CLSH 보고서

> 20191551 구형모

1장 CLSH 설계

옵션 1: --hostfile 옵션을 생략할 경우의 동작

-h 과 —hostfile 모두 생략되었다면, 다음과 같이 동작하도록 설계되었습니다.

1. CLSH_HOSTS 환경 변수에서 호스트이름 읽어오기, 콜론(:)으로 구분
2. CLSH_HOSTFILE 환경 변수에서 파일이름 읽어오기
    a. 기본 상대경로, / 로 시작하면 절대 경로
3. 현재 디렉토리에서 .hostfile 읽어오기
4. 위에 모든 것을 실패하면 `--hostfile 옵션이 제공되지 않았습니다` 출력

옵션 3: 출력 옵션 구현

`clsh -out=/tmp/run/ --err=/tmp/run/error/` 과 같이 옵션을 제공하면 PIPE를 통해 받아온 결과 값을 코드 내에서 직접 write를 수행하게 됩니다.

옵션 4: Interactive Mode

`clsh -i` 과 같이 옵션을 제공하면 인터렉티브 모드로 동작합니다. 이때 옵션 3의 출력 옵션은 무시되며, 콘솔로 출력됩니다. python의 subprocess의 `subprocess.Popen()` 과 `communicate()` 을 통해 구현되었습니다. 제공된 내부 쉘을 통해 매번 명령어를 실행할때마다 위 함수들을 불러 입력된 명령어의 수행결과를 결과가 도착한 순으로 표시합니다. 명령어 앞에 !를 붙이게 되면 LOCAL 쉘에서 동작하도록 설계되었습니다.

옵션 5: 종료

프로그램이 종료되면 각연결과 프로세스는 모두 종료됩니다. 또한 `signal.signal(signal.SIGTERM, handler)` 과 같이 핸들러를 통해 메인 프로세스가 시그널을 수신하면 다음과 같이 동작합니다.

- SIGTERM: 하던일 완료 후 종료
- SIGQUIT, SIGINT: 각 연결 모두 즉시 증지 후 종료

옵션 6: 예외

프로그램이 매 명령어를 실행할때 연결이 끊어지는것을 감지 합니다. 이는 명령어 실행시 경과 시간이 5초가 지나면 다음과 같이 출력하고 다른 모든 연결을 종료후 프로그램을 종료합니다.

ex) ERROR: node1 connection lost

2장 CLSH 구현

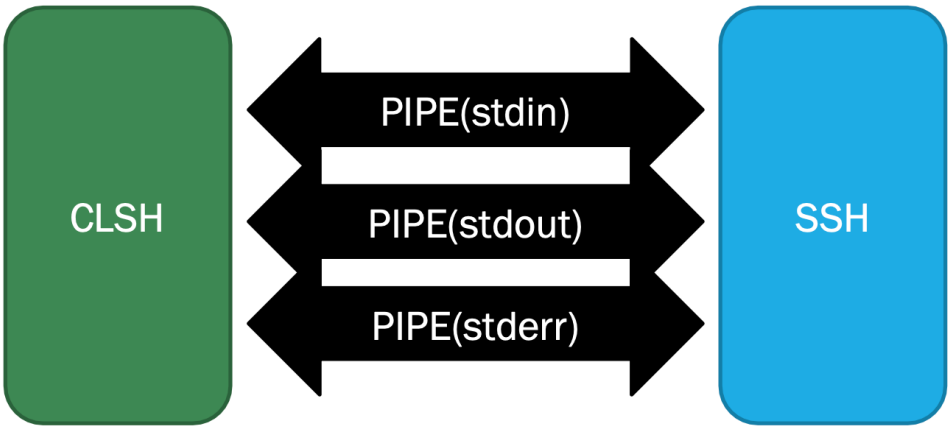
1절 기본구현

- Python 으로 구현되었습니다
- SSH 구현 (clsh.py 72줄 참조)
  - subprocess를 통해 SSH를 적극적으로 활용하였습니다

```
p = subprocess.Popen(f'ssh -i ssh-key.pem {user+"@" if user else "ubuntu@"}{h
ost} -T -o "StrictHostKeyChecking=no"',
                     stdin=subprocess.PIPE,
                     stdout=subprocess.PIPE,
                     stderr=subprocess.PIPE,
                     shell=True,
                     universal_newlines=True)
out, err = p.communicate(command)
if len(out):
    print("LOCAL:", out, end="")
elif len(err):
    print("LOCAL:", err, end="")
else:
    print("LOCAL:")
p.wait()
p.terminate()
```

