GitLab

部署

1、配置基础环境，安装docker

```
yum install -y yum-utils device-mapper-persistent-data lvm2
yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
sed -i 's+download.docker.com+mirrors.aliyun.com/docker-ce+' /etc/yum.repos.d/docker-ce.repo
yum makecache fast

selinuxdefcon 0
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
if egrep "7.[0-9]" /etc/redhat-release &>/dev/null; then
    systemctl stop firewalld
    systemctl disable firewalld
fi
yum install -y iptables-services vim lrzsz zip wget net-tools
systemctl enable iptables --now

yum -y install docker-ce
systemctl start docker &&systemctl enable docker

mkdir -p /etc/docker
tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://zd29wsn0.mirror.aliyuncs.com"]
}
EOF
systemctl daemon-reload
systemctl restart

if ! grep HISTTIMEFORMAT /etc/bashrc; then
    echo 'export HISTTIMEFORMAT="%F %T `whoami` "' >> /etc/bashrc
fi
if ! grep "* soft nofile 65535" /etc/security/limits.conf &>/dev/null; then
    cat >> /etc/security/limits.conf << EOF
    * soft nofile 65535
    * hard nofile 65535
EOF
fi
cat >> /etc/sysctl.conf << EOF
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_tw_buckets = 20480
net.ipv4.tcp_max_syn_backlog = 20480
net.core.netdev_max_backlog = 262144
net.ipv4.tcp_fin_timeout = 20
EOF
echo "0" > /proc/sys/vm/swappiness
sed -i '/SELINUX/{s/permissive/disabled/}' /etc/selinux/config
setenforce 0
```

2、安装docker-compose

```
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
docker-compose --version
```

3、编写gitlab的compose.yaml文件

```
cat >gitlab-compose.yaml <<EOF
version: '3.8'
services:
  gitlab:
    image: gitlab/gitlab-ce:latest
    container_name: gitlab
    ports:
      - "80:80"    # HTTP
      - "443:443"  # HTTPS
      - "2222:22"    # SSH  修改端口号，避免端口重复
    volumes:
      - gitlab_config:/etc/gitlab        # GitLab 配置数据
      - gitlab_logs:/var/log/gitlab      # GitLab 日志
      - gitlab_data:/var/opt/gitlab      # GitLab 数据（包括仓库等）
    restart: unless-stopped
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'http://192.168.72.128'  # 请将此 URL 替换为您自己的域名或 IP 地址
        # 其他 GitLab 配置项可以在这里添加，使用 GitLab Omnibus 配置格式

volumes:
  gitlab_config:
  gitlab_logs:
  gitlab_data:
EOF
```

3.1语法解释

```
image: gitlab/gitlab-ce:latest      # 选择镜像为 gitlab-ce 社区版本
container_name: gitlab              # 定义容器名称

# 声明 volumes 的好处 将数据卷的声明与服务的配置分开，使得文件结构更加清晰，方便阅读和维护。拥有可重用性

# 如果要查看有哪些 volume，可以适用这个命令
docker volume ls
DRIVER    VOLUME NAME
local     composes_gitlab_config
local     composes_gitlab_data
local     composes_gitlab_logs

# 查看 volume 详细信息
docker volume inspect composes_gitlab_config
```

```
[
    {
        "CreatedAt": "2024-04-17T22:30:55-04:00",
        "Driver": "local",
        "Labels": {
            "com.docker.compose.project": "composes",
            "com.docker.compose.version": "1.29.2",
            "com.docker.compose.volume": "gitlab_config"
        },
        "Mountpoint": "/var/lib/docker/volumes/composes_gitlab_config/_data",
        "Name": "composes_gitlab_config",
        "Options": null,
        "Scope": "local"
    }
]

```

4、启动和关闭 gitlab 操作

```
# 语法格式
docker-compose -f compose.yaml 【文件路径，比如 /opt/composes/gitlab-compose.yaml】 up -d【动作】
# 用于首次启动服务。如果服务的容器尚未创建，up 命令会创建并启动容器
docker-compose -f gitlab-compose.yaml up -d

# 用于启动已经存在但是被停止的容器。它不会重新创建容器，也不会应用配置的更新
docker-compose -f gitlab-compose.yaml start

# 用于停止容器
docker-compose -f gitlab-compose.yaml stop

# 查看 docker-compose 中管理容器的状态
docker-compose -f gitlab-compose.yaml ps
       Name                     Command               State
                     Ports
------------------------------------------------------------------------------
composes_jenkins_1   /usr/bin/tini -- /usr/loca ...   Up
0.0.0.0:50000->50000/tcp,:::50000->50000/tcp

0.0.0.0:8080->8080/tcp,:::8080->8080/tcp
```

