2021 SU VE489 Project

Group 16

Name: 陆昊融，娄辰飞

Student ID: 518370910194,

Demo Video Link (Task 2): https://jbox.sjtu.edu.cn/l/611Hsm

In this video I typed some wrong commands when executing the container for minecraft server, but the result is correct.

Task 2 Procedure

As we have already tested the communication between two containers on the same host in Task 1, in Task 2 we will not implement a server and a client on the same host again. We will directly implement cross-host communication, IPTV, gRPC-web, and Minecraft.

For simplicity, we will call the first host machine "host1" and the second host machine "host2".

host1:

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.122.128  netmask 255.255.255.0  broadcast 192.168.122.255
        inet6 fe80::590b:ad68:840d:f564  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:a7:a4:bf  txqueuelen 1000  (Ethernet)
        RX packets 3164132  bytes 2815227124 (2.8 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1665030  bytes 181994243 (181.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

flannel.1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.3.30.0  netmask 255.255.255.255  broadcast 10.3.30.0
        inet6 fe80::7831:c8ff:fe61:c6ec  prefixlen 64  scopeid 0x20<link>
        ether 7a:31:c8:61:c6:ec  txqueuelen 0  (Ethernet)
        RX packets 29016  bytes 252151046 (252.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12271  bytes 775975 (775.9 KB)
        TX errors 0  dropped 798 overruns 0  carrier 0  collisions 0
```

host2:

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.122.129  netmask 255.255.255.0  broadcast 192.168.122.255
        inet6 fe80::7dff:d134:1ee:dac9  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:d4:c2:3c  txqueuelen 1000  (Ethernet)
        RX packets 2167598  bytes 1341737134 (1.3 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1624920  bytes 410052003 (410.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

flannel.1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.3.50.0  netmask 255.255.255.255  broadcast 10.3.50.0
        inet6 fe80::f4f1:dfff:fe77:a2f5  prefixlen 64  scopeid 0x20<link>
        ether f6:f1:df:77:a2:f5  txqueuelen 0  (Ethernet)
        RX packets 12339  bytes 781311 (781.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26213  bytes 252005642 (252.0 MB)
        TX errors 0  dropped 1689 overruns 0  carrier 0  collisions 0
```

1. Cross-Host Etcd Configuration

The procedure is similar to Task1, only the `etcd.service` need to be changed.

```
# Copy from Task1
tar xzvf etcd-v3.3.10-linux-amd64.tar.gz
cd etcd-v3.3.10-linux-amd64
chmod +x {etcd,etcdctl}
cp {etcd,etcdctl} /usr/bin/

# Done both on host1 and host2
systemctl stop etcd
rm -rf /var/lib/etcd # Clear the previous cluster
vim /lib/systemd/system/etcd.service

################################################################################
# /lib/systemd/system/etcd.service
[Unit]
Description=etcd
After=network.target

[Service]
Environment=ETCD_NAME=etcd-1 # 'etcd-2' on host2
Environment=ETCD_DATA_DIR=/var/lib/etcd
Environment=ETCD_LISTEN_CLIENT_URLS=http://192.168.122.128:2379,http://127.0.0.1
:2379
Environment=ETCD_LISTEN_PEER_URLS=http://192.168.122.128:2380
Environment=ETCD_ADVERTISE_CLIENT_URLS=http://192.168.122.128:2379,http://127.0.
0.1:2379
Environment=ETCD_INITIAL_ADVERTISE_PEER_URLS=http://192.168.122.128:2380
Environment=ETCD_INITIAL_CLUSTER_STATE=new
Environment=ETCD_INITIAL_CLUSTER_TOKEN=etcd-cluster-token
Environment=ETCD_INITIAL_CLUSTER=etcd-1=http://192.168.122.128:2380,etcd-
2=http://192.168.122.129:2380
ExecStart=/usr/bin/etcd

[Install]
WantedBy=multi-user.target
################################################################################
```

```
systemctl daemon-reload
systemctl restart etcd
```

