同步时间
yum -y install ntp

systemctl enable ntpd --now
```

```
# 4、永久关闭seLinux(需重启系统生效)
setenforce 0
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
```

```
# 5、永久关闭swap(需重启系统生效)
swapoff -a  # 临时关闭
sed -i 's/.*swap.*/#&/g' /etc/fstab # 永久关闭
```

```
# 6、升级内核为5.4版本(需重启系统生效)
# https://elrepo.org/tiki/kernel-lt
# https://elrepo.org/linux/kernel/el7/x86_64/RPMS/

rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-6.el7.elrepo.noarch.rpm

yum --disablerepo="*" --enablerepo="elrepo-kernel" list available

yum --enablerepo=elrepo-kernel install -y kernel-lt

grub2-set-default 0
```

```
# 这里先重启再继续
# reboot
```

```
# 7、关闭防火墙、清空iptables规则
systemctl disable firewalld && systemctl stop firewalld

iptables -F && iptables -t nat -F && iptables -t mangle -F && iptables -X &&
iptables -P FORWARD ACCEPT
```

```
# 8、关闭 NetworkManager
systemctl disable NetworkManager && systemctl stop NetworkManager
```

```
# 9、加载IPVS模块
yum -y install ipset ipvsadm

mkdir -p /etc/sysconfig/modules
cat > /etc/sysconfig/modules/ipvs.modules <<EOF
modprobe -- ip_vs
modprobe -- ip_vs_rr
modprobe -- ip_vs_wrr
modprobe -- ip_vs_sh
modprobe -- nf_conntrack
EOF

modprobe -- nf_conntrack
```

```
chmod 755 /etc/sysconfig/modules/ipvs.modules && bash
/etc/sysconfig/modules/ipvs.modules && lsmod | grep -e ip_vs -e nf_conntrack
```

```
# 10、开启br_netfilter、ipv4 路由转发
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# 设置所需的 sysctl 参数，参数在重新启动后保持不变
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# 应用 sysctl 参数而不重新启动
sudo sysctl --system

# 查看是否生效
lsmod | grep br_netfilter
lsmod | grep overlay

sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables
net.ipv4.ip_forward
```

```
# 11、内核调优
cat > /etc/sysctl.d/99-sysctl.conf << 'EOF'
# sysctl settings are defined through files in
# /usr/lib/sysctl.d/, /run/sysctl.d/, and /etc/sysctl.d/.
#
# Vendors settings live in /usr/lib/sysctl.d/.
# To override a whole file, create a new file with the same in
# /etc/sysctl.d/ and put new settings there. To override
# only specific settings, add a file with a lexically later
# name in /etc/sysctl.d/ and put new settings there.
#
# For more information, see sysctl.conf(5) and sysctl.d(5).

# Controls IP packet forwarding

# Controls source route verification
net.ipv4.conf.default.rp_filter = 1

# Do not accept source routing
net.ipv4.conf.default.accept_source_route = 0

# Controls the System Request debugging functionality of the kernel

# Controls whether core dumps will append the PID to the core filename.
```

```
# Useful for debugging multi-threaded applications.
kernel.core_uses_pid = 1

# Controls the use of TCP syncookies
net.ipv4.tcp_syncookies = 1

# Controls the maximum size of a message, in bytes
kernel.msgmnb = 65536

# Controls the default maxmimum size of a mesage queue
kernel.msgmax = 65536

net.ipv4.conf.all.promote_secondaries = 1
net.ipv4.conf.default.promote_secondaries = 1
net.ipv6.neigh.default.gc_thresh3 = 4096

kernel.sysrq = 1
net.ipv6.conf.all.disable_ipv6=0
net.ipv6.conf.default.disable_ipv6=0
net.ipv6.conf.lo.disable_ipv6=0
kernel.numa_balancing = 0
kernel.shmmax = 68719476736
kernel.printk = 5
net.core.rps_sock_flow_entries=8192
net.bridge.bridge-nf-call-ip6tables=1
net.ipv4.ip_local_reserved_ports=60001,60002
net.core.rmem_max=16777216
fs.inotify.max_user_watches=524288
kernel.core_pattern=core
net.core.dev_weight_tx_bias=1
net.ipv4.tcp_max_orphans=32768
kernel.pid_max=4194304
kernel.softlockup_panic=1
fs.file-max=3355443
net.core.bpf_jit_harden=1
net.ipv4.tcp_max_tw_buckets=32768
fs.inotify.max_user_instances=8192
net.core.bpf_jit_kallsyms=1
vm.max_map_count=262144
kernel.threads-max=262144
net.core.bpf_jit_enable=1
net.ipv4.tcp_keepalive_time=600
net.ipv4.tcp_wmem=4096 12582912    16777216
net.core.wmem_max=16777216
net.ipv4.neigh.default.gc_thresh1=2048
net.core.somaxconn=32768
net.ipv4.neigh.default.gc_thresh3=8192
net.ipv4.ip_forward=1
net.ipv4.neigh.default.gc_thresh2=4096
net.ipv4.tcp_max_syn_backlog=8096
net.bridge.bridge-nf-call-iptables=1
net.ipv4.tcp_rmem=4096  12582912        16777216
EOF

# 应用 sysctl 参数而不重新启动
```

```
sudo sysctl --system
```

```
# 12、设置资源配置文件
cat >> /etc/security/limits.conf << 'EOF'
* soft nofile 100001
* hard nofile 100002
root soft nofile 100001
root hard nofile 100002
* soft memlock unlimited
* hard memlock unlimited
* soft nproc 254554
* hard nproc 254554
* soft sigpending 254554
* hard sigpending 254554
EOF

grep -vE "^\s*#" /etc/security/limits.conf

ulimit -a
```

1、安装containerd-1.6.27（官方源）(centos-7.9、rocky-9.3)

```
wget -O /etc/yum.repos.d/docker-ce.repo
https://download.docker.com/linux/centos/docker-ce.repo

yum makecache
```

```
yum list containerd.io --showduplicates | sort -r
```

```
yum -y install containerd.io-1.6.27
```

```
containerd config default | sudo tee /etc/containerd/config.toml

# 修改cgroup Driver为systemd
sed -ri 's#SystemdCgroup = false#SystemdCgroup = true#'
/etc/containerd/config.toml

# 更改sandbox_image
sed -ri
's#registry.k8s.io\/pause:3.6#registry.aliyuncs.com\/google_containers\/pause:3.
9#' /etc/containerd/config.toml

# 添加镜像加速
# https://github.com/DaoCloud/public-image-mirror
# 1、指定配置文件目录
sed -i 's/config_path = ""/config_path = "\/etc\/containerd\/certs.d\/"/g'
/etc/containerd/config.toml
# 2、配置加速
# docker.io 镜像加速
mkdir -p /etc/containerd/certs.d/docker.io
cat > /etc/containerd/certs.d/docker.io/hosts.toml << 'EOF'
server = "https://docker.io" # 源镜像地址
```

```
[host."https://xk9ak4u9.mirror.aliyuncs.com"] # 阿里-镜像加速地址
  capabilities = ["pull","resolve"]

[host."https://docker.m.daocloud.io"] # 道客-镜像加速地址
  capabilities = ["pull","resolve"]

[host."https://dockerproxy.com"] # 镜像加速地址
  capabilities = ["pull", "resolve"]

[host."https://docker.mirrors.sjtug.sjtu.edu.cn"] # 上海交大-镜像加速地址
  capabilities = ["pull","resolve"]

[host."https://docker.mirrors.ustc.edu.cn"] # 中科大-镜像加速地址
  capabilities = ["pull","resolve"]

[host."https://docker.nju.edu.cn"] # 南京大学-镜像加速地址
  capabilities = ["pull","resolve"]

[host."https://registry-1.docker.io"]
  capabilities = ["pull","resolve","push"]
EOF

# registry.k8s.io 镜像加速
mkdir -p /etc/containerd/certs.d/registry.k8s.io
cat > /etc/containerd/certs.d/registry.k8s.io/hosts.toml << 'EOF'
server = "https://registry.k8s.io"

[host."https://k8s.m.daocloud.io"]
  capabilities = ["pull", "resolve", "push"]
EOF

# quay.io 镜像加速
mkdir -p /etc/containerd/certs.d/quay.io
cat > /etc/containerd/certs.d/quay.io/hosts.toml << 'EOF'
server = "https://quay.io"

[host."https://quay.m.daocloud.io"]
  capabilities = ["pull", "resolve", "push"]
EOF

# docker.elastic.co镜像加速
mkdir -p /etc/containerd/certs.d/docker.elastic.co
tee /etc/containerd/certs.d/docker.elastic.co/hosts.toml << 'EOF'
server = "https://docker.elastic.co"

[host."https://elastic.m.daocloud.io"]
  capabilities = ["pull", "resolve", "push"]
EOF

systemctl daemon-reload

systemctl enable containerd --now

systemctl restart containerd
```

```
systemctl status containerd
```

> 镜像加速配置无需重启服务，即可生效

```
#设置crictl
cat << EOF >> /etc/crictl.yaml
runtime-endpoint: unix:///var/run/containerd/containerd.sock
image-endpoint: unix:///var/run/containerd/containerd.sock
timeout: 10
debug: false
EOF
```

2、安装k8s（kubeadm-1.29.1、kubelet-1.29.1、kubectl-1.29.1）（官方源）(centos-7.9、rocky-9.3)

```
cat > /etc/yum.repos.d/kubernetes.repo << 'EOF'
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/
enabled=1
gpgcheck=0
EOF

yum makecache

yum -y install kubeadm-1.29.1 kubelet-1.29.1 kubectl-1.29.1

systemctl enable --now kubelet
```

3、初始化 k8s-1.29.1 集群

```
mkdir ~/kubeadm_init && cd ~/kubeadm_init

kubeadm config print init-defaults > kubeadm-init.yaml

cat > ~/kubeadm_init/kubeadm-init.yaml << EOF
apiVersion: kubeadm.k8s.io/v1beta3
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.168.1.200 # 修改自己的ip
  bindPort: 6443
nodeRegistration:
  criSocket: unix:///var/run/containerd/containerd.sock
  imagePullPolicy: IfNotPresent
  name: k8s-master
  taints:
```

```
  - effect: NoSchedule
    key: node-role.kubernetes.io/k8s-master
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta3
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controllerManager: {}
dns: {}
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: registry.aliyuncs.com/google_containers
kind: ClusterConfiguration
kubernetesVersion: v1.29.1
networking:
  dnsDomain: cluster.local
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/12
scheduler: {}
---
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: ipvs
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
EOF
```

```
# 查看所需镜像列表
kubeadm config images list --config kubeadm-init.yaml
```

```
# 预拉取镜像
kubeadm config images pull --config kubeadm-init.yaml
```

```
# 初始化
kubeadm init --config=kubeadm-init.yaml | tee kubeadm-init.log
```

```
# 配置 kubectl
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

4、安装 k8s 集群网络（calico）

查看calico与k8s的版本对应关系

> https://docs.tigera.io/calico/3.27/getting-started/kubernetes/requirements

> 这里k8s-1.29.1，所以使用calico-v3.27.0版本（版本对应很关键）

```
mkdir -p ~/calico-yml

cd ~/calico-yml && wget
https://github.com/projectcalico/calico/raw/v3.27.0/manifests/calico.yaml
```

```
1 修改CIDR
- name: CALICO_IPV4POOL_CIDR
  value: "10.244.0.0/16"

2 指定网卡
# Cluster type to identify the deployment type
  - name: CLUSTER_TYPE
  value: "k8s,bgp"
# 下面添加
  - name: IP_AUTODETECTION_METHOD
    value: "interface=ens33,ens160"
    # ens33为本地网卡名字（自己机器啥网卡就改啥）
```

```
# 1 修改CIDR
sed -i 's/192\.168/10\.244/g' calico.yaml

sed -i 's/# \(- name: CALICO_IPV4POOL_CIDR\)/\1/' calico.yaml
sed -i 's/# \(\s*value: "10.244.0.0\/16"\)/\1/' calico.yaml
```

```
# 2 指定网卡（ens33为本地网卡名字（自己机器啥网卡就改啥））
sed -i '/value: "k8s,bgp"/a \            - name: IP_AUTODETECTION_METHOD'
\calico.yaml

sed -i '/- name: IP_AUTODETECTION_METHOD/a \              value:
"interface=ens33,ens160"' \calico.yaml
```

```
kubectl apply -f ~/calico-yml/calico.yaml
```

