在NX-OS Bash Shell中安装Docker合成

目录

简介

本文档介绍在NX-OS Bash外壳中安装Docker Compose软件包所用的步骤。

从NX-OS版本9.2(1)开始，Cisco Nexus 3000和9000系列设备支持Bash外壳中的Docker功能。 如Docker Compose文档所述，“Compose是定义和运行多容器Docker应用程序的工具。” Docker Compose允许应用程序开发人员定义在名为“docker-compose.yml”的单个YAML文件中构成应用程序的所有服务。 然后，只需一条命令，即可创建、构建和启动所有这些服务。此外，所有服务都可以从Docker Compose命令套件内停止和监控。

虽然NX-OS Bash外壳本地支持Docker功能，但必须单独安装Docker Compose。

先决条件

要求

本文档要求在Cisco Nexus设备上启用Bash外壳。有关启用Bash外壳的说明，请参阅《Cisco Nexus 9000系列NX-OS可编程性指南》中Bash章节的“访问Bash”部分。

本文档要求将Bash外壳配置为能够将域名解析为IP地址的DNS客户端。有关在Bash外壳中配置DNS服务器的说明，请参阅文档。

使用的组件

本文档中的信息基于以下软件和硬件版本：

- 从NX-OS版本9.2(1)开始的Nexus 9000平台
- 从NX-OS版本9.2(1)开始的Nexus 3000平台

本文档中的信息在特定实验室环境设备上创建。本文档中使用的所有设备最初均采用原始（默认）配置。如果您使用的是真实网络，请确保您已经了解所有命令的潜在影响。

配置HTTP/HTTPS代理

如果您的环境需要使用HTTP或HTTPS代理，则需要先配置Bash外壳以使用这些代理，然后才能安装Docker Compose。

通过运行bash sudo su — 命令，以根用户Bash外壳。

```
Nexus# run bash sudo su -
root@Nexus#whoami
root
```

临时配置HTTP/HTTPS代理

要临时配置此会话的HTTP/HTTPS代理，请使用export命令定义“http_proxy”和“https_proxy”环境变量。下面显示了一个示例，其中“proxy.example-domain.com”是假设代理服务器的主机名。

```
root@Nexus#export http_proxy=http://proxy.example-domain.com:80/
root@Nexus#export https_proxy=https://proxy.example-domain.com:80/
```

使用echo $http_proxy和echo $https_proxy命令确量已，如下所示：

```
root@Nexus#echo $http_proxy
http://proxy.example-domain.com:80/ root@Nexus#echo $https_proxy
https://proxy.example-domain.com:80/
```

当会话终止时，将清除分配给这些环境变量的值，并且每次输入Bash外壳时都需要重新配置这些值。在以下示例中，退出上述配置的Bash会话，将提示返回到NX-OS。然后，创建到Bash外壳的新会话，其中已清除环境变量。

```
root@Nexus#export http_proxy=http://proxy.example-domain.com:80/
root@Nexus#export https_proxy=https://proxy.example-domain.com:80/
root@Nexus#echo $http_proxy
http://proxy.example-domain.com:80/ root@Nexus#echo $https_proxy
https://proxy.example-domain.com:80/ root@Nexus#exit
Nexus# run bash sudo su -
root@Nexus#echo $http_proxy

root@Nexus#echo $https_proxy

root@Nexus#
```