Then by running `etcdctl member list`, we have,

```
root@hunter:~# etcdctl member list
5ce6206715976822: name=etcd-1 peerURLs=http://192.168.122.128:2380 clientURLs=http://127.0.0.1:2379,http://192.168.122.128:2379 isLeader=true
c34f2eff3a560282: name=etcd-2 peerURLs=http://192.168.122.129:2380 clientURLs=http://127.0.0.1:2379,http://192.168.122.129:2379 isLeader=false
```

2. Cross-Host Flannel Configuration

The procedure is actually the same as Task1, except that we need to delete the previous flannel network,

```
# Copy from Task1
wget https://github.com/flannel-io/flannel/releases/download/v0.14.0/flanneld-
amd64
chmod +x flanneld-amd64
cp flanneld-amd64 /usr/bin/flanneld

# Done both on host1 and host2
systemctl stop flannel
ifconfig flannel.1 down
ip link delete flannel.1
vim /lib/systemd/system/flannel.service

################################################################################
# /lib/systemd/system/flannel.service
[Unit]
Description=flannel
After=etcd.service network.target

[Service]
ExecStart=/usr/bin/flanneld --etcd-endpoints=http://192.168.122.128:2379 -etcd-
prefix=/docker-flannel/network --iface=ens33 # 192.168.122.129 on host2

[Install]
WantedBy=multi-user.target
################################################################################

etcdctl set /docker-flannel/network/config {"Network":"10.3.0.0/16",
"SubnetLen": 24, "Backend": {"Type": "vxlan"}} # Set flannel network
etcdctl get /docker-flannel/network/config # Check flannel network config
systemctl daemon-reload
systemctl restart flannel

# Check the flannel running status
cat /run/flannel/subnet.env
# You should see something like this
################################################################################
FLANNEL_NETWORK=10.3.0.0/16
FLANNEL_SUBNET=10.3.30.1/24 # 10.3.50.1 on host2
FLANNEL_MTU=1450
FLANNEL_IPMASQ=false
################################################################################
```

Just like Task1, we need to add the configuration of flannel into the docker runtime,

```
 vim /lib/systemd/system/docker.service

# modify this line by adding the exec options
#############################################################################
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
--bip=10.3.30.1//24 --ip-masq=true --mtu=1450 # 10.3.50.1 on host2
#############################################################################

systemctl daemon-reload
systemctl restart docker
```

Then in the container, we can see that the container now has an IP address within the subnet.

```
root@monster:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.3.30.2  netmask 255.255.255.0  broadcast 10.3.30.255
        ether 02:42:0a:03:1e:02  txqueuelen 0  (Ethernet)
        RX packets 23  bytes 2920 (2.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

And the container on host1 now can communicate with the container on host2.

```
root@monster:/# ping 10.3.50.2
PING 10.3.50.2 (10.3.50.2) 56(84) bytes of data.
64 bytes from 10.3.50.2: icmp_seq=1 ttl=62 time=2.31 ms
64 bytes from 10.3.50.2: icmp_seq=2 ttl=62 time=2.08 ms
64 bytes from 10.3.50.2: icmp_seq=3 ttl=62 time=1.65 ms
64 bytes from 10.3.50.2: icmp_seq=4 ttl=62 time=1.45 ms
64 bytes from 10.3.50.2: icmp_seq=5 ttl=62 time=1.43 ms
^C
--- 10.3.50.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 1.430/1.783/2.305/0.350 ms
```

3. Cross-Host IPTV

Now that the basic communication is enabled, we are going to deploy some applications on our containers. We will first depoly IPTV, as its deployment is the most straightforward.

IPTV Server

First we create an IPTV-server container on host2,

```
docker run -it -d --name iptv_grpcweb ubuntu:latest /bin/bash
docker start iptv_grpcweb
docker attach iptv_grpcweb

# In the container, install dependencies
git clone https://github.com/free5gc/IPTV.git
apt udpdate
apt install -y ffmpeg nodejs curl
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add -
```

