Future Internet
Brazilian Environment
for Experimentation

RNP

Rede Nacional de Pesquisa

Tutorial de isolamento de tráfego em camada 3 (transporte) como se fosse camada 2 (endereçamento) usando o protocolo VXLAN.

Autor: Andrei F. T. Rodrigues

UFRGS

Porto Alegre, 25 de Outubro de 2019

Visão Geral:

Este tutorial mostrará como o protocolo VXLAN pode ser utilizado tanto para extender uma LAN (área local de rede) para outro nó de computação quanto para isolar tráfegos entre recursos computacionais, podendo esses serem computadores, containers ou VM, sem a necessidade de configuração dos nós intermediários, caso comum em uma configuração de VLAN.

Será utilizado containers dentro de VMs (Virtual Machines, em português, máquinas virtuais) para simular recursos computacionais e servicos em nós de computação diversos com algumas conexões de rede diferentes.

Conhecimento básico sobre redes de computadores e Linux serão importante para entender o que está acontecendo e como aplicar essa ideia em outros ambientes.

Sumário:

Conceitos Importantes:

O que é uma VM ?

Uma Virtual Machine (VM) é uma máquina que roda um sistema operacional completo como se estivesse rodando em uma máquina física, mas que na verdade está em um ambiente especial em cima de outro sistema operacional, podendo por vezes ter as mesmas características ou não que a máquina física, dependendo do tipo de virtualização. Possibilita um VM ser dividido em diversas VMs, cada uma com seu sistema operacional, programas, bibliotecas e recursos computacionais limitados e isolados um do outro.

O que é um Container?

Container é um espaço de usuário (Namespace), virtualizado em camada de Sistema Operacional, que tem acesso apenas ao conteúdo designado a ele e normalmente com o objetivo de virtualizar o código de uma aplicação. Com isso se pode limitar os recursos computacionais do container e os diretórios acessíveis.

Comparação entre Container e VM:

A vantagem de se usar um container é que todo o código e as bibliotecas necessárias para se rodar uma aplicação ficam contidas e isoladas em um só lugar, evitando o conflito de bibliotecas para aplicações diferentes em um mesmo VM. Enquanto VMs normalmente ocupam GBs no disco, containers variam de algumas dezenas de MBs até poucas centenas, e com um tempo de inicialização normalmente menor do que o de VMs. No entanto, containers compartilham o mesmo sistema operacional e o mesmo Kernel, logo oferecem uma flexibilidade menor de arquitetura em comparação a VMs.

O que é LXD ?

LXD é um sistema de administração de contâiners. Ele tem uma API Rest em um socket que pode tanto ser apenas local quando acessado remotamente (se habilitado) através do qual um cliente em linha de comando se conecta para executar as ações.

LXD é uma nova interface melhorada para o programa bem LXC, que é o que

realmente cria os containers controlando os Sistemas de Contenção implementados no Kernel do Linux.

O que é VXLAN?

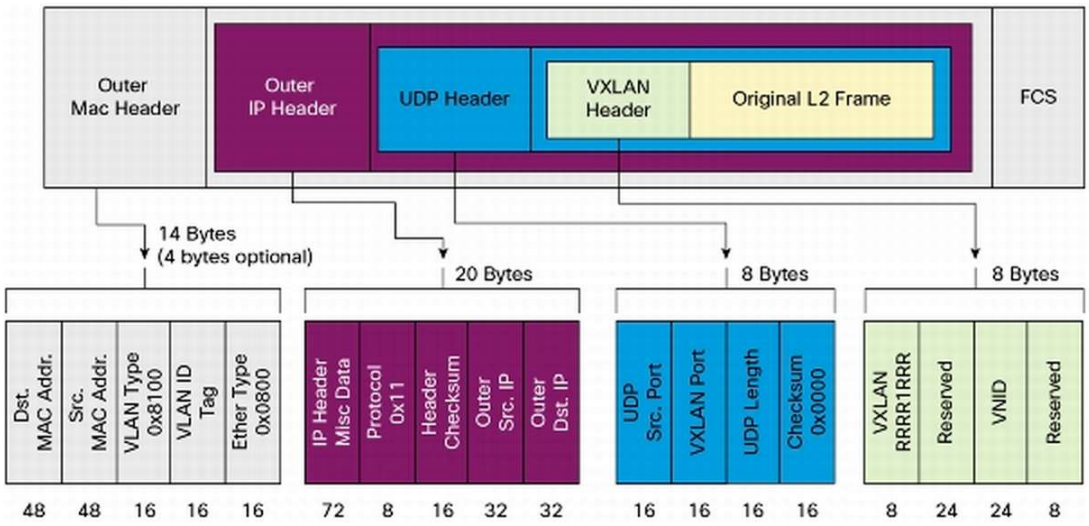
Virtual Extensible Local Area Network, é uma tecnologia de virtualização de rede, que cria uma camada de rede de camada 2 em cima de uma rede de camada 3, basicamente um túnel por dentro de outra rede. Visa resolver o problema de escalabilidade associado com grandes nuvens de computação. Oficialmente documentada pela IETF na RFC 7348.