永久配置HTTP/HTTPS代理

要为进入Bash外壳的特定用户永久配置所有会话的HTTP/HTTPS代理，必须在每次任何用户登录时自动导出“http_proxy”和“https_proxy”环境变量。这可以通过将export命令附加到用户目录中的.bash_profile文件来实现，当该用户登录到Bash外壳时，Bash会自动加载该文件。下面显示了一个示例，其中“proxy.example-domain.com”是假设代理服务器的主机名。

```
root@Nexus#pwd
/root
root@Nexus#ls -al
total 28
drwxr-xr-x 5 root floppy 200 Dec 6 13:22 . drwxrwxr-t 62 root network-admin 1540 Nov 26 18:10 ..
```

```
-rw------- 1 root root 9486 Dec 6 13:22 .bash_history -rw-r--r-- 1 root floppy 703 Dec 6 13:22
.bash_profile drwx------ 3 root root 60 Nov 26 18:10 .config drwxr-xr-x 2 root root 60 Nov 26
18:11 .ncftp -rw------- 1 root root 0 Dec 5 14:37 .python-history -rw------- 1 root floppy 12
Nov 5 05:38 .rhosts drwxr-xr-x 2 root floppy 60 Nov 5 06:17 .ssh -rw------- 1 root root 5499 Dec
6 13:20 .viminfo root@Nexus#echo "export http_proxy=http://proxy.example-domain.com:80/" >>
.bash_profile
root@Nexus#echo "export https_proxy=https://proxy.example-domain.com:80/" >> .bash_profile
root@Nexus#cat .bash_profile | grep proxy
export http_proxy=http://proxy.example-domain.com:80/
export https_proxy=https://proxy.example-domain.com:80/
root@Nexus#exit
Nexus# run bash sudo su - root@Nexus#echo $http_proxy
http://proxy.example-domain.com:80/
root@Nexus#echo $https_proxy
https://proxy.example-domain.com:80/
```

如果要为进入Bash外壳的所有用户的所有会话配置特定HTTP/HTTPS代理，请将这些`export`命令附加到/etc/profile文件。当任何用户登录Bash外壳时，Bash会先自动加载此文件 — 因此，登录Bash外壳的所有用户都将相应地配置其HTTP/HTTPS代理。

下面显示了一个示例，其中“proxy.example-domain.com”是假设代理服务器的主机名。然后，用户帐户“docker-admin”配置了Bash外壳类型，该类型允许用户帐户在远程访问设备时直接登录Bash外壳。然后，使用SSH通过使用docker-admin用户帐户的管理VRF访问Nexus设备的mgmt0 IP地址(192.0.2.1)。本示例显示已设置“http_proxy”和“https_proxy”环境变量，即使已将全新用户帐户登录到Bash外壳中。

```
root@Nexus#echo "export http_proxy=http://proxy.example-domain.com:80/" >> /etc/profile
root@Nexus#echo "export https_proxy=https://proxy.example-domain.com:80/" >> /etc/profile
root@Nexus#cat /etc/profile | grep proxy
export http_proxy=http://proxy.example-domain.com:80/ export https_proxy=https://proxy.example-
domain.com:80/ root@Nexus#exit
Nexus# run bash sudo su -
root@Nexus#echo $http_proxy
http://proxy.example-domain.com:80/ root@Nexus#echo $https_proxy
https://proxy.example-domain.com:80/ root@Nexus#exit
Nexus# configure terminal
Nexus(config)# username docker-admin role dev-ops password example_password
Nexus(config)# username docker-admin shelltype bash
Nexus(config)# exit
Nexus# ssh docker-admin@192.0.2.1 vrf management
Password: -bash-4.3$ whoami
docker-admin
-bash-4.3$ echo $http_proxy
http://proxy.example-domain.com:80/ -bash-4.3$ echo $https_proxy
https://proxy.example-domain.com:80/
```

安装Docker撰写

要安装Docker Compose，必须使用`wget`实用程序下载Docker Compose的最新二进制版本，然后将该二进制文件放在/usr/bin目录中。

1.在Docker Compose GitHub页面上确定最新可用版本中可用的Docker Compose最新稳定版本。在网页顶部找到最新稳定版本的版本号。在撰写本文时，最新稳定版本为1.23.2。

2.使用上一步中找到的最新稳定版本的版本号替换下`URL`{latest-version}，为Docker合成二进制文件创建URL:

https://github.com/docker/compose/releases/download/{latest-version}/docker-compose-Linux-x86_64

例如，在撰写本文时，1.23.2的URL如下：https://github.com/docker/compose/releases/download/1.23.2/docker-compose-Linux-x86_64

3.使用run bash sudo su — 命令从NX-OS提示符中以root身份输Bash shell，如下所示：

```
Nexus# run bash sudo su -
root@Nexus#whoami
root
```