```
echo "deb https://dl.yarnpkg.com/debian/ stable main" | tee
/etc/apt/sources.list.d/yarn.list
apt update && apt install -y yarn

apt install software-properties-common # instead of python-software-properties
add-apt-repository ppa:gias-kay-lee/npm
apt-get update
apt-get install npm

wget https://dl.google.com/go/go1.12.9.linux-amd64.tar.gz
sudo tar -C /usr/local -zxvf go1.12.9.linux-amd64.tar.gz
mkdir -p ~/go/{bin,pkg,src}
echo 'export GOPATH=$HOME/go' >> ~/.bashrc
echo 'export GOROOT=/usr/local/go' >> ~/.bashrc
echo 'export PATH=$PATH:$GOPATH/bin:$GOROOT/bin' >> ~/.bashrc
echo 'export GO111MODULE=on' >> ~/.bashrc
echo 'export GOPROXY=https://goproxy.io' >> ~/.bashrc
source ~/.bashrc

go version # Check go version
# go version go1.12.9 linux/amd64

# Install go packages
go get -u github.com/gin-contrib/static
go get -u github.com/gin-gonic/gin
go get -u github.com/urfave/cli
go get -u gopkg.in/yaml.v2

# Build Web Client
cd IPTV/web-client
yarn install
yarn build
cd ..
vim iptvcfg.conf # Configure iptv details, change IP to the container's IP

vim iptv.go
################################################################################
    "github.com/free5gc/IPTV/factory"
    "github.com/free5gc/IPTV/iptv-server"
    "github.com/free5gc/IPTV/version"
################################################################################
vim iptv-server/iptv_server.go
################################################################################
    "github.com/free5gc/IPTV/factory"
    "github.com/free5gc/IPTV/iptv-server/hls-channel"
################################################################################

mkdir hls        # Create chache folder, default is ./hls
go run iptv.go
```

Now we can see the IPTV server is listening and serving HTTP on 10.3.50.3:8888

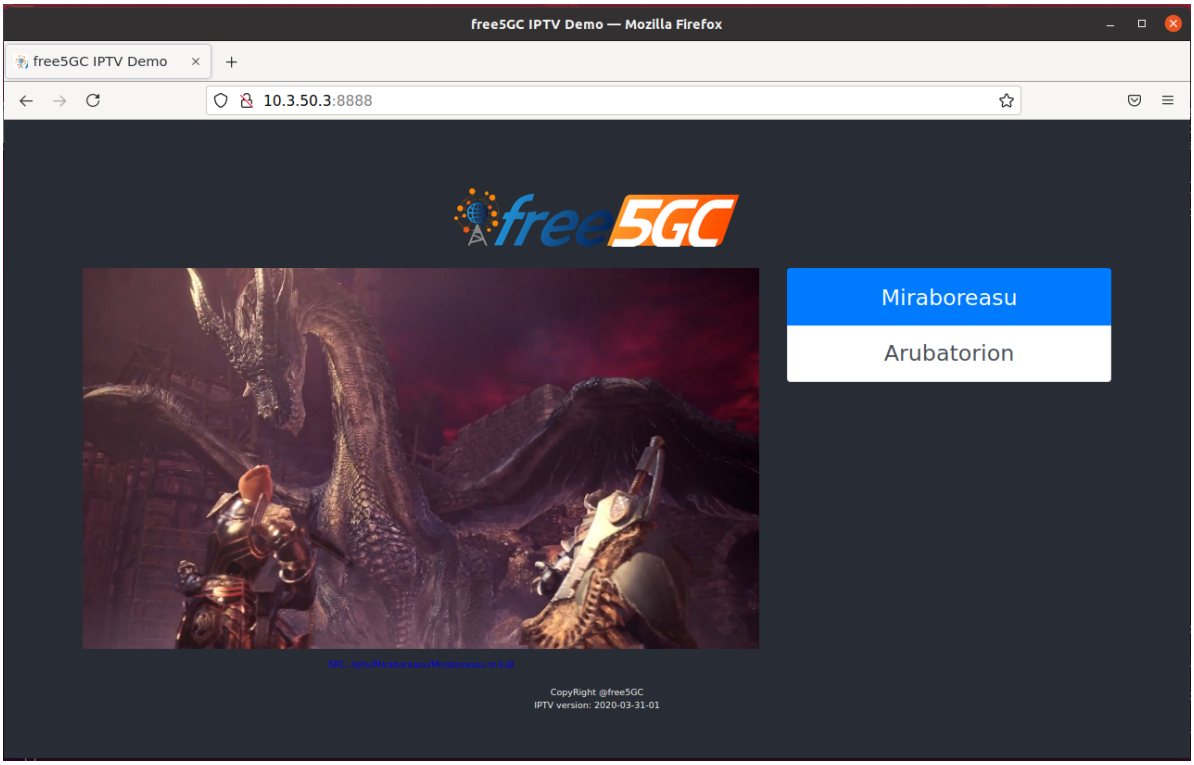
```
root@3bcc1527f4a3:~/IPTV# go run iptv.go
IPTV
Successfully initialize configuration ./iptvcfg.conf
[GIN-debug] [WARNING] Now Gin requires Go 1.12+.