VXLAN usa o protocolo UDP de envio, o que significa que não há garantia de que todos os pacotes chegarão ao seu destino e que não há notificação caso isso aconteceça, assim como as redes normais de computação. Caso haja necessidade, outro protocolo (como TCP) deve ser usado pela aplicação para garantir a transferência correta de todos os pacotes.

Usa um identificador de segmento de 24 bits, possibilitando criar até 16,7 milhões de redes VXLAN, um valor muito maior que as 4096 redes que as VLANs possibilitam com seus identificadores de 12 bits.

A VXLAN funciona encapsulando o pacote original como payload de um novo pacote UDP, como visto abaixo:

Figure 1. VXLAN Packet Format

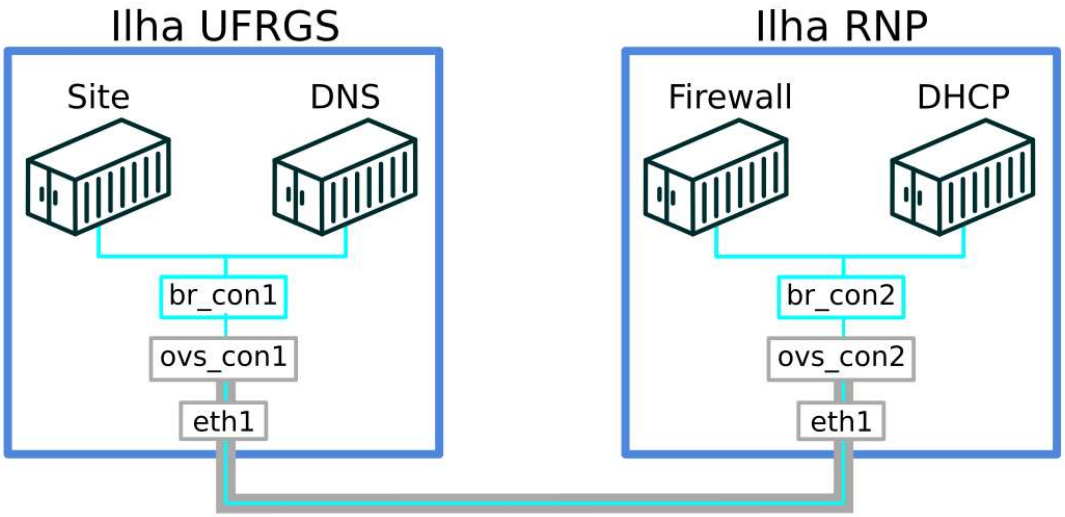


Uma nota importante é que o protocolo VXLAN acrescenta um header de 50 bytes ao pacote original, o que torna necessário um cuidado especial para diminuir o

MTU de todas as interfaces que estão dentro da VXLAN, para que não hava Jabber, um dos problemas mais comuns na configuração dessas redes e que consiste em os pacotes sejam divididos para caber no tamanho máximo de transferência (MTU) de um pacote por essas interfaces. Para redes Ethernet’s comuns, o limite do MTU é 1500 bytes e no nosso experimento será necessário diminuir o MTU das interfaces dentro dos containers para o valor 1440 para evitar problemas.

Descrição do Experimento:

1º Parte



A primeira fase do experimento consistirá na instalação do openvswitch, das bridges linux, da instanciação das bridges (switchs) virtuais e da configuração das conexões. Será notado que é possível realizar transferência de dados entre todos os recursos conectados na mesma rede.