4.如有必要，请将Bash外壳的网络命名空间上下文更改为具有DNS和Internet连接的命名空间。网络命名空间在逻辑上与NX-OS VRF相同。以下示例演示如何切换到在此特定环境中具有DNS和Internet连接的管理网络命名空间上下文。

```
root@Nexus#ip netns exec management bash
root@Nexus#ping cisco.com -c 5
PING cisco.com (72.163.4.161) 56(84) bytes of data. 64 bytes from www1.cisco.com (72.163.4.161):
icmp_seq=1 ttl=239 time=29.2 ms 64 bytes from www1.cisco.com (72.163.4.161): icmp_seq=2 ttl=239
time=29.3 ms 64 bytes from www1.cisco.com (72.163.4.161): icmp_seq=3 ttl=239 time=29.3 ms 64
bytes from www1.cisco.com (72.163.4.161): icmp_seq=4 ttl=239 time=29.2 ms 64 bytes from
www1.cisco.com (72.163.4.161): icmp_seq=5 ttl=239 time=29.2 ms --- cisco.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms rtt min/avg/max/mdev =
29.272/29.299/29.347/0.218 ms
```

5.输入以下命令，用上一的URL替换{docker-url}:wget {docker-url} -O /usr/bin/docker-compose。以下显示了执行此命令的示例，将https://github.com/docker/compose/releases/download/1.23.2/docker-compose-Linux-x86_64用作{docker-url}URL:

```
root@Nexus#wget https://github.com/docker/compose/releases/download/1.23.2/docker-compose-Linux-
x86_64 -O /usr/bin/docker-compose
--2018-12-06 15:24:36-- https://github.com/docker/compose/releases/download/1.23.2/docker-
compose-Linux-x86_64 Resolving proxy.example-domain.com... 2001:DB8::1, 192.0.2.100 Connecting
to proxy.example-domain.com|2001:DB8::1|:80... failed: Cannot assign requested address.
Connecting to proxy.example-domain.com|192.0.2.100|:80... connected. Proxy request sent,
awaiting response... 302 Found Location: https://github-production-release-asset-
2e65be.s3.amazonaws.com/15045751/67742200-f31f-11e8-947e-bd56efcd8886?X-Amz-Algorithm=AWS4-HMAC-
SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20181206%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Date=20181206T152526Z&X-Amz-Expires=300&X-Amz-
Signature=dfccfd5a32a908040fd8c18694d6d912616f644e7ab3564c6b4ce314a0adbbc7&X-Amz-
SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Ddocker-
compose-Linux-x86_64&response-content-type=application%2Foctet-stream [following] --2018-12-06
15:24:36-- https://github-production-release-asset-2e65be.s3.amazonaws.com/15045751/67742200-
f31f-11e8-947e-bd56efcd8886?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=AKIAIWNJYAX4CSVEH53A%2F20181206%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Date=20181206T152526Z&X-Amz-Expires=300&X-Amz-
Signature=dfccfd5a32a908040fd8c18694d6d912616f644e7ab3564c6b4ce314a0adbbc7&X-Amz-
SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Ddocker-
compose-Linux-x86_64&response-content-type=application%2Foctet-stream Connecting to
proxy.example-domain.com|192.0.2.100|:80... connected. Proxy request sent, awaiting response...
200 OK Length: 11748168 (11M) [application/octet-stream] Saving to: ‚Äò/usr/bin/docker-
compose‚Äô /usr/bin/docker-compose
100%[=============================================================================================
==========================================>] 11.20M 6.44MB/s in 1.7s 2018-12-06 15:24:38 (6.44
MB/s) - ‚Äò/usr/bin/docker-compose‚Äô saved [11748168/11748168] root@Nexus#
```