5、coredns 解析测试是否正常

```
[root@k8s-master ~]# kubectl run -it --rm dns-test --image=busybox:1.28.4 sh
If you don't see a command prompt, try pressing enter.
/ # nslookup kubernetes
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local   # 看到这个说明dns解
析正常

Name:      kubernetes
Address 1: 10.96.0.1 kubernetes.default.svc.cluster.local
/ #
```

```
kubectl run -it --rm dns-test --image=busybox:1.28.4 sh

kubectl run -it --rm dns-test --
image=ccr.ccs.tencentyun.com/huanghuanhui/busybox:1.28.4 sh
```

```
nslookup kubernetes
```

6、k8s-node节点后期的加入命令（按照上面操作安装好containerd、kubeadm、kubelet、kubectl）

```
kubeadm token list

kubeadm token create --print-join-command
```

1、helm、kubens、kubectl补全

helm

> https://github.com/helm/helm

```
cd && wget https://repo.huaweicloud.com/helm/v3.14.0/helm-v3.14.0-linux-amd64.tar.gz

tar xf ~/helm-v3.14.0-linux-amd64.tar.gz

cp ~/linux-amd64/helm /usr/local/sbin/helm

rm -rf ~/helm-v3.14.0-linux-amd64.tar.gz && rm -rf ~/linux-amd64

helm version
```

kubectx、kubens

```
wget -O /usr/local/sbin/kubens https://github.com/ahmetb/kubectx/raw/v0.9.5/kubens

chmod +x /usr/local/sbin/kubens

wget -O /usr/local/sbin/kubectx https://github.com/ahmetb/kubectx/raw/v0.9.5/kubectx

# chmod +x /usr/local/sbin/kubectx
```

kubectl 补全

```
yum -y install bash-completion

source /etc/profile.d/bash_completion.sh

echo "source <(crictl completion bash)" >> ~/.bashrc
echo "source <(kubectl completion bash)" >> ~/.bashrc
echo "source <(helm completion bash)" >> ~/.bashrc

source ~/.bashrc && su -
```

别名

```

```

```
cat >> ~/.bashrc << 'EOF'
alias pod='kubectl get pod'
alias po='kubectl get pod'
alias svc='kubectl get svc'
alias ns='kubectl get ns'
alias pvc='kubectl get pvc'
alias pv='kubectl get pv'
alias sc='kubectl get sc'
alias ingress='kubectl get ingress'
alias all='kubectl get all'
alias deployment='kubectl get deployments'
alias vs='kubectl get vs'
alias gateway='kubectl get gateway'
EOF

source ~/.bashrc
```

2、ingress-nginx

helm安装 ingress-nginx（k8s-master边缘节点）

> master（ingress-nginx边缘节点）
>
> chart version：4.9.0 （k8s：1.29、1.28、1.27、1.26、1.25）
>
> 当前版本：k8s-v1.29.1

https://kgithub.com/kubernetes/ingress-nginx

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm search repo ingress-nginx/ingress-nginx

helm pull ingress-nginx/ingress-nginx --version 4.9.0 --untar
```

```
cat > ~/ingress-nginx/values-prod.yaml << 'EOF'
controller:
  name: controller
  image:
    registry: dyrnq
    image: controller
    tag: "v1.9.5"
    digest:
    pullPolicy: IfNotPresent

  dnsPolicy: ClusterFirstWithHostNet

  hostNetwork: true

  publishService:  # hostNetwork 模式下设置为false，通过节点IP地址上报ingress status数据
    enabled: false
```

```
  kind: DaemonSet

  tolerations:   # kubeadm 安装的集群默认情况下 k8s-master 是有污点，需要容忍这个污点才可以部署
  - key: "node-role.kubernetes.io/k8s-master"
    operator: "Equal"
    effect: "NoSchedule"

  nodeSelector:   # 固定到k8s-master节点(自己master啥名字就写啥)
    kubernetes.io/hostname: "k8s-master"

  service:  # HostNetwork 模式不需要创建service
    enabled: false

  admissionWebhooks: # 强烈建议开启 admission webhook
    enabled: true
    patch:
      enabled: true
      image:
        registry: dyrnq
        image: kube-webhook-certgen
        tag: v20231226-1a7112e06
        digest:
        pullPolicy: IfNotPresent

defaultBackend:
  enabled: true
  name: defaultbackend
  image:
    registry: dyrnq
    image: defaultbackend-amd64
    tag: "1.5"
    digest:
    pullPolicy: IfNotPresent
EOF
```

```
kubectl create ns ingress-nginx

helm upgrade --install --namespace ingress-nginx ingress-nginx -f ./values-prod.yaml .
```

卸载

```
[root@k8s-master ~/ingress-nginx]# helm delete ingress-nginx -n ingress-nginx

[root@k8s-master ~/ingress-nginx]# kubectl delete ns ingress-nginx
```

3、istio

```
cd && wget https://github.com/istio/istio/releases/download/1.20.2/istio-1.20.2-linux-amd64.tar.gz
```

```
tar xf istio-1.20.2-linux-amd64.tar.gz

cp ~/istio-1.20.2/bin/istioctl /usr/bin/istioctl
```

```
# istioctl version
no ready Istio pods in "istio-system"
1.20.2
```

```
istioctl install --set profile=demo -y
```

```
# istioctl version
client version: 1.20.2
control plane version: 1.20.2
data plane version: 1.20.2 (2 proxies)
```

stioctl 命令补全

```
yum -y install bash-completion

source /etc/profile.d/bash_completion.sh

cp ~/istio-1.20.2/tools/istioctl.bash ~/.istioctl.bash

source ~/.istioctl.bash
```

4、Argo Rollouts

```
mkdir -p ~/argo-rollouts-yml

kubectl create ns argo-rollouts
```

```
cd ~/argo-rollouts-yml && wget https://github.com/argoproj/argo-rollouts/releases/download/v1.6.4/install.yaml

cd ~/argo-rollouts-yml && wget https://github.com/argoproj/argo-rollouts/releases/download/v1.6.4/dashboard-install.yaml
```

```
kubectl apply -n argo-rollouts -f ~/argo-rollouts-yml/install.yaml

kubectl apply -n argo-rollouts -f ~/argo-rollouts-yml/dashboard-install.yaml
```

```
curl -LO https://github.com/argoproj/argo-rollouts/releases/download/v1.6.4/kubectl-argo-rollouts-linux-amd64

chmod +x ./kubectl-argo-rollouts-linux-amd64

mv ./kubectl-argo-rollouts-linux-amd64 /usr/local/bin/kubectl-argo-rollouts

kubectl argo rollouts version
```

```
cat > ~/argo-rollouts-yml/argo-rollouts-dashboard-Ingress.yml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: argo-rollouts-dashboard-ingress
  namespace: argo-rollouts
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/proxy-body-size: '4G'
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: argo-rollouts-dashboard-auth
spec:
  ingressClassName: nginx
  rules:
  - host: argo-rollouts-dashboard.huanghuanhui.cloud
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: argo-rollouts-dashboard
            port:
              number: 3100

  tls:
  - hosts:
    - argo-rollouts-dashboard.huanghuanhui.cloud
    secretName: argo-rollouts-dashboard-ingress-tls
EOF
```

```
yum -y install httpd-tools

$ htpasswd -nb admin Admin@2024 > ~/argo-rollouts-yml/auth

kubectl create secret generic argo-rollouts-dashboard-auth --from-
file=/root/argo-rollouts-yml/auth -n argo-rollouts
```

```
kubectl create secret -n argo-rollouts \
tls argo-rollouts-dashboard-ingress-tls \
--key=/root/ssl/huanghuanhui.cloud.key \
--cert=/root/ssl/huanghuanhui.cloud.crt
```

```
kubectl apply -f ~/argo-rollouts-yml/argo-rollouts-dashboard-Ingress.yml
```

> 访问地址：kargo-rollouts-dashboard.huanghuanhui.cloud
>
> 用户密码：admin、Admin@2024

5、nfs-subdir-external-provisioner

k8s（pv 与 pvc）动态存储 StorageClass

k8s-1.29.1 持久化存储（nfs动态存储）

1、部署nfs

nfs 服务端（k8s-master）

```
# 所有服务端节点安装nfs
yum -y install nfs-utils

systemctl enable nfs-server rpcbind --now

# 创建nfs共享目录、授权
mkdir -p /data/k8s && chmod -R 777 /data/k8s

# 写入exports
cat > /etc/exports << EOF
/data/k8s 192.168.1.0/24(rw,sync,no_root_squash)
EOF

systemctl reload nfs-server

使用如下命令进行验证
# showmount -e 192.168.1.200
Export list for 192.168.1.200:
/data/k8s 192.168.1.0/24
```

nfs 客户端（k8s-node）

```
yum -y install nfs-utils

systemctl enable rpcbind --now

使用如下命令进行验证
# showmount -e 192.168.1.200
Export list for 192.168.1.200:
/data/k8s 192.168.1.0/24
```

备份

```
mkdir -p /data/k8s && chmod -R 777 /data/k8s

rsync -avzP /data/k8s root@192.168.1.203:/data

00 2 * * * rsync -avz /data/k8s root@192.168.1.203:/data &>/dev/null
```

2、动态创建 NFS存储（动态存储）

> https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner

```
mkdir ~/nfs-subdir-external-provisioner-4.0.18 && cd ~/nfs-subdir-external-provisioner-4.0.18
```

> 版本：nfs-subdir-external-provisioner-4.0.18
>
> https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner/tree/nfs-subdir-external-provisioner-4.0.18/deploy

```
wget https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner/raw/nfs-subdir-external-provisioner-4.0.18/deploy/deployment.yaml

wget https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner/raw/nfs-subdir-external-provisioner-4.0.18/deploy/rbac.yaml

wget https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner/raw/nfs-subdir-external-provisioner-4.0.18/deploy/class.yaml
```

```
# 1、修改镜像（默认谷歌k8s.gcr.io）
sed -i 's/registry.k8s.io\/sig-storage/dyrnq/g' deployment.yaml
# 2、修改nfs服务端地址
sed -i 's/10.3.243.101/192.168.1.200/g' deployment.yaml
# 3、修改存储地址（/data/k8s）
sed -i 's#\/ifs\/kubernetes#\/data\/k8s#g' deployment.yaml

sed -i 's#nfs-client#nfs-storage#g' class.yaml

sed -i 's/namespace: default/namespace: nfs-storage/g' rbac.yaml deployment.yaml
```

> 使用这个镜像：dyrnq/nfs-subdir-external-provisioner:v4.0.2
>
> dockerhub 地址：https://hub.docker.com/r/dyrnq/nfs-subdir-external-provisioner/tags