  - `ssh -i *.pem` 옵션을 통해 pem 키를 핸들링 합니다. AWS 인스턴스는 다음 정보로 SSH 연결을 구축해야합니다
    - username: ec2-user
    - password 대신 pem 키
  - subprocess의 `Popen()` 을 통해 프로세스를 스폰하고, `communicate()` 를 통해 하위 프로세스에 명령을 실행합니다.
  - `ssh -T` 옵션을 통해 pseudo 터미널을 강제합니다. 로딩바등의 임의의 화면 기반 프로그램을 보여줄수 있습니다.

- PIPE를 사용하여 직접 연결 하였습니다.



- 원격의 다수에 쉘에 명령어를 실행하는 부분은 하위프로세스를 호스트 갯수만큼 실행하는 방법으로 해결하였습니다 (clsh.py 72줄 참조)

```
for host in hosts: # here
      p = subprocess.Popen(args)
      # do somthing...
```

- 옵션들과 원격실행 명령은 argparse를 통해 런타임에서 파싱되며, 프로그램은 이를 분석하여 주어진 상황에 맞도록 실행됩니다 (clsh.py 107줄 참조)
  - 아래 코드를 통해 명령어는 dictionary에 담기게 됩니다.
  - ex) `clsh -h 192.168.0.1` → `get_args()` → `option['h'] # 192.168.0.1`

```
def get_args():
    parser = argparse.ArgumentParser(add_help=False)
    parser.add_argument("--hostfile")
    parser.add_argument("--user")
    parser.add_argument("-h")
    parser.add_argument("-i", action='store_true')
    parser.add_argument("--out")
    parser.add_argument("--err")

    parsed = parser.parse_known_args()
    return vars(parsed[0]), " ".join(parsed[1])
```

2절 옵션 1: --hostfile 옵션을 생략할 경우의 동작

다음 코드를 통해 동작합니다. 위에서 아래로 순서대로 option이나 환경 변수들을 파싱합니다. (clsh.py 120줄 참조)

```
def execute(args, command):
    if args["h"]:
        hosts = args["h"].split(",")
        connect_SSH(hosts, command, args["i"], args["out"], args["err"], args["user"])
    elif args["hostfile"]:
        path = args["hostfile"]
        with open(path) as file:
            hosts = [line.rstrip() for line in file]
            connect_SSH(hosts, command, args["i"], args["out"], args["err"], args["use
r"])
    elif os.environ.get("CLSH_HOSTS"):
        print("Note: use CLSH_HOSTS environment")
        hosts = os.environ.get("CLSH_HOSTS").split(":")
        connect_SSH(hosts, command, args["i"], args["out"], args["err"], args["user"])
    elif os.environ.get("CLSH_HOSTFILE"):
        path = os.environ.get("CLSH_HOSTFILE")
        print(f"Note: use hostfile ‘{path}’ (CLSH_HOSTFILE env)")
        with open(path) as file:
            hosts = [line.rstrip() for line in file]
            connect_SSH(hosts, command, args["i"], args["out"], args["err"], args["use
r"])
    elif os.path.exists(".hostfile"):
        print("Note: use hostfile ‘.hostfile’ (default)")
        with open(".hostfile") as file:
            hosts = [line.rstrip() for line in file]
            connect_SSH(hosts, command, args["i"], args["out"], args["err"], args["use
r"])
    else:
        print("--hostfile 옵션이 제공되지 않았습니다")
```

- 옵션은 argparse를 통해 만들어진 dictionary value 입니다. 기본 구현 내 설명 참조
- 환경 변수는 python에서 제공하는 `os.environ.get()` 함수를 통해 가져옵니다
- 파일이 존재 하는지 여부는 `os.path.exists()` 를 통해 판단후 `with open(".hostfile") as file:` 를 통해 open 합니다
- 마지막 else문을 제외한 매 조건문 안에서 `connect_SSH()` 를 호출해 SSH 연결을 시도 합니다