2º Parte

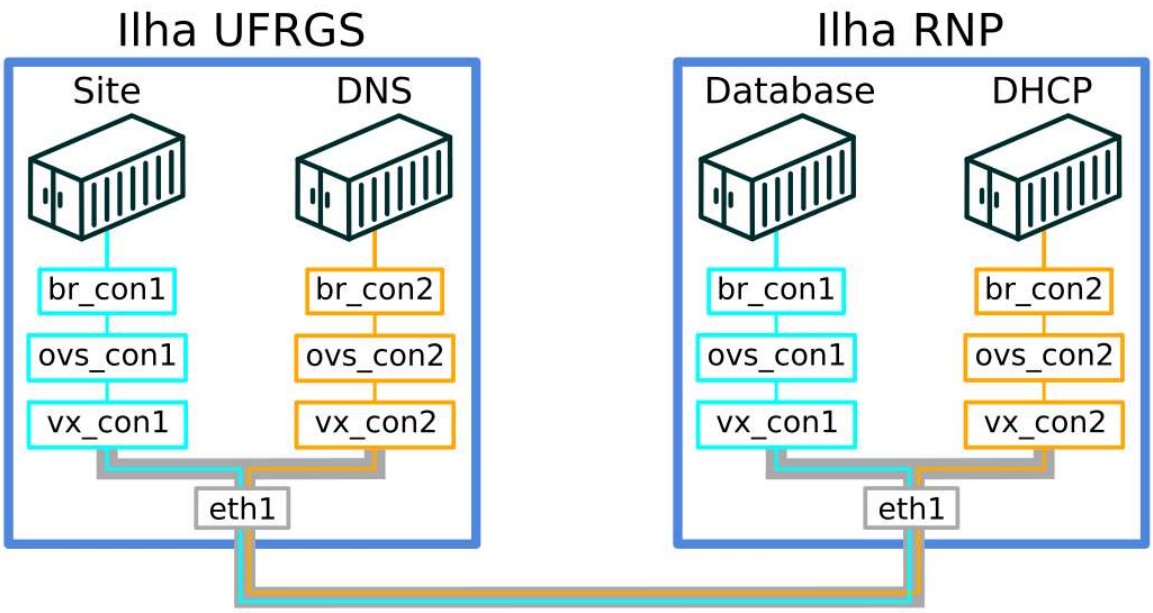


Figura ilustrando a infraestrutura virtual de rede que será criada. Os nomes dos containers representam possíveis serviços que poderiam ser designados.

A seguir iremos criar uma segunda infraestrutura de rede virtual, que permitirá isolar o tráfego como desejarmos, sem a necessidade de configuração manual da rede do Fibre. Neste experimento, recursos que estão no mesmo host não conseguirão se ver. Isto pode ilustrar a necessidade de alocar containers dentro de uma Cloud no host que estiver com mais recurso computacional disponível, ao contrário de ficar confinado ao host que contém uma determinada configuração ou VLAN.

Alocação:

Para o processo de alocação, o documento omf6-hello-world cobre os passos básicos a serem seguidos.

Os recursos utilizados são:

2 VMs Ubuntu, de no mínimo 256Mb de RAM, em ilhas diferentes.

1 VLAN.

Após todos os recursos serem alocados, criar uma reserva para utilização dos recursos.

Na área de experimentação, selecionar o script ‘Start Experiment’.

A partir da área inicial do slice, entrar em cada um dos hosts para verificar o IP na interface eth1.<vlan_id> e realizar o comando

```
$ ping -I eth1.<vlan_id> <ip_address>
```

com o ip address da outra VM para realizar um teste de conexão. Caso não haja conexão entre os nós, entrar em contato com o suporte da RNP para eles verificarem.

Configuração das VMs:

Todos os passos abaixo deverão ser executados nas duas máquinas que serão configuradas, com exceção de alguns passos que conterão detalhes extras.