```
kubectl create ns nfs-storage

kubectl -n nfs-storage apply -f .

kubectl get pods -n nfs-storage -l app=nfs-client-provisioner

kubectl get storageclass
```

6、metrics-server

> https://github.com/kubernetes-sigs/metrics-server

> 版本：v0.7.0
>
> k8s-v1.29.1

Metrics Server	Metrics API group/version	Supported Kubernetes version
0.7.x	`metrics.k8s.io/v1beta1`	1.19+
0.6.x	`metrics.k8s.io/v1beta1`	1.19+
0.5.x	`metrics.k8s.io/v1beta1`	*1.8+

Metrics Server	Metrics API group/version	Supported Kubernetes version
0.4.x	`metrics.k8s.io/v1beta1`	*1.8+
0.3.x	`metrics.k8s.io/v1beta1`	1.8-1.21

```
mkdir -p ~/metrics-server
```

```
cd ~/metrics-server &&  wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

```
#1、添加"- --kubelet-insecure-tls"参数（匹配行后）
sed -i '/15s/a\        - --kubelet-insecure-tls' ~/metrics-server/components.yaml

#2、 修改镜像（默认谷歌k8s.gcr.io）
sed -i 's/registry.k8s.io\/metrics-server/dyrnq/g' ~/metrics-server/components.yaml
```

```
kubectl apply -f ~/metrics-server/components.yaml

kubectl get pods -n kube-system -l k8s-app=metrics-server
```

```
[root@k8s-master ~/metrics-server]# kubectl top node
NAME         CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
k8s-master   209m         5%     1993Mi          25%
k8s-node1    100m         1%     749Mi           4%
k8s-node2    57m          0%     802Mi           5%
k8s-node3    64m          0%     891Mi           5%
[root@k8s-master ~/metrics-server]# kubectl top pod
NAME                                       CPU(cores)   MEMORY(bytes)
calico-kube-controllers-5fc7d6cf67-c9l8v   1m           26Mi
calico-node-lgvrg                          25m          197Mi
calico-node-nsns8                          38m          170Mi
calico-node-z2lv4                          28m          193Mi
calico-node-zn4k5                          46m          178Mi
coredns-857d9ff4c9-l8ltv                   2m           22Mi
coredns-857d9ff4c9-v9bn2                   2m           24Mi
etcd-k8s-master                            30m          145Mi
kube-apiserver-k8s-master                  68m          841Mi
kube-controller-manager-k8s-master         23m          66Mi
kube-proxy-6h7k8                           1m           27Mi
kube-proxy-7kwdk                           5m           30Mi
kube-proxy-fqbpm                           6m           26Mi
kube-proxy-q868k                           11m          35Mi
kube-scheduler-k8s-master                  3m           23Mi
metrics-server-84957d8477-wmpwc            3m           18Mi
[root@k8s-master ~/metrics-server]# kubectl get node
NAME         STATUS   ROLES           AGE   VERSION
k8s-master   Ready    control-plane   13h   v1.29.1
k8s-node1    Ready    <none>          13h   v1.29.1
k8s-node2    Ready    <none>          13h   v1.29.1
k8s-node3    Ready    <none>          13h   v1.29.1
```

```
[root@k8s-master ~/metrics-server]#
```

7、gitlab

> 4c8g、100g

docker安装gitlab（使用k8s的ingress暴露）

版本：https://gitlab.com/gitlab-org/gitlab-foss/-/tags?sort=version_desc

官方docker仓库：https://hub.docker.com/r/gitlab/gitlab-ce/tags

```
docker pull gitlab/gitlab-ce:16.8.0-ce.0

docker pull ccr.ccs.tencentyun.com/huanghuanhui/gitlab:16.8.0-ce.0
```

```
cd && mkdir gitlab && cd gitlab && export GITLAB_HOME=/root/gitlab
```

```
docker run -d \
--name gitlab \
--hostname 'gitlab.huanghuanhui.cloud' \
--restart always \
--privileged=true \
-p 9797:80 \
-v $GITLAB_HOME/config:/etc/gitlab \
-v $GITLAB_HOME/logs:/var/log/gitlab \
-v $GITLAB_HOME/data:/var/opt/gitlab \
-e TIME_ZONE='Asia/Shanghai' \
ccr.ccs.tencentyun.com/huanghuanhui/gitlab:16.8.0-ce.0
```

初始化默认密码：

```
docker exec -it gitlab grep 'Password:' /etc/gitlab/initial_root_password
```

使用k8s的ingress暴露

```
mkdir -p ~/gitlab-yml

kubectl create ns gitlab
```

```
cat > ~/gitlab-yml/gitlab-endpoints.yml << 'EOF'
apiVersion: v1
kind: Endpoints
metadata:
  name: gitlab-service
  namespace: gitlab
subsets:
  - addresses:
      - ip: 192.168.1.201
    ports:
      - port: 9797
EOF
```

```
kubectl apply -f ~/gitlab-yml/gitlab-endpoints.yml
```

```
cat > ~/gitlab-yml/gitlab-Service.yml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: gitlab-service
  namespace: gitlab
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9797
EOF
```

```
kubectl apply -f ~/gitlab-yml/gitlab-Service.yml
```

```
cat > ~/gitlab-yml/gitlab-Ingress.yml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gitlab-ingress
  namespace: gitlab
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/proxy-body-size: '4G'
spec:
  ingressClassName: nginx
  rules:
  - host: gitlab.huanghuanhui.cloud
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: gitlab-service
            port:
              number: 80

  tls:
  - hosts:
    - gitlab.huanghuanhui.cloud
    secretName: gitlab-ingress-tls
EOF
```

```
kubectl create secret -n gitlab \
tls gitlab-ingress-tls \
--key=/root/ssl/huanghuanhui.cloud.key \
--cert=/root/ssl/huanghuanhui.cloud.crt
```

```
kubectl apply -f ~/gitlab-yml/gitlab-Ingress.yml
```

> 访问地址：gitlab.huanghuanhui.cloud
>
> https://gitlab.huanghuanhui.cloud/admin/users/root/edit
>
> 设置账号密码为：root、huanghuanhui@2024

计划任务备份

```
[root@gitlab ~]# crontab -l
0 0 * * * sync && echo 3 > /proc/sys/vm/drop_caches
0 0 * * * docker exec -t gitlab gitlab-backup create
```

8、harbor

> 2c4g、400g
>
> harbor（centos-7.9）（4c8g-400g）

```
echo "PS1='\[\033[35m\][\[\033[00m\]\[\033[31m\]\u\[\033[33m\]\[\033[33m\]@\
[\033[03m\]\[\033[35m\]\h\[\033[00m\] \[\033[5;32m\]\w\[\033[00m\]\[\033[35m\]]\
[\033[00m\]\[\033[5;31m\]\\$\[\033[00m\] '" >> ~/.bashrc && source ~/.bashrc

hostnamectl set-hostname harbor && su -
```

docker-compose安装harbor-v2.10.0

1、安装 docker

腾讯源

```
wget -O /etc/yum.repos.d/docker-ce.repo
https://download.docker.com/linux/centos/docker-ce.repo
```

```
sudo sed -i 's+download.docker.com+mirrors.cloud.tencent.com/docker-ce+'
/etc/yum.repos.d/docker-ce.repo
```

```
yum -y install docker-ce

systemctl enable docker
systemctl start docker
```

2、安装 docker-compose

官方文档：https://docs.docker.com/compose/install/

github：https://github.com/docker/compose/releases/

```
wget -O /usr/local/sbin/docker-compose
https://github.com/docker/compose/releases/download/v2.24.1/docker-compose-linux-x86_64

chmod +x /usr/local/sbin/docker-compose
```

3、安装 harbor

https://github.com/goharbor/harbor/releases （离线下载上传）

```
wget https://github.com/goharbor/harbor/releases/download/v2.10.0/harbor-offline-installer-v2.10.0.tgz
```

```
cd && tar xf harbor-offline-installer-v2.10.0.tgz -C /usr/local/
```

```
ls -la /usr/local/harbor/

cp /usr/local/harbor/harbor.yml.tmpl /usr/local/harbor/harbor.yml
```

```
修改配置文件:
#  harbor.yml
1、改成本机ip（域名）
hostname: harbor.huanghuanhui.cloud

2、修改https协议证书位置
https:
  port: 443
  certificate: /root/ssl/huanghuanhui.cloud.crt
  private_key: /root/ssl/huanghuanhui.cloud.key

3、修改登录密码（生产环境一定要修改）
harbor_admin_password: Admin@2024
```

```
sed -i.bak 's/reg\.mydomain\.com/harbor.huanghuanhui.cloud/g'
/usr/local/harbor/harbor.yml

sed -i 's#certificate: .*#certificate: /root/ssl/huanghuanhui.cloud.crt#g'
/usr/local/harbor/harbor.yml

sed -i 's#private_key: .*#private_key: /root/ssl/huanghuanhui.cloud.key#g'
/usr/local/harbor/harbor.yml

sed -i 's/Harbor12345/Admin@2024/g' /usr/local/harbor/harbor.yml
```

```
# ./install.sh（执行安装脚本）
/usr/local/harbor/install.sh
```

```
docker ps |grep harbor
```