3절 옵션 3: 출력 옵션 구현

명령어 실행을 통해 받아온 stdout과 stderr를 옵션에 따라 콘솔에 출력하거나 주어진 파일에 write 합니다. (clsh.py 80줄 참조)

```
stdout, stderr = p.communicate(command, timeout=5)
if len(stdout):
    if out:
        with open(out, "a+") as file:
            file.write(f"{host}: {stdout}")
    else:
```

```
        print(f"{host}:", stdout, end="")
elif len(stderr):
    if err:
        with open(err, "a+") as file:
            file.write(f"{host}: {stderr}")
    else:
        print(f"{host}:", stderr, end="")
else:
    if out:
        with open(out, "a+") as file:
            file.write(f"{host}:\n")
    else:
        print(f"{host}:")
```

- out, err 변수는 파일 이름입니다. 없다면(옵션으로 제공되지 않으면) 콘솔에 출력 합니다
- 만약 결과값이 비어있으나 에러는 아니라면 `node1:` 과 같이 출력하거나 파일에 write 합니다

4절 옵션 4: Interactive Mode

-i 옵션을 주게되면 내부쉘을 통해 각 호스트와 통신 할 수 있습니다. while loop을 통해 quit 이 입력될때 까지 입력을 받습니다. (clsh.py 14줄 참조)

```
def connect_SSH(hosts, command, interactive, out, err, user):
    print("Enter 'quit' to leave this interactive mode")
    print(f"Working with nodes: {', '.join(hosts)}")
    while True:
        inp = input("clsh> ")
        if inp == "quit":
            break

        print("------------------------")
        if inp.startswith("!"):
            p = subprocess.Popen(inp[1:],
                                 stdin=subprocess.PIPE,
                                 stdout=subprocess.PIPE,
                                 stderr=subprocess.PIPE,
                                 shell=True,
                                 universal_newlines=True)
            out, err = p.communicate(inp)
            if len(out):
                print("LOCAL:", out, end="")
            elif len(err):
                print("LOCAL:", err, end="")
            else:
                print("LOCAL:")
            p.wait()
            p.terminate()
        else:
            for host in hosts:
                p = subprocess.Popen(f'ssh -i ssh-key.pem {user+"@" if user else "ubun
```

```
tu@"}{host} -T -o "StrictHostKeyChecking=no"',
                                    stdin=subprocess.PIPE,
                                    stdout=subprocess.PIPE,
                                    stderr=subprocess.PIPE,
                                    shell=True,
                                    universal_newlines=True,
                                    preexec_fn=os.setsid)

                try:
                    out, err = p.communicate(inp, timeout=5)
                    if len(out):
                        print(f"{host}:", out, end="")
                    elif len(err):
                        print(f"{host}:", err, end="")
                    else:
                        print(f"{host}:")
                    p.wait()
                    p.terminate()
                except subprocess.TimeoutExpired:
                    os.killpg(os.getpgid(p.pid), signal.SIGTERM)
                    print(f"ERROR: {host} connection lost")
                    exit(0)
        print("------------------------")
```

- 연결시 `subprocess.Popen()`를 통해 파이프로 연결하며 `p.communicate()` 를 통해 명령어를 전달합니다. 이후 stdout과 stderr를 출력합니다
- 명령어를 quit이 입력되기 전까지 매 루프마다 실행합니다
  - 명령어: N개의 각 호스트에게 N번 ssh subprocess를 생성한후 각 노드별로 결과값을 출력합니다
  - !명령어: 로컬 쉘에서 실행합니다. `inp[1:]` 을 통해 !를 없애준후 해당 명령어를 실행하는 subprocess를 생성합니다.

5절 옵션 5: 종료

시그널 핸들러를 통해 각 시그널을 처리하며, 생성된 프로세스로 전달합니다 (clsh.py 148 줄 참조)

```
def handler(signum, frame):
    if signum == signal.SIGTERM:
        for p in p_pool:
            p.send_signal(signal.SIGTERM)
            p.wait()  # wait until shutdown
        exit(0)
    if signum == signal.SIGQUIT:
        for p in p_pool:
            p.send_signal(signal.SIGQUIT)
            os.kill(p.pid, signal.SIGQUIT)
        exit(0)
    if signum == signal.SIGINT:
```

```
        for p in p_pool:
            p.send_signal(signal.SIGINT)
            os.kill(p.pid, signal.SIGINT)
        exit(0)

signal.signal(signal.SIGTERM, handler)
signal.signal(signal.SIGQUIT, handler)
signal.signal(signal.SIGINT, handler)
```