[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /version                  --> github.com/free5gc/IPTV/iptv-server.Server.Start.func3 (6 handlers)
[GIN-debug] Listening and serving HTTP on 10.3.50.3:8888
```

IPTV Client

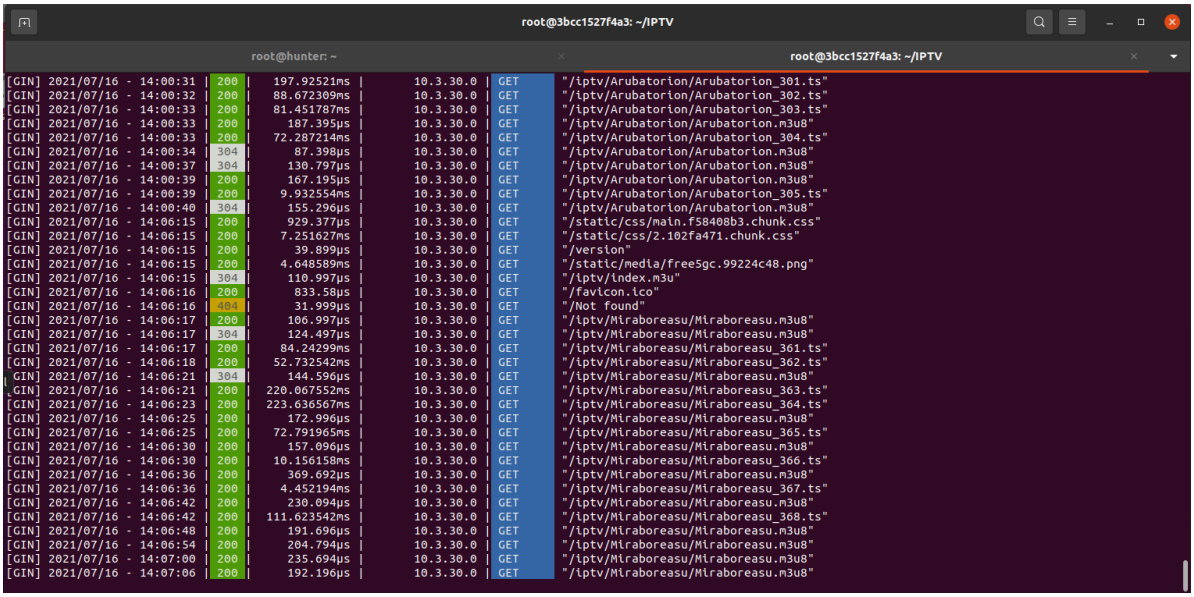
Then we create an IPTV client container on host2,

```
docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=$DISPLAY -h $HOSTNAME -v
$HOME/.Xauthority:/home/li/.Xauthority -itd --name=firefox ubuntu:latest
docker start firefox
docker attach firefox
xhost + # access control disabled, clients can connect from any host

# In the container
apt update
apt install -y firefox xorg openbox # install firefox and x11 components
firefox # Open firefox browser, it should be a GUI
```

In the firefox, enter `<IPTV Server IP>:8888`, we can see the GUI of IPTV,



And on the server side, we have the logs,



We can confirm that cross-host IPTV works well.

4. Cross-Host gRPC-Web

Then we will deploy gRPC-Web, as its setup is similar to IPTV. We deploy them in the same containers.

gRPC-Web Server

Using the same container `iptv_grpcweb` on host2,