> 访问地址：harbor.huanghuanhui.cloud

> 账号密码：admin、Admin@2024

9、jenkins

k8s手撕yml方式部署最新版 Jenkins 2.441（jdk-21版）（jenkins-prod）

```
mkdir -p ~/jenkins-prod-yml

kubectl create ns jenkins-prod
```

```
kubectl label node k8s-node1 jenkins-prod=jenkins-prod
```

```
cat > ~/jenkins-prod-yml/Jenkins-prod-rbac.yml << 'EOF'
apiVersion: v1
kind: Namespace
metadata:
  name: jenkins-prod
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins-prod
  namespace: jenkins-prod
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: jenkins-prod
rules:
- apiGroups:
  - '*'
  resources:
  - statefulsets
  - services
  - replicationcontrollers
  - replicasets
  - podtemplates
  - podsecuritypolicies
  - pods
  - pods/log
  - pods/exec
  - podpreset
  - poddisruptionbudget
  - persistentvolumes
  - persistentvolumeclaims
  - jobs
  - endpoints
  - deployments
  - deployments/scale
  - daemonsets
```

```
  - cronjobs
  - configmaps
  - namespaces
  - events
  - secrets
  verbs:
  - create
  - get
  - watch
  - delete
  - list
  - patch
  - update
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - update
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: jenkins-prod
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: jenkins-prod
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:jenkins-prod
EOF
```

```
kubectl apply -f ~/jenkins-prod-yml/Jenkins-prod-rbac.yml
```

```
cat > ~/jenkins-prod-yml/Jenkins-prod-Service.yml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: jenkins-prod
  namespace: jenkins-prod
  labels:
    app: jenkins-prod
spec:
  selector:
    app: jenkins-prod
```

```
  type: NodePort
  ports:
  - name: web
    nodePort: 30456
    port: 8080
    targetPort: web
  - name: agent
    nodePort: 30789
    port: 50000
    targetPort: agent
EOF
```

```
kubectl apply -f ~/jenkins-prod-yml/Jenkins-prod-Service.yml
```

```
cat > ~/jenkins-prod-yml/Jenkins-prod-Deployment.yml << 'EOF'
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins-prod
  namespace: jenkins-prod
  labels:
    app: jenkins-prod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jenkins-prod
  template:
    metadata:
      labels:
        app: jenkins-prod
    spec:
      tolerations:
      - effect: NoSchedule
        key: no-pod
        operator: Exists
      nodeSelector:
        jenkins-prod: jenkins-prod
      containers:
      - name: jenkins-prod
        #image: jenkins/jenkins:2.441-jdk21
        image: ccr.ccs.tencentyun.com/huanghuanhui/jenkins:2.441-jdk21
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            cpu: "2"
            memory: "4Gi"
          requests:
            cpu: "1"
            memory: "2Gi"
        securityContext:
          runAsUser: 0
        ports:
        - containerPort: 8080
```

```
          name: web
          protocol: TCP
        - containerPort: 50000
          name: agent
          protocol: TCP
        env:
        - name: LIMITS_MEMORY
          valueFrom:
            resourceFieldRef:
              resource: limits.memory
              divisor: 1Mi
        - name: JAVA_OPTS
          value: -
Dhudson.security.csrf.GlobalCrumbIssuerConfiguration.DISABLE_CSRF_PROTECTION=tru
e
        volumeMounts:
        - name: jenkins-home-prod
          mountPath: /var/jenkins_home
        - mountPath: /etc/localtime
          name: localtime
      volumes:
      - name: jenkins-home-prod
        persistentVolumeClaim:
          claimName: jenkins-home-prod
      - name: localtime
        hostPath:
          path: /etc/localtime

---
apiVersion: v1
kind:  PersistentVolumeClaim
metadata:
  name: jenkins-home-prod
  namespace: jenkins-prod
spec:
  storageClassName: "nfs-storage"
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 2Ti
EOF
```

```
kubectl apply -f ~/jenkins-prod-yml/Jenkins-prod-Deployment.yml
```

```
cat > ~/jenkins-prod-yml/Jenkins-prod-Ingress.yml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: jenkins-prod-ingress
  namespace: jenkins-prod
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/proxy-body-size: '4G'
spec:
```

```
  ingressClassName: nginx
  rules:
  - host: jenkins-prod.huanghuanhui.cloud
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: jenkins-prod # 将所有请求发送到 jenkins-prod 服务的 8080 端口
            port:
              number: 8080
  tls:
  - hosts:
    - jenkins-prod.huanghuanhui.cloud
    secretName: jenkins-prod-ingress-tls
EOF
```

```
kubectl create secret -n jenkins-prod \
tls jenkins-prod-ingress-tls \
--key=/root/ssl/huanghuanhui.cloud.key \
--cert=/root/ssl/huanghuanhui.cloud.crt
```

```
kubectl apply -f ~/jenkins-prod-yml/Jenkins-prod-Ingress.yml
```

> 访问地址：jenkins-prod.huanghuanhui.cloud
>
> 设置账号密码为：admin、Admin@2024

```
# 插件
1、Localization: Chinese (Simplified)
2、Pipeline
3、Kubernetes
4、Git
5、Git Parameter
6、GitLab                    # webhook 触发构建
7、Config FIle Provider      # 连接远程k8s集群
#8、Extended Choice Parameter
9、SSH Pipeline Steps        # Pipeline通过ssh远程执行命令
10、Pipeline: Stage View
11、Role-based Authorization Strategy
12、DingTalk                 # 钉钉机器人

http://jenkins-prod.jenkins-prod:8080
```

```
cat > ~/jenkins-prod-yml/Jenkins-prod-slave-maven-cache.yml << 'EOF'
apiVersion: v1
kind:  PersistentVolumeClaim
metadata:
  name: jenkins-prod-slave-maven-cache
  namespace: jenkins-prod
spec:
  storageClassName: "nfs-storage"
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 2Ti
EOF
```

```
cat > ~/jenkins-prod-yml/Jenkins-prod-slave-node-cache.yml << 'EOF'
apiVersion: v1
kind:  PersistentVolumeClaim
metadata:
  name: jenkins-prod-slave-node-cache
  namespace: jenkins-prod
spec:
  storageClassName: "nfs-storage"
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 2Ti
EOF
```

```
cat > ~/jenkins-prod-yml/Jenkins-prod-slave-golang-cache.yml << 'EOF'
apiVersion: v1
kind:  PersistentVolumeClaim
metadata:
  name: jenkins-prod-slave-golang-cache
  namespace: jenkins-prod
spec:
  storageClassName: "nfs-storage"
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 2Ti
EOF

cat > ~/jenkins-prod-yml/Jenkins-prod-slave-go-build-cache.yml << 'EOF'
apiVersion: v1
kind:  PersistentVolumeClaim
metadata:
  name: jenkins-prod-slave-go-build-cache
  namespace: jenkins-prod
spec:
  storageClassName: "nfs-storage"
  accessModes: [ReadWriteOnce]
  resources:
    requests:
      storage: 2Ti
```

```
EOF
```

Jenkins （jdk-21） （pipeline）

测试 docker、测试 maven、 测试 node、测试 golang、 测试 gcc、 测试 kubectl

```
#!/usr/bin/env groovy

pipeline {
    agent {
        kubernetes {
            yaml '''
apiVersion: v1
kind: Pod
metadata:
  name: jenkins-slave
  namespace: jenkins-prod
spec:
  tolerations:
  - key: "no-pod"
    operator: "Exists"
    effect: "NoSchedule"
  containers:
  - name: docker
    #image: docker:24.0.6
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6
    imagePullPolicy: IfNotPresent
    readinessProbe:
      exec:
        command: [sh, -c, "ls -S /var/run/docker.sock"]
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: docker-daemon
    #image: docker:24.0.6-dind
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6-dind
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: maven
    #image: maven:3.8.1-jdk-8
    image: ccr.ccs.tencentyun.com/huanghuanhui/maven:3.8.1-jdk-8
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
```

```
    - name: maven-cache
      mountPath: /root/.m2/repository
  - name: node
    #image: node:16.17.0-alpine
    image: ccr.ccs.tencentyun.com/huanghuanhui/node:16.17.0-alpine
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: node-cache
      mountPath: /root/.npm
  - name: golang
    #image: golang:1.21.3
    image: ccr.ccs.tencentyun.com/huanghuanhui/golang:1.21.3
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
  - name: gcc
    #image: gcc:13.2.0
    image: ccr.ccs.tencentyun.com/huanghuanhui/gcc:13.2.0
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
  - name: kubectl
    #image: kostiscodefresh/kubectl-argo-rollouts:v1.6.0
    #image: kubectl:v1.28.4
    #image: ccr.ccs.tencentyun.com/huanghuanhui/kubectl:v1.6.0
    image: ccr.ccs.tencentyun.com/huanghuanhui/kubectl:v1.28.4
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
  volumes:
  - name: docker-socket
    emptyDir: {}
  - name: maven-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-maven-cache
  - name: node-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-node-cache
'''
        }
    }

    stages {
        stage('测试 docker') {
            steps {
```

```
                container('docker') {
                  sh """
                    docker version
                  """
                  }
              }
          }

          stage('测试 maven') {
              steps {
                container('maven') {
                  sh """
                    mvn -version && java -version && javac -version
                  """
                  }
              }
          }

          stage('测试 node') {
              steps {
                container('node') {
                  sh """
                    node --version && npm --version && yarn --version
                  """
                  }
              }
          }

          stage('测试 golang') {
              steps {
                container('golang') {
                  sh """
                    go version
cat > HelloWorld.go << 'EOF'
package main

import "fmt"

func main() {
    fmt.Println("Hello, world! My Name is go!")
}
EOF

go build -o HelloWorld-go HelloWorld.go && ./HelloWorld-go
                  """
                  }
              }
          }

          stage('测试 gcc') {
              steps {
                container('gcc') {
                  sh """
                    gcc --version && g++ --version && make --version
cat > HelloWorld.cpp << 'EOF'
```

```
#include <iostream>

int main() {
    std::cout << "Hello, World! My Name is C++!" << std::endl;
    return 0;
}
EOF

g++ -o HelloWorld-cpp HelloWorld.cpp && ./HelloWorld-cpp
                """
                }
            }
        }

        stage('测试 kubectl') {
            steps {
              container('kubectl') {
                sh """
                  kubectl get node
                """
                }
            }
        }

    }
}
```

二、RuoYi-Cloud 业务组件

```
0、mysql-8.0.22

1、nacos-2.1.0

2、redis-7.2
```

0、mysql-8.0.28

k8s手撕yml方式

> 适合开发、测试环境
>
> 版本：mysql-8.0.28

```
mkdir -p ~/mysql-yml

kubectl create ns mysql
```

优化配置

```
cat > ~/mysql-yml/mysql-cm.yml << 'EOF'
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-config
```

```
  namespace: mysql
data:
  my.cnf: |
    [mysqld]
    pid-file        = /var/run/mysqld/mysqld.pid
    socket          = /var/run/mysqld/mysqld.sock
    datadir         = /var/lib/mysql
    secure-file-priv= NULL

    # Custom config should go here
    !includedir /etc/mysql/conf.d/

    # 优化配置
    # 设置最大连接数为 2500
    max_connections = 2500
    # 设置字符集为 UTF-8
    character-set-server=utf8mb4
    collation-server=utf8mb4_general_ci
    # 设置 InnoDB 引擎的缓冲区大小(InnoDB 缓冲池设置为内存的50%-75%)
    innodb_buffer_pool_size=4G
EOF
```

```
kubectl apply -f ~/mysql-yml/mysql-cm.yml
```

```
cat > ~/mysql-yml/mysql.yml << 'EOF'
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
  namespace: mysql
spec:
  serviceName: mysql-headless
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:8.0.28
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            cpu: 2
            memory: 4Gi
          requests:
            cpu: 2
            memory: 4Gi
        ports:
        - name: mysql
```

```
          containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "Admin@2024"
        volumeMounts:
        - name: mysql-data-pvc
          mountPath: /var/lib/mysql
        - name: mysql-config
          mountPath: /etc/mysql/my.cnf
          subPath: my.cnf
        - mountPath: /etc/localtime
          name: localtime
      volumes:
      - name: mysql-config
        configMap:
          name: mysql-config
      - name: localtime
        hostPath:
          path: /etc/localtime
  volumeClaimTemplates:
  - metadata:
      name: mysql-data-pvc
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: nfs-storage
      resources:
        requests:
          storage: 2Ti

---
apiVersion: v1
kind: Service
metadata:
  name: mysql-headless
  namespace: mysql
  labels:
    app: mysql
spec:
  clusterIP: None
  ports:
  - port: 3306
    name: mysql
    targetPort: 3306
  selector:
    app: mysql

---
apiVersion: v1
kind: Service
metadata:
  name: mysql
  namespace: mysql
  labels:
    app: mysql
spec:
```

```
  type: NodePort
  ports:
  - port: 3306
    targetPort: 3306
    nodePort: 30336
  selector:
    app: mysql
EOF
```

```
kubectl apply -f ~/mysql-yml/mysql.yml
```

> 代码连接地址：mysql-headless.mysql.svc.cluster.local:3306
>
> 访问地址：ip（192.168.1.200） + 端口（30336）
>
> 用户密码：root、Admin@2024

```
# 创建 RuoYi-Cloud 数据库并且导入数据
cd && git clone https://gitee.com/y_project/RuoYi-Cloud.git

mysql -h 192.168.1.200 -u root -P 30336 -pAdmin@2024 -e "create database \`ry-cloud\`;"
mysql -h 192.168.1.200 -u root -P 30336 -pAdmin@2024 ry-cloud < ry_20231130.sql

mysql -h 192.168.1.200 -u root -P 30336 -pAdmin@2024 -e "create database \`ry-config\`;"
mysql -h 192.168.1.200 -u root -P 30336 -pAdmin@2024 ry-cloud < ry_config_20231204.sql


mysql -h 192.168.1.200 -u root -P 30336 -pAdmin@2024 -e "show databases;"
```