6.修改/usr/bin/docker-compose二进制文件的权限，使其可以使用`chmod +x /usr/bin/docker-compose`。下面演示了这一点：

```
root@Nexus#docker-compose
bash: /usr/bin/docker-compose: Permission denied
root@Nexus#chmod +x /usr/bin/docker-compose
root@Nexus#docker-compose
Define and run multi-container applications with Docker. Usage: docker-compose [-f --help--file
FILE Specify an alternate compose file--project-name NAME Specify an alternate project
namedirectory--verbose Show more output--log-level LEVEL Set log level (DEBUG, INFO, WARNING,
ERROR, CRITICAL)--no-ansi Do not print ANSI control characters--version Print version and exit--
host HOST Daemon socket to connect to--tls Use TLS; implied by --tlsverify--tlscacert CA_PATH
Trust certs signed only by this CA--tlscert CLIENT_CERT_PATH Path to TLS certificate file--
tlskey TLS_KEY_PATH Path to TLS key file--tlsverify Use TLS and verify the remote--skip-
hostname-check Don't check the daemon's hostname against theinthe--project-directory PATH
Specify an alternate working directorytheofthefile--compatibility If set, Compose will attempt
to convert
deploykeysinfilestoorafromthefileandthefilecreateandandtimefromacommandinarunningcontaineronacom
mandkillfromtheforaaonecommandnumberofforastartstoptheandstartversiontheversion
```

验证Docker撰写功能

您可以通过创建并运行小型docker-compose.yml文件来验证Docker Compose是否安装成功且运行正常。以下示例将逐步完成此过程。

```
root@Nexus#mkdir docker-compose-example
root@Nexus#cd docker-compose-example/
root@Nexus#ls -al
total 0
drwxr-xr-x 2 root root    40 Dec  6 15:31 .
drwxr-xr-x 6 root floppy 260 Dec  6 15:31 ..
root@Nexus#vi docker-compose.yml
root@Nexus#cat docker-compose.yml
version: "3"
services:
  example_mongo:
    image: mongo:latest
    container_name: "example_mongo"
  example_alpine:
    image: alpine:latest
    container_name: "example_alpine"
root@Nexus#docker-compose up
Creating network "docker-compose-example_default" with the default driver
Pulling example_mongo (mongo:latest)...
latest: Pulling from library/mongo
7b8b6451c85f: Pull complete
ab4d1096d9ba: Pull complete
e6797d1788ac: Pull complete
e25c5c290bde: Pull complete
45aa1a4d5e06: Pull complete
b7e29f184242: Pull complete
ad78e42605af: Pull complete
1f4ac0b92a65: Pull complete
55880275f9fb: Pull complete
bd0396c9dcef: Pull complete
28bf9db38c03: Pull complete
3e954d14ae9b: Pull complete
cd245aa9c426: Pull complete
Creating example_mongo  ... done
```

```
Creating example_alpine ... done
Attaching to example_alpine, example_mongo
example_mongo     | 2018-12-06T15:36:18.710+0000 I CONTROL [main] Automatically disabling TLS
1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none' example_mongo | 2018-12-
06T15:36:18.717+0000 I CONTROL [initandlisten] MongoDB starting : pid=1 port=27017
dbpath=/data/db 64-bit host=c4f095f9adb0 example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL
[initandlisten] db version v4.0.4 example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL
[initandlisten] git version: f288a3bdf201007f3693c58e140056adf8b04839 example_mongo | 2018-12-
06T15:36:18.717+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.2g 1 Mar 2016
example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL [initandlisten] allocator: tcmalloc
example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL [initandlisten] modules: none
example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL [initandlisten] build environment:
example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL [initandlisten] distmod: ubuntu1604
example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL [initandlisten] distarch: x86_64
example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL [initandlisten] target_arch: x86_64
example_mongo | 2018-12-06T15:36:18.717+0000 I CONTROL [initandlisten] options: { net: {
bindIpAll: true } } example_mongo | 2018-12-06T15:36:18.717+0000 I STORAGE [initandlisten]
example_mongo | 2018-12-06T15:36:18.717+0000 I STORAGE [initandlisten] ** WARNING: Using the XFS
filesystem is strongly recommended with the WiredTiger storage engine example_mongo | 2018-12-
06T15:36:18.717+0000 I STORAGE [initandlisten] ** See http://dochub.mongodb.org/core/prodnotes-
filesystem example_mongo | 2018-12-06T15:36:18.717+0000 I STORAGE [initandlisten]
wiredtiger_open config:
create,cache_size=31621M,session_max=20000,eviction=(threads_min=4,threads_max=4),config_base=fa
lse,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manage
r=(close_idle_time=100000),statistics_log=(wait=0),verbose=(recovery_progress), example_alpine
exited with code 0 example_mongo | 2018-12-06T15:36:19.722+0000 I STORAGE [initandlisten]
WiredTiger message [1544110579:722686][1:0x7f9d5de45a40], txn-recover: Set global recovery
timestamp: 0 example_mongo | 2018-12-06T15:36:19.745+0000 I RECOVERY [initandlisten] WiredTiger
recoveryTimestamp. Ts: Timestamp(0, 0) example_mongo | 2018-12-06T15:36:19.782+0000 I CONTROL
[initandlisten] example_mongo | 2018-12-06T15:36:19.782+0000 I CONTROL [initandlisten] **
WARNING: Access control is not enabled for the database. example_mongo | 2018-12-
06T15:36:19.782+0000 I CONTROL [initandlisten] ** Read and write access to data and
configuration is unrestricted. example_mongo | 2018-12-06T15:36:19.782+0000 I CONTROL
[initandlisten] example_mongo | 2018-12-06T15:36:19.783+0000 I STORAGE [initandlisten]
createCollection: admin.system.version with provided UUID: dc0b3249-576e-4546-9d97-de841f5c45c4
example_mongo | 2018-12-06T15:36:19.810+0000 I COMMAND [initandlisten] setting
featureCompatibilityVersion to 4.0 example_mongo | 2018-12-06T15:36:19.814+0000 I STORAGE
[initandlisten] createCollection: local.startup_log with generated UUID: 2f9820f5-11ad-480d-
a46c-c58222beb0ad example_mongo | 2018-12-06T15:36:19.841+0000 I FTDC [initandlisten]
Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'
example_mongo | 2018-12-06T15:36:19.842+0000 I NETWORK [initandlisten] waiting for connections
on port 27017 example_mongo | 2018-12-06T15:36:19.842+0000 I STORAGE
[LogicalSessionCacheRefresh] createCollection: config.system.sessions with generated UUID:
d4aeac07-29fd-4208-9f83-394b4af648a2 example_mongo | 2018-12-06T15:36:19.885+0000 I INDEX
[LogicalSessionCacheRefresh] build index on: config.system.sessions properties: { v: 2, key: {
lastUse: 1 }, name: "lsidTTLIndex", ns: "config.system.sessions", expireAfterSeconds: 1800 }
example_mongo | 2018-12-06T15:36:19.885+0000 I INDEX [LogicalSessionCacheRefresh] building index
using bulk method; build may temporarily use up to 500 megabytes of RAM example_mongo | 2018-12-
06T15:36:19.886+0000 I INDEX [LogicalSessionCacheRefresh] build index done. scanned 0 total
records. 0 secs ^C
Gracefully stopping... (press Ctrl+C again to force)
Stopping example_mongo  ... done
root@Nexus#
```