```
# In the container for server
# Install dependencies
git clone https://github.com/grpc/grpc-web.git
wget
https://github.com/protocolbuffers/protobuf/releases/download/v3.17.3/protoc-
3.17.3-linux-x86_64.zip
unzip protoc-3.17.3-linux-x86_64.zip
mv bin/protoc /usr/bin
wget https://github.com/grpc/grpc-web/releases/download/1.2.1/protoc-gen-grpc-
web-1.2.1-linux-x86_64
mv protoc-gen-grpc-web-1.2.1-linux-x86_64 /usr/bin/protoc-gen-grpc-web
wget https://github.com/improbable-eng/grpc-
web/releases/download/v0.14.0/grpcwebproxy-v0.14.0-linux-x86_64.zip
mv dist/grpcwebproxy-v0.14.0-linux-x86_64 /usr/bin/grpcwebproxy
rm -rf dist bin include readme.txt # Clear the remaining file

# Generate Protobuf Messages and Client Service Stub
cd grpc-web/net/grpc/gateway/examples/helloworld/
protoc -I=. helloworld.proto \
  --js_out=import_style=commonjs:. \
  --grpc-web_out=import_style=commonjs,mode=grpcwebtext:.
# This will generate helloworld_pb.js and helloworld_grpc_web_pb.js under the
same directory

# Compile the Client JavaScript Code
npm install # This took long time on my computer
npx webpack client.js
```

```
# Run the Example
node server.js &
python3 -m http.server 8081 &
grpcwebproxy \
    --backend_addr=localhost:9090 \
    --run_tls_server=false \
    --allow_all_origins
```

Now the grpc-web is running,

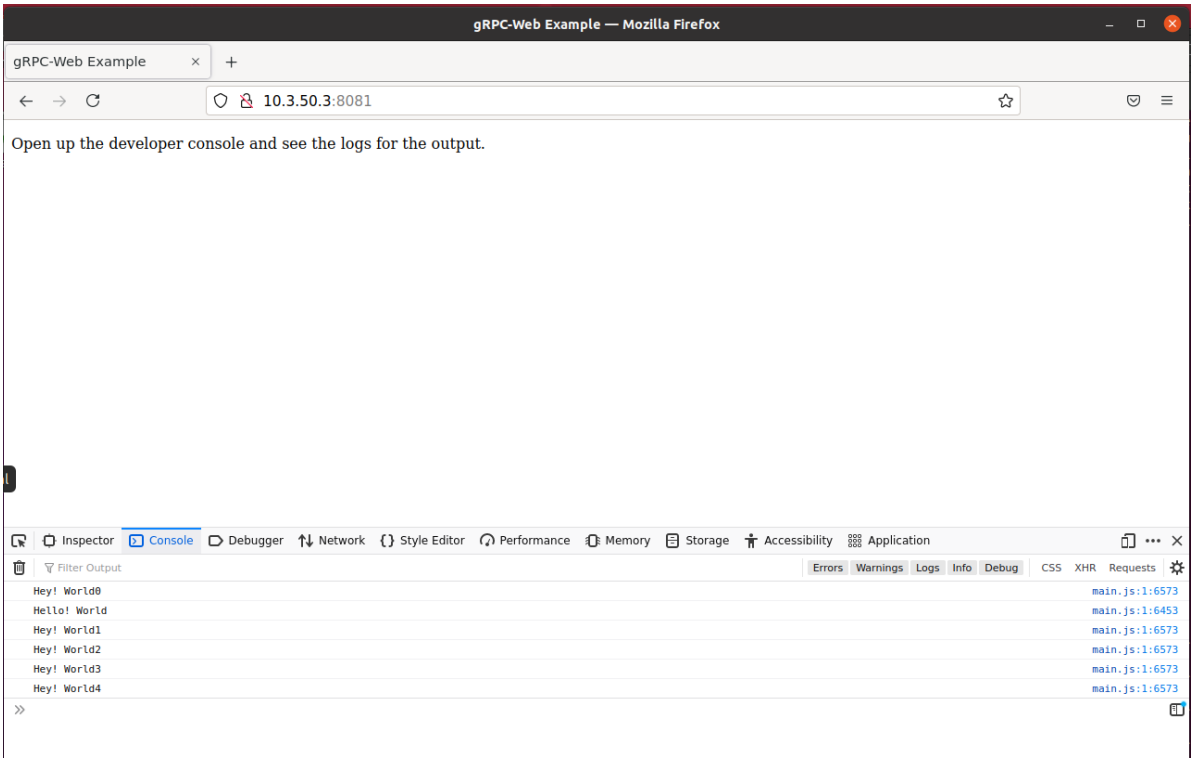
```
root@3bcc1527f4a3:~/grpc-web/net/grpc/gateway/examples/helloworld# node server.js &
[1] 135
root@3bcc1527f4a3:~/grpc-web/net/grpc/gateway/examples/helloworld# python3 -m http.server 8081 &
[2] 142
root@3bcc1527f4a3:~/grpc-web/net/grpc/gateway/examples/helloworld# Serving HTTP on 0.0.0.0 port 8081 (http://0.0.0.0:8081/) ...