1、nacos-2.1.0

```
mkdir -p ~/nacos-yml

kubectl create ns nacos
```

```
cat > ~/nacos-yml/nacos-mysql.yml << 'EOF'
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
  namespace: nacos
spec:
  serviceName: mysql-headless
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
```

```
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7.40
        imagePullPolicy: IfNotPresent
        resources:
          limits:
            cpu: "2"
            memory: "4Gi"
          requests:
            cpu: "2"
            memory: "4Gi"
        ports:
        - name: mysql
          containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "Admin@2024"
        - name: MYSQL_DATABASE
          value: "nacos"
        - name: MYSQL_USER
          value: "nacos"
        - name: MYSQL_PASSWORD
          value: "nacos@2024"
        volumeMounts:
        - name: nacos-mysql-data-pvc
          mountPath: /var/lib/mysql
        - mountPath: /etc/localtime
          name: localtime
      volumes:
      - name: localtime
        hostPath:
          path: /etc/localtime
  volumeClaimTemplates:
  - metadata:
      name: nacos-mysql-data-pvc
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: nfs-storage
      resources:
        requests:
          storage: 10Gi

---
apiVersion: v1
kind: Service
metadata:
  name: mysql-headless
  namespace: nacos
  labels:
    app: mysql
spec:
  clusterIP: None
  ports:
```

```
  - port: 3306
    name: mysql
    targetPort: 3306
  selector:
    app: mysql
EOF
```

```
kubectl apply -f ~/nacos-yml/nacos-mysql.yml
```

```
https://github.com/alibaba/nacos/blob/2.1.0/config/src/main/resources/META-INF/nacos-db.sql（sql地址）

cd ~/nacos-yml && wget https://github.com/alibaba/nacos/raw/2.1.0/config/src/main/resources/META-INF/nacos-db.sql

kubectl cp nacos-db.sql mysql-0:/
kubectl exec mysql-0 -- mysql -pAdmin@2024 -e "use nacos;source /nacos-db.sql;"

kubectl exec mysql-0 -- mysql -pAdmin@2024 -e "use nacos;show tables;"
```

```
cat > ~/nacos-yml/nacos-v2.1.0-yml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: nacos-headless
  namespace: nacos
  labels:
    app: nacos
spec:
  clusterIP: None
  ports:
    - port: 8848
      name: server
      targetPort: 8848
    - port: 9848
      name: client-rpc
      targetPort: 9848
    - port: 9849
      name: raft-rpc
      targetPort: 9849
    ## 兼容1.4.x版本的选举端口
    - port: 7848
      name: old-raft-rpc
      targetPort: 7848
  selector:
    app: nacos

---
apiVersion: v1
kind: Service
metadata:
  name: nacos
```

```
  namespace: nacos
  labels:
    app: nacos
spec:
  type: NodePort
  ports:
    - port: 8848
      name: server
      targetPort: 8848
      nodePort: 31000
    - port: 9848
      name: client-rpc
      targetPort: 9848
      nodePort: 32000
    - port: 9849
      name: raft-rpc
      nodePort: 32001
    ## 兼容1.4.x版本的选举端口
    - port: 7848
      name: old-raft-rpc
      targetPort: 7848
      nodePort: 30000
  selector:
    app: nacos

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nacos-cm
  namespace: nacos
data:
  mysql.host: "mysql-headless.nacos.svc.cluster.local"
  mysql.db.name: "nacos"
  mysql.port: "3306"
  mysql.user: "nacos"
  mysql.password: "nacos@2024"

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nacos
  namespace: nacos
spec:
  serviceName: nacos-headless
  replicas: 3
  template:
    metadata:
      labels:
        app: nacos
      annotations:
        pod.alpha.kubernetes.io/initialized: "true"
    spec:
      affinity:
```

```
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: "app"
                    operator: In
                    values:
                      - nacos-headless
              topologyKey: "kubernetes.io/hostname"
      containers:
        - name: k8snacos
          image: nacos/nacos-server:v2.1.0
          imagePullPolicy: IfNotPresent
          resources:
            limits:
              cpu: 2
              memory: 4Gi
            requests:
              cpu: 2
              memory: 4Gi
          ports:
            - containerPort: 8848
              name: client
            - containerPort: 9848
              name: client-rpc
            - containerPort: 9849
              name: raft-rpc
            - containerPort: 7848
              name: old-raft-rpc
          livenessProbe:
            httpGet:
              path: /nacos/actuator/health
              port: 8848
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /nacos/actuator/health
              port: 8848
            initialDelaySeconds: 30
            periodSeconds: 10
          env:
            - name: NACOS_REPLICAS
              value: "3"
            - name: MYSQL_SERVICE_HOST
              valueFrom:
                configMapKeyRef:
                  name: nacos-cm
                  key: mysql.host
            - name: MYSQL_SERVICE_DB_NAME
              valueFrom:
                configMapKeyRef:
                  name: nacos-cm
                  key: mysql.db.name
            - name: MYSQL_SERVICE_PORT
```

```
              valueFrom:
                configMapKeyRef:
                  name: nacos-cm
                  key: mysql.port
            - name: MYSQL_SERVICE_USER
              valueFrom:
                configMapKeyRef:
                  name: nacos-cm
                  key: mysql.user
            - name: MYSQL_SERVICE_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: nacos-cm
                  key: mysql.password
            - name: SPRING_DATASOURCE_PLATFORM
              value: "mysql"
            - name: MODE
              value: "cluster"
            - name: NACOS_SERVER_PORT
              value: "8848"
            - name: PREFER_HOST_MODE
              value: "hostname"
            - name: NACOS_SERVERS
              value: "nacos-0.nacos-headless.nacos.svc.cluster.local:8848 nacos-
1.nacos-headless.nacos.svc.cluster.local:8848 nacos-2.nacos-
headless.nacos.svc.cluster.local:8848"
  selector:
    matchLabels:
      app: nacos
EOF
```

```
kubectl apply -f ~/nacos-yml/nacos-v2.1.0-yml
```

```
cat > ~/nacos-yml/nacos-Ingress.yml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nacos-ingress
  namespace: nacos
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/proxy-body-size: '4G'
spec:
  ingressClassName: nginx
  rules:
  - host: nacos.huanghuanhui.cloud
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: nacos-headless
            port:
```

```
              number: 8848

  tls:
  - hosts:
    - nacos.huanghuanhui.cloud
    secretName: nacos-ingress-tls
EOF
```

```
kubectl create secret -n nacos \
tls nacos-ingress-tls \
--key=/root/ssl/huanghuanhui.cloud.key \
--cert=/root/ssl/huanghuanhui.cloud.crt
```

```
kubectl apply -f ~/nacos-yml/nacos-Ingress.yml
```

```
kubectl exec -it nacos-0 bash

# 进容器里面执行
curl -X POST 'http://nacos-
headless.nacos.svc.cluster.local:8848/nacos/v1/ns/instance?
serviceName=nacos.naming.serviceName&ip=20.18.7.10&port=8080'

# 容器外执行
curl -X POST 'http://192.168.1.200:31000/nacos/v1/ns/instance?
serviceName=nacos.naming.serviceName&ip=20.18.7.10&port=8080'
```

> 代码连接地址：nacos-headless.nacos.svc.cluster.local:8848
>
> 访问地址ip：http://192.168.1.200:31000/nacos/#/login
>
> 访问地址域名：https://nacos.huanghuanhui.cloud/nacos/#/login
>
> 默认用户密码：nacos、nacos
>
> 用户密码：nacos、nacos@2024

导入配置文件

1、ruoyi-gateway-dev.yml

```
spring:
  redis:
    host: redis.redis.svc.cluster.local
    port: 6379
    password: Admin@2024
  cloud:
    gateway:
      discovery:
        locator:
          lowerCaseServiceId: true
          enabled: true
      routes:
        - id: ruoyi-auth
          uri: lb://ruoyi-auth
          predicates:
```

```
            - Path=/auth/**
          filters:
            - CacheRequestFilter
            - ValidateCodeFilter
            - StripPrefix=1
        - id: ruoyi-gen
          uri: lb://ruoyi-gen
          predicates:
            - Path=/code/**
          filters:
            - StripPrefix=1
        - id: ruoyi-job
          uri: lb://ruoyi-job
          predicates:
            - Path=/schedule/**
          filters:
            - StripPrefix=1
        - id: ruoyi-system
          uri: lb://ruoyi-system
          predicates:
            - Path=/system/**
          filters:
            - StripPrefix=1
        - id: ruoyi-file
          uri: lb://ruoyi-file
          predicates:
            - Path=/file/**
          filters:
            - StripPrefix=1

security:
  captcha:
    enabled: true
    type: math
  xss:
    enabled: true
    excludeUrls:
      - /system/notice
  ignore:
    whites:
      - /auth/logout
      - /auth/login
      - /auth/register
      - /*/v2/api-docs
      - /csrf
```

2、ruoyi-auth-dev.yml

```
spring:
  redis:
    host: redis.redis.svc.cluster.local
    port: 6379
    password: Admin@2024
```

3、ruoyi-system-dev.yml

```
spring:
  redis:
    host: redis.redis.svc.cluster.local
    port: 6379
    password: Admin@2024
  datasource:
    druid:
      stat-view-servlet:
        enabled: true
        loginUsername: admin
        loginPassword: 123456
    dynamic:
      druid:
        initial-size: 5
        min-idle: 5
        maxActive: 20
        maxWait: 60000
        timeBetweenEvictionRunsMillis: 60000
        minEvictableIdleTimeMillis: 300000
        validationQuery: SELECT 1 FROM DUAL
        testWhileIdle: true
        testOnBorrow: false
        testOnReturn: false
        poolPreparedStatements: true
        maxPoolPreparedStatementPerConnectionSize: 20
        filters: stat,slf4j
        connectionProperties:
druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000
      datasource:
          master:
            driver-class-name: com.mysql.cj.jdbc.Driver
            url: jdbc:mysql://192.168.1.201:3306/ry-cloud?
useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL
=true&serverTimezone=GMT%2B8
            username: root
            password: Admin@2023

mybatis:
  typeAliasesPackage: com.ruoyi.system
  mapperLocations: classpath:mapper/**/*.xml

swagger:
  title: 系统模块接口文档
  license: Powered By ruoyi
  licenseUrl: https://ruoyi.vip
```

2、redis-7.2.4

（单机）

```
mkdir -p ~/redis-yml

kubectl create ns redis
```

```
cat > ~/redis-yml/redis-ConfigMap.yml << 'EOF'
kind: ConfigMap
apiVersion: v1
metadata:
  name: redis-cm
  namespace: redis
  labels:
    app: redis
data:
  redis.conf: |-
    dir /data
    port 6379
    bind 0.0.0.0
    appendonly yes
    protected-mode no
    requirepass Admin@2024
    pidfile /data/redis-6379.pid
    save 900 1
    save 300 10
    save 60 10000
    appendfsync always
EOF
```