Instalar open-vswitch:

```
$ sudo apt-get install openvswitch-switch
```

Verifique se foi instalado corretamente com o comando e o retorno a seguir:

```
$ sudo ovs-vsctl --version
ovs-vsctl (Open vSwitch) 2.9.2
DB Schema 7.15.1
```

Crie 1 linux bridge em cada VM:

```
$ sudo brctl addbr br_con1
```

Crie 1 bridge no openvswitch em cada VM:

```
$ sudo ovs-vsctl add-br ovs_con1
```

Conecte a bridge linux na bridge do openvswitch:

```
$ sudo brctl addif br_con1 ovs_con1
```

Criaremos agora a conexão VXLAN. Adicione uma nova porta na bridge criada de cada VM e configure o <ip remoto> com o IP da outra VM a ser direcionado o tráfego:

```
$ sudo ovs-vsctl add-port ovs_con1 vx_con1
$ sudo ovs-vsctl set interface vx_con1 type=vxlan
$ sudo ovs-vsctl set interface vx_con1 options:remote_ip=<ip_remoto>
```

Esta última série de comandos também pode ser resumida em um único comando:

```
$ sudo ovs-vsctl add-port ovs_con1 vx_con1    \
-- set interface vx_con1    \
type=vxlan    \
options:remote_ip=<ip_remoto>
```

Verificando que todos os comandos foram executados corretamente:

```
$ sudo ovs-vsctl show
f0e157ad-dc88-4724-95ff-e15d33b662bf
  Bridge "ovs_con1"
    Port "ovs_con1"
        Interface "ovs_con1"
            type: internal
    Port "vx_con1"
        Interface "vx_con1"
            type: vxlan
            options: {remote_ip="192.168.5.2"}
```

Ou o mesmo resultado mas com uma versão resumida:

```
$ sudo ovs-appctl dpif/show
```

Agora levantamos a rede.

```
$ sudo ifconfig ovs_con1 up
$ sudo ifconfig br_con1 up
```

Para tornar essas configurações persistentes, podemos usar as linhas abaixo inseridas no arquivo /etc/network/interfaces:

```
auto ovs_con1
auto br_con1
iface br_con1 inet static
  bridge_ports  ovs_con1
  netmask 255.255.255.0
```

E devemos estar com uma configuração parecida com a abaixo:

```
$ brctl show
bridge name     bridge id               STP enabled           interfaces
br_con1      8000.1e4101bf5145          no                    ovs_con1
lxdbr0       8000.82702e1121d6          no
```

Instanciação e Configuração dos containers:

Para este experimento, será necessário inicializar 2 containers por host.

O lxd vem com 3 repositórios online de imagens de contâiners:

1. ubuntu: (for stable Ubuntu images)
2. ubuntu-daily: (for daily Ubuntu images)
3. images: (for a bunch of other distros)

Para inicializar uma imagem contida em um repositório online, basta usar o comando abaixo que automaticamente fará download e inicializará o container:

```
$ lxc launch <nome do repositório>:<alias ou fingerprint da imagem>
```

Caso já tivéssemos a imagem como um arquivo em nossa VM, usaríamos o comando abaixo para importar a imagem para a nossa pool de images e depois inicializar um container a partir daquela imagem:

```
$ lxc image import <arquivo> --alias <alias ou apelido para a imagem>
$ lxc launch <alias> <nome que será dada ao container>
```

Usaremos a imagem do alpine, devido ao seu tamanho compacto para o download, utilizando o comando abaixo:

```
$ lxc launch images:alpine/3.10/i386 <container name>
```

Verificamos o staus, nome e outras informações básicas referente ao container com:

```
$ lxc list
```

Há diversas formas de se conectar a imagem:

Via console do lxc, vai precisar de usuário e senha do container:

```
$ lxc console <container name>
```

Use a sequência de comando ctrl+a q para sair do console.

Também é possível executar comandos dentro do container usando

```
$ lxc exec <container name> <comandos>
```

Entrar no container via terminal bash:

```
$ lxc exec <container name> -- /bin/bash
```

Ou se for uma imagem de alpine, que não tem bash, via shell:

```
$ lxc exec <container name> -- /bin/sh
```

Visualizar informações detalhadas do container:

```
$ lxc info <container name>
```

E para informações dos dispositivos adicionados a um container:

```
$ lxc config device show <container name>
```

Conecte os dois containers na mesma bridge linux usando o comando abaixo. Dê preferência não conecte diretamente nas bridges do openvswitch, pois algumas vezes o LXC tem problemas em remover uma interface do openvswitch, caso que não ocorre nas bridges linux.

```
$ lxc config device add < containerName >  < deviceName > nic    \
    nictype=bridged    \
    parent=< bridgeNoHost >   \
    name=< interfaceNameInsideContainer > ;
```