警告：确保执行`docker-compose`命令时，该命令是在具有DNS和Internet连接的网络命名空间的上下文中执行的。否则，Docker Compose将无法从Docker Hub提取请求的映像。

注意：要在连接到Docker Compose会话时停止由Docker Compose启动的多容器Docker应用，请按“Ctrl+C”键组合。

相关信息

- Docker撰写安装文档
- Docker撰写文档概述
- Cisco Nexus 9000系列NX-OS可编程性指南，版本9.x
- Cisco Nexus 9000系列NX-OS可编程性指南，版本7.x
- Cisco Nexus 9000系列NX-OS可编程性指南，版本6.x
- Cisco Nexus 3000系列NX-OS可编程性指南，版本9.x
- Cisco Nexus 3000系列NX-OS可编程性指南，版本7.x
- Cisco Nexus 3000系列NX-OS可编程性指南，版本6.x
- Cisco Nexus 3500系列NX-OS可编程性指南，版本9.x
- Cisco Nexus 3500系列NX-OS可编程性指南，版本7.x
- Cisco Nexus 3500系列NX-OS可编程性指南，版本6.x
- Cisco Nexus 3600系列NX-OS可编程性指南，版本9.x
- Cisco Nexus 3600系列NX-OS可编程性指南，版本7.x
- 借助思科开放式NX-OS实现可编程性和自动化