> # 开启 RDB 持久化
>
> save 900 1 # 在900秒（15分钟）内，如果至少有1个 key 发生变化，则执行一次持久化
>
> save 300 10 # 在300秒（5分钟）内，如果至少有10个 key 发生变化，则执行一次持久化
>
> save 60 10000 # 在60秒（1分钟）内，如果至少有10000个 key 发生变化，则执行一次持久化
>
> 开启 AOF 持久化
>
> appendfsync always #每次写入都会立即同步到磁盘

```
kubectl apply -f ~/redis-yml/redis-ConfigMap.yml
```

```
cat > ~/redis-yml/redis-StatefulSet.yml << 'EOF'
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis
  namespace: redis
spec:
  replicas: 1
  serviceName: redis
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      name: redis
      labels:
        app: redis
    spec:
      affinity:
```

```
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchLabels:
                  app: redis
              topologyKey: kubernetes.io/hostname
      containers:
      - name: redis
        image: redis:7.2.4-alpine
        imagePullPolicy: IfNotPresent
        env:
        - name: TZ
          value: Asia/Shanghai
        command:
          - "sh"
          - "-c"
          - "redis-server /etc/redis/redis.conf"
        ports:
        - containerPort: 6379
          name: tcp-redis
          protocol: TCP
        resources:
          limits:
            cpu: "2"
            memory: "4Gi"
          requests:
            cpu: "1"
            memory: "2Gi"
        volumeMounts:
          - name: redis-data
            mountPath: /data
          - name: config
            mountPath: /etc/redis/redis.conf
            subPath: redis.conf
      volumes:
        - name: config
          configMap:
            name: redis-cm
  volumeClaimTemplates:
  - metadata:
      name: redis-data
    spec:
      storageClassName: "nfs-storage"
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 2Ti
EOF
```

```
kubectl apply -f ~/redis-yml/redis-StatefulSet.yml
```

```
cat > ~/redis-yml/redis-Service.yml << 'EOF'
apiVersion: v1
kind: Service
```

```
metadata:
  name: redis
  namespace: redis
spec:
  type: NodePort
  ports:
    - name: redis
      port: 6379
      targetPort: 6379
      protocol: TCP
      nodePort: 30078
  selector:
    app: redis
EOF
```

```
kubectl apply -f ~/redis-yml/redis-Service.yml
```

> 访问地址：ip：192.168.1.200（端口30078）
>
> 代码连接地址：redis.redis.svc.cluster.local:6379
>
> 密码：Admin@2024

三、gitlab（操作）配置webhook钩子（触发构建）

```
https://gitlab.huanghuanhui.cloud/admin/application_settings/network
Outbound requests

https://gitlab.huanghuanhui.cloud/admin/application_settings/general
Import and export settings
Visibility and access controls

# 导入gitlab
https://gitee.com/y_project/RuoYi-Cloud.git

# 配置webhook钩子（触发构建）
https://jenkins-prod.huanghuanhui.cloud/project/ruoyi-auth
```

四、harbor（操作）

```
# 创建仓库
ruoyi-cloud

ruoyi-gateway
ruoyi-auth
ruoyi-system
ruoyi-vue

openjdk
```

```
docker login https://harbor.huanghuanhui.cloud --username=admin

Admin@2024

docker pull openjdk:8-jre
docker tag openjdk:8-jre harbor.huanghuanhui.cloud/openjdk/openjdk:8-jre
docker push harbor.huanghuanhui.cloud/openjdk/openjdk:8-jre
```

五、Jenkins-pipeline（操作）

```
1、创建 RuoYi-Cloud   RuoYi-Vue
2、创建 git_auth、harbor_auth 秘钥
3、创建 kubeconfig
4、创建 token （webhook自动触发）
webhook  11198baeb81a20a9b734b9ece849dcb541

5、匿名用户具有可读权限（webhook需要用到）
https://jenkins-prod.huanghuanhui.cloud/manage/configureSecurity/
```

后端（pipeline）

1、ruoyi-gateway-pipeline

```
# 参数化构建
AppName
服务名称
ruoyi-gateway

GitRepo
代码仓库
http://gitlab.huanghuanhui.cloud/root/RuoYi-Cloud.git

GitBranch
master
代码分支

HarborUrl
镜像仓库地址
harbor.huanghuanhui.cloud

Image
基础镜像
ccr.ccs.tencentyun.com/huanghuanhui/openjdk:8-jre

JAVA_OPTS
jar 运行时的参数配置
-Xms1024M -Xmx1024M -Xmn256M -Dspring.config.location=app.yml -
Dserver.tomcat.max-threads=800
```

```
#!/usr/bin/env groovy

def git_auth = "77066368-e8a8-4edb-afaf-53aaf90c31a9"
```

```
def harbor_auth = "9c10572f-c324-422f-b0c0-1b80d2ddb857"
def kubectl_auth = "c26898c2-92c3-4c19-8490-9cf8ff7918ef"

pipeline {
    agent {
        kubernetes {
            yaml '''
apiVersion: v1
kind: Pod
metadata:
  name: jenkins-slave
  namespace: jenkins-prod
spec:
  tolerations:
  - key: "no-pod"
    operator: "Exists"
    effect: "NoSchedule"
  containers:
  - name: docker
    #image: docker:24.0.6
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6
    imagePullPolicy: IfNotPresent
    readinessProbe:
      exec:
        command: [sh, -c, "ls -S /var/run/docker.sock"]
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: docker-daemon
    #image: docker:24.0.6-dind
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6-dind
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: maven
    #image: maven:3.8.1-jdk-8
    image: ccr.ccs.tencentyun.com/huanghuanhui/maven:3.8.1-jdk-8
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: maven-cache
      mountPath: /root/.m2/repository
  - name: kubectl
    image: kostiscodefresh/kubectl-argo-rollouts:v1.6.0
    imagePullPolicy: IfNotPresent
```

```
    command:
    - sleep
    args:
    - 99d
  volumes:
  - name: docker-socket
    emptyDir: {}
  - name: maven-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-maven-cache
  - name: node-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-node-cache
'''
        }
    }

environment {
AppName = "${AppName}"
GitRepo = "${GitRepo}"
GitBranch = "${GitBranch}"
HarborUrl = "${HarborUrl}"
Image = "${Image}"
JAVA_OPTS = "${JAVA_OPTS}"
}

    stages {
        stage('拉取代码') {
            steps {
            git branch: "${GitBranch}", credentialsId: "${git_auth}", url:
"${GitRepo}"
            }
        }

        stage('代码编译') {
            steps {
              container('maven') {
                sh """
                  mvn -U clean install -Dmaven.test.skip=true
                """
                }
            }
        }

        stage('打包镜像') {
            steps {
              script {env.GIT_COMMIT_MSG = sh (script: 'git rev-parse --short
HEAD', returnStdout: true).trim()}
              container('docker') {
sh '''cat > entrypoint.sh << EOF
#! /bash/bin -e
env
java $JAVA_OPTS -jar ./*.jar
EOF
```

```
cat > app.yml << EOF
# Tomcat
server:
  port: 8080

# Spring
spring:
  application:
    # 应用名称
    name: ${AppName}
  profiles:
    # 环境配置
    active: dev
  cloud:
    nacos:
      discovery:
        # 服务注册地址
        server-addr: nacos-headless.nacos.svc.cluster.local:8848
      config:
        # 配置中心地址
        server-addr: nacos-headless.nacos.svc.cluster.local:8848
        # 配置文件格式
        file-extension: yml
        # 共享配置
        shared-configs:
          - application
    sentinel:
      # 取消控制台懒加载
      eager: true
      transport:
        # 控制台地址
        dashboard: 127.0.0.1:8718
      # nacos配置持久化
      datasource:
        ds1:
          nacos:
            server-addr: 127.0.0.1:8848
            dataId: sentinel-ruoyi-gateway
            groupId: DEFAULT_GROUP
            data-type: json
            rule-type: gw-flow
EOF

cat > Dockerfile << EOF
FROM ${Image}
WORKDIR /usr/local/src/
ADD ./ruoyi-gateway/target/ruoyi-gateway.jar /usr/local/src/ruoyi-gateway.jar
ADD app.yml .
ADD entrypoint.sh .
ENTRYPOINT ["sh","./entrypoint.sh"]
EOF

docker build -t ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID} .
'''
```

```
                }
            }
        }

        stage('推送镜像') {
            steps {
              container('docker') {
                withCredentials([usernamePassword(credentialsId:
"${harbor_auth}", passwordVariable: 'password', usernameVariable: 'username')])
{
                sh """
                docker login -u ${username} -p '${password}'
harbor.huanghuanhui.cloud
                docker push ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}
                """
                   }
                }
            }
        }

        stage('argo-rollouts + istio（金丝雀发布）（渐进式交付）') {
            steps {
              container('kubectl') {
              configFileProvider([configFile(fileId: "${kubectl_auth}",
variable: 'kubeconfig')]) {
                sh """
                mkdir -p ~/.kube && cp ${kubeconfig} ~/.kube/config
                 /app/kubectl-argo-rollouts-linux-amd64 set image ${AppName}
"*=${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}" -n ruoyi
                """
                   }
                }
            }
        }
    }
    post {
        success {
            dingtalk (
                robot: "Jenkins-Dingtalk",
                type:'ACTION_CARD',
                atAll: false,
                title: "构建成功：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=8CE600 >成功</font>',
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
```

```
                ]
            )
        }

        failure {
            dingtalk (
                robot: "Jenkins-Dingtalk",
                type:'ACTION_CARD',
                atAll: false,
                title: "构建失败：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=#EE0000 >失败</font>',
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
                ]
            )
        }

    }

}
```