Verificando que foi criado corretamente e um exemplo de resultado:

```
$ lxc config device show <containerName>
veth1:
  name: eth1
  nictype: bridged
  parent: br_con1
  type: nic
```

Caso precise alterar algum campo de configuração de algum device do container, o comando abaixo pode ser utilizado:

```
$lxc config device set <containerName> <deviceName> <campo>=<valor>
```

Dentro de cada container, para configurar a rede e ter persistência dessa configuração, iremos editar o arquivo /etc/network/interfaces e para ter o conteúdo igual ao exemplo abaixo. Importante notar que cada container deve ter um endereço de IP único. Use um editor de texto de linha de comando para isso, como vi, nano ou vim:

```
$ touch /etc/network/interfaces
$ vi /etc/network/interfaces
$ cat /etc/network/interfaces
auto eth0
iface eth0 inet dhcp
hostname $(hostname)

auto eth1
iface eth1 inet static
  address 10.0.0.4
  netmask 255.255.255.0
  mtu 1440
```

E por fim, dentro do container, use o comando abaixo, que reiniciará o serviço de networking do container, fazendo com que as configurações no arquivo sejam realizadas:

```
$ /etc/init.d/networking restart
```

Caso surja o erro “ip: an inet prefix is expected rather than < ip address >”, verifique a quebra de linha do final de cada linha do arquivo acima.

Agora de dentro de cada container use o comando ping para testar a conexão entre os containers e a latência entre eles.

```
$ ping <ip address>
```

Perceba que todos os containers tem acesso a todos os outros containers nessa configuração.

Instale o programa tcpdump em um container e verifique que o tráfego está chegando no destino.

```
$ apk add tcpdump
```

E o faço monitorar a interface que está conectada a VXLAN, com o comando abaixo:

```
$ tcpdump -i eth1
```

Você também pode utilizar o comando iperf para verificar a largura da banda da conexão e para verificar se está ocorrendo perda de pacotes ou se o MTU está incorreto.

```
$ apk add iperf3
```

No **Apêndice 2 – Exemplo de Ferramentas para medir Desempenho de Redes** há uma lista das ferramentas aqui sugeridas para este experimento.

Criando redes com isolamento

Isolaremos o tráfego entre determinados containers criando criando outro par de switchs virtuais e bridges linux e um par de keys diferentes.Para esse isolamento, usaremos a opção de “key”, que será o ID da VXLAN, e apenas outras ports que contenham a mesma key receberão esse tráfego.

Para adicionar uma key um switch, você seleciona a interface que contêm a porta VXLAN, e usa o comando abaixo. Tenha certeza de colocar o nome da interface corretamente. Todo o switch agora receberá o tráfego que contenha aquela VXLAN_ID.

```
$ sudo ovs-vsctl set interface <interface name> options:key=<vxlan_id>
```

Adicione a mesma key as interfaces vxlan que já foram criadas.

Em cada máquina virtual agora crie outro switch no openvswitch, adicione e configure uma porta vxlan nele, podendo usar o template abaixo, mais compacto:

```
$ sudo ovs-vsclt add-br <bridge name>
$ sudo ovs-vsctl add-port <bridge name> <interface name>    \
-- set interface <interface name>    \
type=vxlan    \
options:remote_ip=<ip_remoto>    \
options:key=<vxlan_id> ;
```

E verifique que os switchs e interfaces foram criadas corretamente e que não há nenhuma mensagem de erro. Como no exemplo de execução abaixo:

```
$ sudo ovs-vsctl show
f0e157ad-dc88-4724-95ff-e15d33b662bf
    Bridge "ovs_con2"
        Port "ovs_con2"
            Interface "ovs_con2"
                type: internal
        Port "vx_con2"
            Interface "vx_con2"
                type: vxlan
                options: {key="20", remote_ip="192.168.1.2"}
    Bridge "ovs_con1"
        Port "ovs_con1"
            Interface "ovs_con1"
                type: internal
        Port "vx_con1"
            Interface "vx_con1"
                type: vxlan
                options: {key="35", remote_ip="192.168.1.2"}
```

Note como as portas vxlan (vx_con) apontam para o mesmo endereço IP, que é o da nossa outra máquina, e que cada uma tem uma key diferente. A key é importante para que as redes sejam isoladas uma da outra.