root@3bcc1527f4a3:~/grpc-web/net/grpc/gateway/examples/helloworld# grpcwebproxy \
>     --backend_addr=localhost:9090 \
>     --run_tls_server=false \
>     --allow_all_origins
INFO[0000] [core] parsed scheme: ""                      system=system
INFO[0000] [core] scheme "" not registered, fallback to default scheme  system=system
INFO[0000] [core] ccResolverWrapper: sending update to cc: {[{localhost:9090  <nil> 0 <nil>}] <nil> <nil>}  system=system
INFO[0000] [core] ClientConn switching balancer to "pick_first"  system=system
INFO[0000] [core] Channel switches to new LB policy "pick_first"  system=system
INFO[0000] [core] Subchannel Connectivity change to CONNECTING  system=system
INFO[0000] [core] Channel Connectivity change to CONNECTING  system=system
INFO[0000] [core] Subchannel picks a new address "localhost:9090" to connect  system=system
INFO[0000] listening for http on: [::]:8080
INFO[0000] [core] Subchannel Connectivity change to READY  system=system
INFO[0000] [core] Channel Connectivity change to READY  system=system
```

gRPC-Web Client

Using the same container `firefox` on host1,

```
# In the container for client
firefox
```

In the firefox, enter `<IPTV Server IP>:8081`, and press F12 to open the console,



We receive the the message "Hello! World", which yields that our gRPC-web server and client work well.

5. Cross-Host Minecraft

Now we come to the most entertaining but also the most problem-prone part, Minecraft.

Minecraft Server

The image of Minecraft server we used is built by kitematic, and we directly pulled it from Docker Hub,

```
docker pull kitematic/minecraft
docker run --name=mc_server kitematic/minecraft
docker exec -it mc_server bash

# In the docker
ps aux | grep java | grep -v grep | grep -v sh
```

```
root@b85ba4a9bb48:/data# ps aux | grep java | grep -v grep | grep -v sh
root           7  5.1  9.2 3570828 371560 pts/0  Sl+  03:51   9:53 java -jar /minecraft_server.1.12.2.jar
```

We can see that the Minecraft is running, but we need to change some configuration,