- p_pool은 현재 동작하고 있는 process들이 담겨 있습니다. 다음과 같이 ssh 프로세스가 생성될때 append 되며, 프로세스가 종료될때 remove 됩니다. (clsh.py 53줄 참조)

```
p = subprocess.Popen()
p_pool.append(p)
# do something
p_pool.remove(p)
p.wait()
p.terminate()
```

- 프로세스가 실행되고 있는 `do something` 블럭 안에서 signal을 수신하면 signalhandler에서 해당 프로세스에게 시그널을 보냅니다
    - SIGQUIT과 SIGINT는 해당 시그널을 전송하고 바로 종료되며, SIGTERM는 해당 시그널을 전송하고 작업이 완료되기 까지 기다립니다. (이후 종료됨)

6절 옵션 6: 예외

타임아웃(5초)을 통해 매번 명령어 실행시 연결이 정상적으로 되었는지 확인하고 그렇지 않다면 자원을 회수하고 프로그램을 종료합니다

```
p = subprocess.Popen(f'ssh -i ssh-key.pem {user+"@" if user else "ubuntu@"}{host} -T -
o "StrictHostKeyChecking=no"',
                     stdin=subprocess.PIPE,
                     stdout=subprocess.PIPE,
                     stderr=subprocess.PIPE,
                     shell=True,
                     universal_newlines=True,
                     preexec_fn=os.setsid)

try:
    p_pool.append(p)
    out, err = p.communicate(inp, timeout=5) # here
    if len(out):
        print(f"{host}:", out, end="")
    elif len(err):
        print(f"{host}:", err, end="")
    else:
        print(f"{host}:")
    p.wait()
```

```
    p.terminate()
    p_pool.remove(p)
except subprocess.TimeoutExpired: # here
    os.killpg(os.getpgid(p.pid), signal.SIGTERM)
    print(f"ERROR: {host} connection lost")
    exit(0)
```

- 이때 메인 프로세스는 즉시 알아채고 `ERROR: {host} connection lost` 을 출력하고, 모든 연결을 종료하고 프로그램을 종료합니다.

7절 추가 옵션 : 유저 이름

ssh 연결시 기본적으로 ubuntu@hostname 로 연결하지만 `clsh --user username` 옵션을 주면 주어진 username을 이용하여 접속합니다

```
p = subprocess.Popen(f'ssh -i ssh-key.pem {user+"@" if user else "ubuntu@"}{host} -T -
o "StrictHostKeyChecking=no"',
                     stdin=subprocess.PIPE,
                     stdout=subprocess.PIPE,
                     stderr=subprocess.PIPE,
                     shell=True,
                     universal_newlines=True,
                     preexec_fn=os.setsid)
```

- AWS의 EC2 인스턴스는 ssh 사용시 `ec2-user`로 로그인 해야합니다

3장 CLSH 결과

> 💡 강의 자료 기준인 Linux 기준으로 작성하였습니다. (단, AWS EC2 환경의 리눅스를 사용합니다)

1절 설치하기

파이썬 설치

파이썬이 이미 설치 되어 있다면 건너뛰어도 됩니다

```
$ which amazon-linux-extras
```

만약 아무것도 뜨지 않는다면 패키지를 설치해줍니다.

```
$ sudo yum install -y amazon-linux-extras
```

파이썬 설치를 활성해줍니다.

```
$ sudo amazon-linux-extras enable python3.8
```

Python 설치

```
$ sudo yum install python3.8 -y
```

이제 파이썬을 사용할 준비가 되었습니다.

pyinstaller 설치

pyinstaller를 통해 실행 가능한 bin 파일로 컴파일 합니다.

```
$ pip3 install pyinstaller
```

pyinstaller 가 설치되었습니다.