Crie a bridge linux e associe o switch virtual recém criado a ela:

```
$sudo brctl addbr <bridgeName2>
$sudo brctl addif <switchName2>
```

O resultado será algo parecido com o abaixo. Omiti as interfaces dos containers para facilitar a visualização:

```
$sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br_con1         8000.1e4101bf5145       no              ovs_con1
br_con2         8000.9699832abc42       no              ovs_con2
lxdbr0          8000.000000000000       no
```

Agora configure o dispositivo de rede de um dos containers para se conectar a bridge linux recém criada e reinicialize o container. Abaixo estão os templates dos comandos para visualizar os dispositivos criados em um container, alterar o valor de um campo em determinado dispositivo de um container e para reinicializar o container:

```
$ lxc config device show <containerName
$ lxc config device set <containerName> <deviceName> <campo>=<valor>
$ lxc restart <containerName>
```

Abaixo um exemplo de execução:

```
$ lxc config device show Container1
veth1:
  name: eth1
  nictype: bridged
  parent: br_con1
  type: nic
$ lxc config device set Container1 veth1 parent=br_con2
$ lxc restart Container1
```

Agora realize a mesma operação na outra outra VM e no outro container.

Após a inicialização dos containers, entre em cada container e utilize os comandos de teste de rede anteriores para verificar como as métricas continuam iguais e de como mesmo os containers tendo o mesmo IP, há agora duas redes diferentes isoladas, e que nenhum container consegue ver todos os containers.

No **Apêndice 2 – Exemplo de Ferramentas para medir Desempenho de Redes** há uma lista com algumas ferramentas que podem ser utilizadas para realizar teste de desempenho na rede.

Apêndice 1 – Comando para instalação no Alpine:

Caso deseje instalar algo no container Alpine, o gerenciado de pacotes do Alpine se chama apk.

Para instalar algo que já esteja no repositórios do Alpine, use:

```
$ apk update
$ apk upgrade
$ apk add <utilitário pra instalar>
```

Podemos instalar o shell bash com:

```
$ apk add bash
```

Instalar bash auto-completar:

```
$ apk add bash-completion
```

Tornar o bash o shell padrão alterando a linha correspondente ao root do arquivo */etc/passwd* *de:*

```
root:x:0:0:root:/root:/bin/ash
```

para:

```
root:x:0:0:root:/root:/bin/bash
```

Apêndice 2 – Exemplo de Ferramentas para medir Desempenho de Redes

Ping:

Envia pacotes do protocolo ICMP para testar conectividade entre nós de internet e consegue estimar uma latência entre esses nós. Um pacote do ping tem tamanho padrão 56 bytes, que combinados com os 8 bytes do header do protocolo ICMP resultam em 64 bytes, e por isso não percebem problemas devido ao tamanho do MTU.

Exemplo de uso:

```
$ ping -I <interface> <ip_address>
```

Iperf:

Performa uma teste de taxa de transferência entre dois nós de computação, tentando transmitir uma grande quantidade de pacotes na rede durante um período determinado, por padrão de 10 segundos. Pode usar protocolo UDP quanto TCP. É possível perceber problemas de MTU ao ter uma alta taxa de retransmissão (RTR).

Instalação no alpine:

```
$ apk add iperf3
```

Há ajuda disponível por linha de comando:

```
$ iper3 –help
```

Também utilizado verificar a largura da banda da conexão e pode-se usar para verificar se está ocorrendo perda de pacotes ou se o MTU está incorreto, caso aparente quando há fragmentação de pacotes e erro em serviços como ssh ou http.

Exemplo de uso comum do Iperf3:

Iniciar o lado do iperf que receberá os pacotes em um dos nós, também conhecido como o “server”:

```
$ iperf3 -s
```

Iniciar e realizar a medição enviando pacotes em direção aooutro nó em, isto é, para o “server”:

```
$ iperf3 -c <ip_address_do_server>
```

TCPDUMP:

Programa utilizado para ler todos os pacotes que estão passando pelo nó de computação ou por uma determinada interface.

```
$ apk add tcpdump
```

Exemplo de uso para monitorar o tráfego em uma interface:

```
$ tcpdump -i eth1
```