2、ruoyi-auth-pipeline

```
# 参数化构建
AppName
服务名称
ruoyi-auth

GitRepo
代码仓库
http://gitlab.huanghuanhui.cloud/root/RuoYi-Cloud.git

GitBranch
master
代码分支

HarborUrl
镜像仓库地址
harbor.huanghuanhui.cloud

Image
基础镜像
ccr.ccs.tencentyun.com/huanghuanhui/openjdk:8-jre

JAVA_OPTS
jar 运行时的参数配置
```

```
-Xms1024M -Xmx1024M -Xmn256M -Dspring.config.location=app.yml -
Dserver.tomcat.max-threads=800
```

```
#!/usr/bin/env groovy

def git_auth = "77066368-e8a8-4edb-afaf-53aaf90c31a9"
def harbor_auth = "9c10572f-c324-422f-b0c0-1b80d2ddb857"
def kubectl_auth = "c26898c2-92c3-4c19-8490-9cf8ff7918ef"

pipeline {
    agent {
        kubernetes {
            yaml '''
apiVersion: v1
kind: Pod
metadata:
  name: jenkins-slave
  namespace: jenkins-prod
spec:
  tolerations:
  - key: "no-pod"
    operator: "Exists"
    effect: "NoSchedule"
  containers:
  - name: docker
    #image: docker:24.0.6
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6
    imagePullPolicy: IfNotPresent
    readinessProbe:
      exec:
        command: [sh, -c, "ls -S /var/run/docker.sock"]
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: docker-daemon
    #image: docker:24.0.6-dind
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6-dind
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: maven
    #image: maven:3.8.1-jdk-8
    image: ccr.ccs.tencentyun.com/huanghuanhui/maven:3.8.1-jdk-8
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
```

```
    volumeMounts:
    - name: maven-cache
      mountPath: /root/.m2/repository
  - name: kubectl
    image: kostiscodefresh/kubectl-argo-rollouts:v1.6.0
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
  volumes:
  - name: docker-socket
    emptyDir: {}
  - name: maven-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-maven-cache
  - name: node-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-node-cache
'''
        }
    }

environment {
AppName = "${AppName}"
GitRepo = "${GitRepo}"
GitBranch = "${GitBranch}"
HarborUrl = "${HarborUrl}"
Image = "${Image}"
JAVA_OPTS = "${JAVA_OPTS}"
}

    stages {
        stage('拉取代码') {
            steps {
            git branch: "${GitBranch}", credentialsId: "${git_auth}", url:
"${GitRepo}"
            }
        }

        stage('代码编译') {
            steps {
              container('maven') {
                sh """
                  mvn -U clean install -Dmaven.test.skip=true
                """
                }
            }
        }

        stage('打包镜像') {
            steps {
              script {env.GIT_COMMIT_MSG = sh (script: 'git rev-parse --short
HEAD', returnStdout: true).trim()}
              container('docker') {
```

```
sh '''cat > entrypoint.sh << EOF
#! /bash/bin -e
env
java $JAVA_OPTS -jar ./*.jar
EOF

cat > app.yml << EOF
# Tomcat
server:
  port: 9200

# Spring
spring:
  application:
    # 应用名称
    name: ruoyi-auth
  profiles:
    # 环境配置
    active: dev
  cloud:
    nacos:
      discovery:
        # 服务注册地址
        server-addr: nacos-headless.nacos.svc.cluster.local:8848
      config:
        # 配置中心地址
        server-addr: nacos-headless.nacos.svc.cluster.local:8848
        # 配置文件格式
        file-extension: yml
        # 共享配置
        shared-configs:
          - application
EOF

cat > Dockerfile << EOF
FROM ${Image}
WORKDIR /usr/local/src/
ADD ./ruoyi-auth/target/ruoyi-auth.jar /usr/local/src/ruoyi-auth.jar
ADD app.yml .
ADD entrypoint.sh .
ENTRYPOINT ["sh","./entrypoint.sh"]
EOF

docker build -t ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID} .
'''
                }
            }
        }

        stage('推送镜像') {
            steps {
              container('docker') {
```

```
                withCredentials([usernamePassword(credentialsId:
"${harbor_auth}", passwordVariable: 'password', usernameVariable: 'username')])
{
                sh """
                docker login -u ${username} -p '${password}'
harbor.huanghuanhui.cloud
                docker push ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}
                """
                   }
                }
            }
        }

        stage('argo-rollouts + istio（金丝雀发布）（渐进式交付）') {
            steps {
              container('kubectl') {
              configFileProvider([configFile(fileId: "${kubectl_auth}",
variable: 'kubeconfig')]) {
                sh """
                mkdir -p ~/.kube && cp ${kubeconfig} ~/.kube/config
                 /app/kubectl-argo-rollouts-linux-amd64 set image ${AppName}
"*=${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}" -n ruoyi
                """
                   }
                }
            }
        }
    }
    post {
        success {
            dingtalk (
                robot: "Jenkins-Dingtalk",
                type:'ACTION_CARD',
                atAll: false,
                title: "构建成功：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=8CE600 >成功</font>',
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
                ]
           )
        }

        failure {
            dingtalk (
                robot: "Jenkins-Dingtalk",
```

```
                type:'ACTION_CARD',
                atAll: false,
                title: "构建失败：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=#EE0000 >失败</font>',
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
                ]
            )
        }

    }

}
```

3、ruoyi-system-pipeline

```
# 参数化构建
AppName
服务名称
ruoyi-system

GitRepo
代码仓库
http://gitlab.huanghuanhui.cloud/root/RuoYi-Cloud.git

GitBranch
master
代码分支

HarborUrl
镜像仓库地址
harbor.huanghuanhui.cloud

Image
基础镜像
ccr.ccs.tencentyun.com/huanghuanhui/openjdk:8-jre

JAVA_OPTS
jar 运行时的参数配置
-Xms1024M -Xmx1024M -Xmn256M -Dspring.config.location=app.yml -
Dserver.tomcat.max-threads=800
```

```
#!/usr/bin/env groovy

def git_auth = "77066368-e8a8-4edb-afaf-53aaf90c31a9"
def harbor_auth = "9c10572f-c324-422f-b0c0-1b80d2ddb857"
```

```
def kubectl_auth = "c26898c2-92c3-4c19-8490-9cf8ff7918ef"

pipeline {
    agent {
        kubernetes {
            yaml '''
apiVersion: v1
kind: Pod
metadata:
  name: jenkins-slave
  namespace: jenkins-prod
spec:
  tolerations:
  - key: "no-pod"
    operator: "Exists"
    effect: "NoSchedule"
  containers:
  - name: docker
    #image: docker:24.0.6
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6
    imagePullPolicy: IfNotPresent
    readinessProbe:
      exec:
        command: [sh, -c, "ls -S /var/run/docker.sock"]
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: docker-daemon
    #image: docker:24.0.6-dind
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6-dind
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: maven
    #image: maven:3.8.1-jdk-8
    image: ccr.ccs.tencentyun.com/huanghuanhui/maven:3.8.1-jdk-8
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: maven-cache
      mountPath: /root/.m2/repository
  - name: kubectl
    image: kostiscodefresh/kubectl-argo-rollouts:v1.6.0
    imagePullPolicy: IfNotPresent
    command:
```

```
    - sleep
    args:
    - 99d
  volumes:
  - name: docker-socket
    emptyDir: {}
  - name: maven-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-maven-cache
  - name: node-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-node-cache
'''
        }
    }

environment {
AppName = "${AppName}"
GitRepo = "${GitRepo}"
GitBranch = "${GitBranch}"
HarborUrl = "${HarborUrl}"
Image = "${Image}"
JAVA_OPTS = "${JAVA_OPTS}"
}

    stages {
        stage('拉取代码') {
            steps {
            git branch: "${GitBranch}", credentialsId: "${git_auth}", url:
"${GitRepo}"
            }
        }

        stage('代码编译') {
            steps {
              container('maven') {
                sh """
                  mvn -U clean install -Dmaven.test.skip=true
                """
                }
            }
        }

        stage('打包镜像') {
            steps {
              script {env.GIT_COMMIT_MSG = sh (script: 'git rev-parse --short
HEAD', returnStdout: true).trim()}
              container('docker') {
sh '''cat > entrypoint.sh << EOF
#! /bash/bin -e
env
java $JAVA_OPTS -jar ./*.jar
EOF
cat > app.yml << EOF
# Tomcat
```

```
server:
  port: 9201

# Spring
spring:
  application:
    # 应用名称
    name: $AppName
  profiles:
    # 环境配置
    active: dev
  cloud:
    nacos:
      discovery:
        # 服务注册地址
        server-addr: nacos-headless.nacos.svc.cluster.local:8848
      config:
        # 配置中心地址
        server-addr: nacos-headless.nacos.svc.cluster.local:8848
        # 配置文件格式
        file-extension: yml
        # 共享配置
        shared-configs:
          - application
EOF

cat > Dockerfile << EOF
FROM ${Image}
WORKDIR /usr/local/src/
ADD ./ruoyi-modules/ruoyi-system/target/ruoyi-modules-system.jar
/usr/local/src/ruoyi-modules-system.jar
ADD app.yml .
ADD entrypoint.sh .
ENTRYPOINT ["sh","./entrypoint.sh"]
EOF

docker build -t ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID} .
'''
                }
            }
        }

        stage('推送镜像') {
            steps {
              container('docker') {
                withCredentials([usernamePassword(credentialsId:
"${harbor_auth}", passwordVariable: 'password', usernameVariable: 'username')])
{
                sh """
                docker login -u ${username} -p '${password}'
harbor.huanghuanhui.cloud
                docker push ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}
                """
```

```
                    }
                }
            }
        }

        stage('argo-rollouts + istio（金丝雀发布）（渐进式交付）') {
            steps {
              container('kubectl') {
              configFileProvider([configFile(fileId: "${kubectl_auth}",
variable: 'kubeconfig')]) {
                sh """
                mkdir -p ~/.kube && cp ${kubeconfig} ~/.kube/config
                 /app/kubectl-argo-rollouts-linux-amd64 set image ${AppName}
"*=${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}" -n ruoyi
                """
                    }
                }
            }
        }
    }
    post {
        success {
            dingtalk (
                robot: "Jenkins-Dingtalk",
                type:'ACTION_CARD',
                atAll: false,
                title: "构建成功：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=8CE600 >成功</font>',
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
                ]
           )
        }

        failure {
            dingtalk (
                robot: "Jenkins-Dingtalk",
                type:'ACTION_CARD',
                atAll: false,
                title: "构建失败：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=#EE0000 >失败</font>',
```

```
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
                ]
            )
        }

    }

}
```

前端（pipeline）

4、ruoyi-vue-pipeline

```
# 参数化构建
AppName
服务名称
ruoyi-vue

GitRepo
代码仓库
http://gitlab.huanghuanhui.cloud/root/RuoYi-Cloud.git

GitBranch
master
代码分支

HarborUrl
镜像仓库地址
harbor.huanghuanhui.cloud

Image
基础镜像
ccr.ccs.tencentyun.com/huanghuanhui/nginx:1.25.3-alpine
```

```
#!/usr/bin/env groovy

def git_auth = "77066368-e8a8-4edb-afaf-53aaf90c31a9"
def harbor_auth = "9c10572f-c324-422f-b0c0-1b80d2ddb857"
def kubectl_auth = "c26898c2-92c3-4c19-8490-9cf8ff7918ef"

pipeline {
    agent {
        kubernetes {
            yaml '''
apiVersion: v1
kind: Pod
metadata:
  name: jenkins-slave
  namespace: jenkins-prod
spec:
```

```
  tolerations:
  - key: "no-pod"
    operator: "Exists"
    effect: "NoSchedule"
  containers:
  - name: docker
    #image: docker:24.0.6
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6
    imagePullPolicy: IfNotPresent
    readinessProbe:
      exec:
        command: [sh, -c, "ls -S /var/run/docker.sock"]
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: docker-daemon
    #image: docker:24.0.6-dind
    image: ccr.ccs.tencentyun.com/huanghuanhui/docker:24.0.6-dind
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: true
    volumeMounts:
    - name: docker-socket
      mountPath: /var/run
  - name: node
    #image: node:16.17.0-alpine
    image: ccr.ccs.tencentyun.com/huanghuanhui/node:16.17.0-alpine
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
    volumeMounts:
    - name: node-cache
      mountPath: /root/.npm
  - name: kubectl
    image: kostiscodefresh/kubectl-argo-rollouts:v1.6.0
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    args:
    - 99d
  volumes:
  - name: docker-socket
    emptyDir: {}
  - name: maven-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-maven-cache
  - name: node-cache
    persistentVolumeClaim:
      claimName: jenkins-prod-slave-node-cache
```

```
'''
        }
    }

environment {
AppName = "${AppName}"
GitRepo = "${GitRepo}"
GitBranch = "${GitBranch}"
HarborUrl = "${HarborUrl}"
Image = "${Image}"
JAVA_OPTS = "${JAVA_OPTS}"
}

    stages {
        stage('拉取代码') {
            steps {
            git branch: "${GitBranch}", credentialsId: "${git_auth}", url:
"${GitRepo}"
            }
        }

        stage('代码编译') {
            steps {
              container('node') {
                sh """
                  cd ruoyi-ui && sed -i \'s/localhost/ruoyi-gateway-svc/g\'
vue.config.js && npm install --registry=https://registry.npmmirror.com && npm
run build:prod
                """
                }
            }
        }

        stage('打包镜像') {
            steps {
              script {env.GIT_COMMIT_MSG = sh (script: 'git rev-parse --short
HEAD', returnStdout: true).trim()}
              container('docker') {
sh '''cat > nginx.conf << 'EOF'
worker_processes  auto;

events {
    worker_connections  10240;
}

http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;
```

```
        location / {
            root   /usr/share/nginx/html;
            try_files $uri $uri/ /index.html;
            index  index.html index.htm;
        }

        location /prod-api/{
            proxy_pass http://ruoyi-gateway-svc:8080/;
            proxy_set_header Host $http_host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header REMOTE-HOST $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_http_version 1.1;
        }

        # 避免actuator暴露
        if ($request_uri ~ "/actuator") {
            return 403;
        }

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
EOF

cat > dockerfile << 'EOF'
FROM ccr.ccs.tencentyun.com/huanghuanhui/nginx:1.25.3-alpine

WORKDIR /usr/share/nginx/html

COPY nginx.conf /etc/nginx/nginx.conf

COPY ./ruoyi-ui/dist /usr/share/nginx/html
EOF

docker build -t ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID} .
'''
                }
            }
        }

        stage('推送镜像') {
            steps {
              container('docker') {
                withCredentials([usernamePassword(credentialsId:
"${harbor_auth}", passwordVariable: 'password', usernameVariable: 'username')])
{
                sh """
                docker login -u ${username} -p '${password}'
harbor.huanghuanhui.cloud
```

```
                docker push ${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}
                """
                   }
                }
            }
        }

        stage('argo-rollouts + istio（金丝雀发布）（渐进式交付）') {
            steps {
              container('kubectl') {
              configFileProvider([configFile(fileId: "${kubectl_auth}",
variable: 'kubeconfig')]) {
                sh """
                mkdir -p ~/.kube && cp ${kubeconfig} ~/.kube/config
                 /app/kubectl-argo-rollouts-linux-amd64 set image ${AppName}
"*=${HarborUrl}/ruoyi-
cloud/${AppName}:${GitBranch}-${GIT_COMMIT_MSG}-${BUILD_ID}" -n ruoyi
                """
                   }
                }
            }
        }
    }
    post {
        success {
            dingtalk (
                robot: "Jenkins-Dingtalk",
                type:'ACTION_CARD',
                atAll: false,
                title: "构建成功：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=8CE600 >成功</font>',
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
                ]
           )
        }

        failure {
            dingtalk (
                robot: "Jenkins-Dingtalk",
                type:'ACTION_CARD',
                atAll: false,
                title: "构建失败：${env.JOB_NAME}",
                //messageUrl: 'xxxx',
                text: [
                    "### [${env.JOB_NAME}](${env.JOB_URL}) ",
```

```
                    '---',
                    "- 任务：[${currentBuild.displayName}](${env.BUILD_URL})",
                    '- 状态：<font color=#EE0000 >失败</font>',
                    "- 持续时间：${currentBuild.durationString}".split("and
counting")[0],
                    "- 执行人：${currentBuild.buildCauses.shortDescription}",
                    "- 环境：开发环境",
                    "- 构建日志地址：${BUILD_URL}console",
                ]
            )
        }

    }

}
```