5、访问GitLab web

```
#访问地址
http://ip:80

#首次访问jenkins web界面时，需要输入密码。获取密码的指令如下：
docker exec -it gitlab grep 'Password:' /etc/gitlab/initial_root_password
vO+Uc1RcOwB1MD1E5iFr7y+TQBt/ioKT85oVkZGjmxo=
```

注意：

在新建完成项目后，使用ssh密钥将公钥上传到 git 上所关联的账号，使用如下命令创建钥匙对

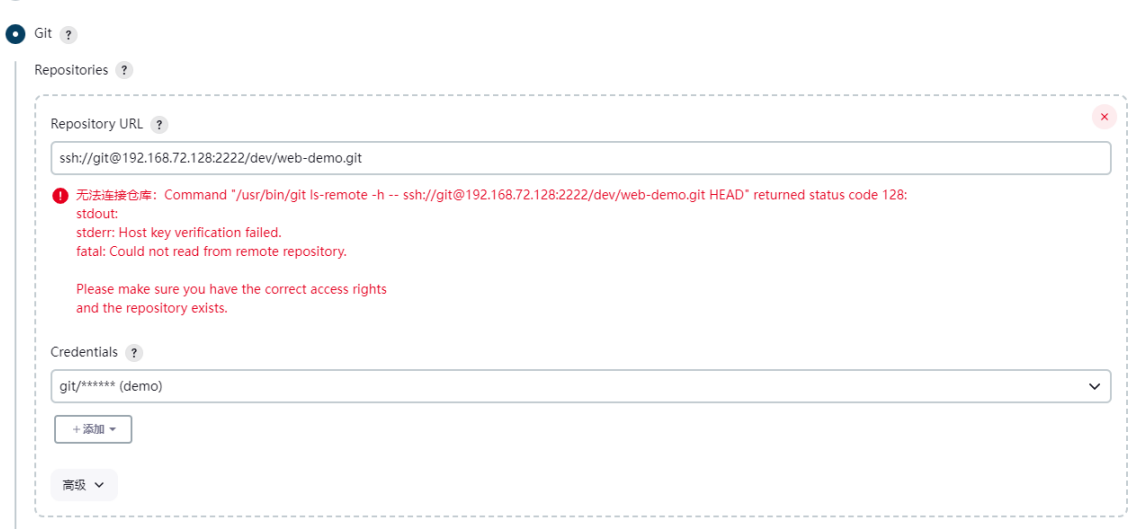
```
[root@hecs-131633 ~]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:ZhfoUld5JDAkO1TI27RSydSFuScOsDs+5Q3+6YPZNbw root@hecs-131633
The key's randomart image is:
+---[RSA 2048]----+
|        o==*==.+.|
|         +=o*.=  |
|        o oB o . |
|       o .+.+ o .|
|      . S .o o.o |
|       + .o o .o.|
|         . = * .o|
|          o = +E |
|           . o+. |
+----[SHA256]-----+

[root@hecs-131633 ~]# ls /root/.ssh/id_rsa*
/root/.ssh/id_rsa  /root/.ssh/id_rsa.pub
# /root/.ssh/id_rsa  私钥，不要随意泄露
# /root/.ssh/id_rsa.pub 公钥，上传到 git
```

在下载克隆项目的时候，会出现异常。因为我们的端口发生了改变。使用下列方法进行克隆

```
git clone ssh://git@192.168.72.128:2222/dev/web-demo.git
```

在jenkins上配置和git代码关联时，链接不上。可能造成的原因



```
Repository URL
?
ssh://git@192.168.72.128:2222/dev/web-demo.git
无法连接仓库：Command "/usr/bin/git ls-remote -h --
ssh://git@192.168.72.128:2222/dev/web-demo.git HEAD" returned status code
128:
stdout:
stderr: Host key verification failed.
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

解决方法

出现的错误提示 "Host key verification failed" 表明 Jenkins 服务器在尝试通过 SSH 连接到 Git 服务器时因为无法验证主机密钥而失败了。这通常发生在首次尝试连接到一个 SSH 服务器时，因为该服务器的密钥还没有被添加到 Jenkins 服务器的 known_hosts 文件中。

要解决这个问题，您需要在 Jenkins 服务器上手动接受 Git 服务器的 SSH 密钥，或者禁用主机密钥验证（后者不推荐，因为这会降低安全性）。以下是解决步骤：

手动接受 SSH 密钥

登录到运行 Jenkins 的服务器。

作为 Jenkins 运行的用户，手动 SSH 连接到 Git 服务器。例如：

```
ssh -p 2222 git@192.168.72.128
```