```
sed -i '/online-mode/s/true/false/g' /data/server.properties # Set online-mode
to false
exit
# Exit the container
# On host2, restart mc_server
docker kill mc_server
docker start mc_server
```

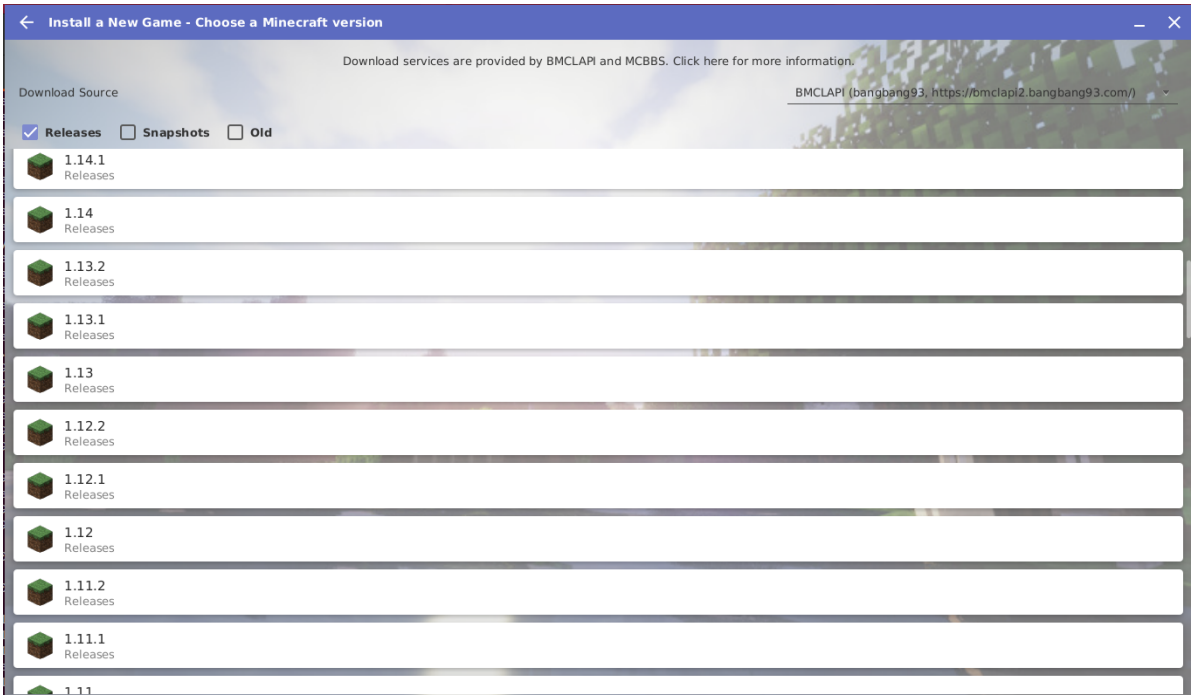
Now the Minecraft Server should be running.

Minecraft Client

We create a new container `mc_client` on host1 supporting GUI,

```
xhost +
docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=$DISPLAY -h $HOSTNAME -v
$HOME/.Xauthority:/home/li/.Xauthority -itd --name=mc_client ubuntu:latest
docker start mc_client
docker attach mc_client

# In the container, install dependencies
apt update && apt install -y wget xorg openbox # Install wget and X11 components
apt install -y openjdk-11-jdk openjdk-8-jdk # Install java11 and java8
wget http://ci.huangyuhui.net/job/HMCL/188/artifact/HMCL/build/libs/HMCL-
3.3.188.jar # Install HMCL

# Open HMCL launcher
java -jar HMCL-3.3.188.jar
```

Then in HMCL launcher, we install a new game of version 1.12.2.



Also, we should change the java directory in the global game settings to java8, as Minecraft of 1.12.2 version does not support java11.

After configuration, we can start playing Minecraft in the container!

We choose "Multiplayer" mode, and add a new server of our Minecraft Server's IP address,

We can see that the Minecraft Server is working well, then we can join this server.

Enjoy the Minecraft!

Discussion

- When changing the flannel config, although the config stored in the etcd cluster has been changed, the network interface still remains the same. Therefore, when we want to change the flannel config, we need to delete the original network interface `flannel.1` by,

```
ifconfig flannel.1 down
ip link delete flannel.1
systemctl restart flannel
```

- When installing Minecraft client, we need to make sure that the X11 is completely installed. As the HMCL launcher can be opened correctly, I think the X11 has already been installed well, however, when I try to launch the Minecraft client, I got an error `java.lang.ArrayIndexOutOfBoundsException`. The suggestion of Mojang is updating/reinstalling the graphic driver, while in our case is installing X11 components completely, and type `xhost +` in the host machine.