六、Argo-Rollouts+ istio 部署前后端服务

```
mkdir -p ~/RuoYi-Cloud-rollout-yml

cd ~/RuoYi-Cloud-rollout-yml

kubectl create namespace ruoyi

kubectl label namespace ruoyi istio-injection=enabled
```

1、ruoyi-gateway

```
cat > ruoyi-gateway-rollout.yml << 'EOF'
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: ruoyi-gateway
  namespace: ruoyi
spec:
  replicas: 3
  strategy:
    canary:
      steps:
      - setWeight: 20
      - pause: {} # 人工卡点
      - setWeight: 40
      - pause: {duration: 10}
      - setWeight: 60
      - pause: {duration: 10}
      - setWeight: 80
      - pause: {duration: 10}
      - setWeight: 100
      - pause: {} # 人工卡点
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: ruoyi-gateway
  template:
```

```
    metadata:
      labels:
        app: ruoyi-gateway
    spec:
      containers:
      - name: ruoyi-gateway
        image: harbor.huanghuanhui.cloud/ruoyi-gateway/ruoyi-gateway:master-
78e61d8-1
        ports:
        - name: http
          containerPort: 8080
          protocol: TCP
EOF
```

```
cat > ruoyi-gateway-svc.yml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: ruoyi-gateway-svc
  namespace: ruoyi
  labels:
    app: ruoyi-gateway
spec:
  type: ClusterIP
  ports:
  - port: 8080
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: ruoyi-gateway
EOF
```

2、ruoyi-auth

```
cat > ruoyi-auth-rollout.yml << 'EOF'
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: ruoyi-auth
  namespace: ruoyi
spec:
  replicas: 3
  strategy:
    canary:
      steps:
      - setWeight: 20
      - pause: {} # 人工卡点
      - setWeight: 40
      - pause: {duration: 10}
      - setWeight: 60
      - pause: {duration: 10}
      - setWeight: 80
      - pause: {duration: 10}
```

```
      - setWeight: 100
      - pause: {} # 人工卡点
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: ruoyi-auth
  template:
    metadata:
      labels:
        app: ruoyi-auth
    spec:
      containers:
      - name: ruoyi-auth
        image: harbor.huanghuanhui.cloud/ruoyi-auth/ruoyi-auth:master-78e61d8-2
        ports:
        - name: http
          containerPort: 9200
          protocol: TCP
EOF
```

3、ruoyi-system

```
cat > ruoyi-system-rollout.yml << 'EOF'
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: ruoyi-system
  namespace: ruoyi
spec:
  replicas: 3
  strategy:
    canary:
      steps:
      - setWeight: 20
      - pause: {} # 人工卡点
      - setWeight: 40
      - pause: {duration: 10}
      - setWeight: 60
      - pause: {duration: 10}
      - setWeight: 80
      - pause: {duration: 10}
      - setWeight: 100
      - pause: {} # 人工卡点
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: ruoyi-system
  template:
    metadata:
      labels:
        app: ruoyi-system
    spec:
      containers:
      - name: ruoyi-system
```

```
        image: harbor.huanghuanhui.cloud/ruoyi-system/ruoyi-system:master-
78e61d8-2
        ports:
        - name: http
          containerPort: 9201
          protocol: TCP
EOF
```

4、ruoyi-vue

```
cat > ruoyi-vue-rollout.yml << 'EOF'
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: ruoyi-vue
  namespace: ruoyi
spec:
  replicas: 3
  strategy:
    canary:
      steps:
      - setWeight: 20
      - pause: {} # 人工卡点
      - setWeight: 40
      - pause: {duration: 10}
      - setWeight: 60
      - pause: {duration: 10}
      - setWeight: 80
      - pause: {duration: 10}
      - setWeight: 100
      - pause: {} # 人工卡点
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: ruoyi-vue
  template:
    metadata:
      labels:
        app: ruoyi-vue
    spec:
      containers:
      - name: ruoyi-vue
        image: harbor.huanghuanhui.cloud/ruoyi-cloud/ruoyi-vue:master-78e61d8-2
        imagePullPolicy: Always
        ports:
        - name: http
          containerPort: 80
          protocol: TCP
EOF
```

```
cat > ruoyi-vue-svc.yml << 'EOF'
apiVersion: v1
kind: Service
metadata:
```

```
  name: ruoyi-vue-svc
  namespace: ruoyi
  labels:
    app: ruoyi-vue
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: ruoyi-vue
EOF
```

配置域名

```
cat > ~/RuoYi-Cloud-rollout-yml/ruoyi-vue-Ingress.yml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ruoyi-vue-ingress
  namespace: ruoyi
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/proxy-body-size: '4G'
spec:
  ingressClassName: nginx
  rules:
  - host: ruoyi.huanghuanhui.cloud
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ruoyi-vue-svc
            port:
              number: 80
  tls:
  - hosts:
    - ruoyi.huanghuanhui.cloud
    secretName: ruoyi-ingress-tls
EOF
```

```
kubectl create secret -n ruoyi \
tls ruoyi-ingress-tls \
--key=/root/ssl/huanghuanhui.cloud.key \
--cert=/root/ssl/huanghuanhui.cloud.crt
```

```
kubectl apply -f ~/RuoYi-Cloud-rollout-yml/ruoyi-vue-Ingress.yml
```

> 访问地址：ruoyi.huanghuanhui.cloud

七、istio

前端：ruoyi-vue

1、ruoyi-vue

```
cat > ~/RuoYi-Cloud-rollout-yml/ruoyi-vue-rollout-istio.yml << 'EOF'
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: ruoyi-vue
  namespace: ruoyi
spec:
  replicas: 3
  strategy:
    canary:
      canaryService: ruoyi-vue-svc-canary # 关联 canary Service
      stableService: ruoyi-vue-svc-stable # 关联 stable Service
      trafficRouting:
        istio:
          virtualServices:
          - name: ruoyi-vue-vsvc # 关联的 Istio virtualService
            routes:
            - primary
      steps:
      - setWeight: 20
      - pause: {} # 人工卡点
      - setWeight: 40
      - pause: {duration: 10}
      - setWeight: 60
      - pause: {duration: 10}
      - setWeight: 80
      - pause: {duration: 10}
      - setWeight: 100
      - pause: {}
  revisionHistoryLimit: 5
  selector:
    matchLabels:
      app: ruoyi-vue
  template:
    metadata:
      labels:
        app: ruoyi-vue
        istio-injection: enabled
    spec:
      containers:
      - name: ruoyi-vue
        image: harbor.huanghuanhui.cloud/ruoyi-vue/ruoyi-vue:3
        ports:
        - name: http
          containerPort: 80
          protocol: TCP
EOF
```

```
kubectl delete -f ruoyi-vue-rollout.yml

kubectl apply -f ruoyi-vue-rollout-istio.yml
```

```
cat > ruoyi-vue-rollout-istio-svc.yml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: ruoyi-vue-svc-canary
  namespace: ruoyi
  labels:
    app: ruoyi-vue
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: ruoyi-vue
    # This selector will be updated with the pod-template-hash of the canary
ReplicaSet. e.g.:
    # rollouts-pod-template-hash: 7bf84f9696

---
apiVersion: v1
kind: Service
metadata:
  name: ruoyi-vue-svc-stable
  namespace: ruoyi
  labels:
    app: ruoyi-vue
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: ruoyi-vue
    # This selector will be updated with the pod-template-hash of the stable
ReplicaSet. e.g.:
    # rollouts-pod-template-hash: 789746c88d
EOF
```

```
kubectl delete -f ruoyi-vue-svc.yml

kubectl apply -f ruoyi-vue-rollout-istio-svc.yml
```

```
# 实现加请求头实现版本控制
cat > ruoyi-vue-vsvc.yml << 'EOF'
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ruoyi-vue-vsvc
  namespace: ruoyi
spec:
  gateways:
  - ruoyi-vue-gateway
  hosts:
  - "*"
  http:
  - name: primary
    match:
    - headers:
        x-canary:
          exact: test-user
      uri:
        prefix: /
    route:
    - destination:
        host: ruoyi-vue-svc-stable
      weight: 0
    - destination:
        host: ruoyi-vue-svc-canary
      weight: 100
  - route:
    - destination:
        host: ruoyi-vue-svc-stable
      weight: 100
EOF
```

```
kubectl apply -f ruoyi-vue-vsvc.yml
```

```
cat > ruoyi-vue-gateway.yml << 'EOF'
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: ruoyi-vue-gateway
  namespace: ruoyi
spec:
  selector:
    istio: ingressgateway # 默认创建的 istio ingressgateway pod 有这个 Label
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "ruoyi.huanghuanhui.cloud" # 匹配 host
EOF
```

```
kubectl apply -f ruoyi-vue-gateway.yml
```

```
kubectl argo rollouts get rollout ruoyi-vue

kubectl describe vs ruoyi-vue-vsvc
```