소스 컴파일 & bin 파일 생성

소스코드가 위치한 곳으로 이동합니다.

```
$ cd 경로
```

아래 명령어를 통해 clsh bin 파일을 생성합니다.

```
$ pyinstaller clsh.py --onefile
```

첨부된 ssh-key.pem 파일을 dist 폴더로 옮겨준후 400 권한을 부여합니다

```
$ mv ssh-key.pem dist/
$ cd dist
$ chmod 400 ssh-key.pem
```

clsh에 실행권한을 부여합니다

이제 dist/ 디렉토리내에 clsh 파일이 생성되었습니다. 다음을 통해 프로그램을 실행할 수 있습니다.

```
$ cd dist
$ ./clsh [옵션] 명령어
```

2절 빠르게 시작하기

AWS EC2에 SSH 요청을 받을수 있는 3개의 인스턴스를 미리 생성해 두었습니다.
username은 ec2-user로 접속해야하며(추가 옵션이므로 아래서 기술예정), 프로그램이 자동으로 첨부된 pem key를 사용합니다.

Hosts

- ec2-user@13.124.15.148
- ec2-user@3.38.165.191
- ec2-user@43.201.19.126

중요) 유저네임으로 접속하기

예제에 사용된 호스트는 유저네임과 ssh key를 필요로 합니다. 이에 `--user` 옵션을 통해 유저네임을 전달합니다. 프로그램은 유저네임과 pem 키를 활용하여 ssh 접속을 시도합니다.

만약 입력하지 않으면 유저네임은 기본적으로 **ubuntu** 입니다.

```
$ ./clsh --user ec2-user [command]
```

노드별로 명령어 수행결과 읽어오기

주어진 호스트의 /proc/loadavg를 읽어온다고 가정해봅시다.

- `-h` 옵션으로 호스트들을 입력할수 있습니다(쉼표로 구분됨)

```
$ ./clsh --username ec2-user -h 13.124.15.148,3.38.165.191,43.201.19.126 cat /pro
c/loadavg
```

```
❯ ./clsh --user ec2-user -h 13.124.15.148,3.38.165.191,43.201.19.126 cat /proc/loadavg
13.124.15.148: 0.00 0.00 0.00 1/130 5658
3.38.165.191: 0.00 0.00 0.00 1/125 9323
43.201.19.126: 0.00 0.00 0.00 1/131 11024
```

- hostfile을 통해 호스트들을 입력할수 있습니다 (개행으로 구분됨)

```
13.124.15.148
3.38.165.191
43.201.19.126
```

```
$ ./clsh --user ec2-user --hostfile ./hostfile cat /proc/loadavg
```

```
❯ ./clsh --user ec2-user --hostfile ./hostfile cat /proc/loadavg
13.124.15.148: 0.00 0.00 0.00 1/130 5713
3.38.165.191: 0.00 0.00 0.00 1/126 9440
43.201.19.126: 0.00 0.00 0.00 1/131 11081
```

3절 사용방법

옵션1: --hostfile 옵션을 생략할 경우

만약 `-h` 옵션도 없고, hostfile도 생략되었다면 다음과 같이 동작합니다. 나열된 번호순대로 프로그램이 조건을 확인합니다.

1. **CLSH_HOSTS** 환경변수에서 읽어옵니다.

```
$ CLSH_HOSTS=13.124.15.148:3.38.165.191:43.201.19.126 ./clsh --user ec2-user cat /
proc/loadavg
```

```
❯ CLSH_HOSTS=13.124.15.148:3.38.165.191:43.201.19.126 ./clsh --user ec2-user cat /proc/loadavg
Note: use CLSH_HOSTS environment
13.124.15.148: 0.00 0.00 0.00 1/130 5764
3.38.165.191: 0.00 0.00 0.00 1/125 9543
43.201.19.126: 0.00 0.00 0.00 1/131 11129
```

2. **CLSH_HOSTFILE** 환경 변수에서 파일이름 읽어옵니다

```
❯ ls
clsh        hostfile    ssh-key.pem
❯ CLSH_HOSTFILE=./hostfile ./clsh --user ec2-user cat /proc/loadavg
Note: use hostfile './hostfile' (CLSH_HOSTFILE env)
13.124.15.148: 0.05 0.01 0.00 1/131 5807
3.38.165.191: 0.00 0.00 0.00 1/129 9592
43.201.19.126: 0.00 0.00 0.00 1/133 11223
```

3. 현재 디렉토리에 위치한 **.hostfile** 에서 읽어옵니다.

```
$ ./clsh --user ec2-user cat /proc/loadavg
```

```
❯ ls -a
.           ..          .hostfile   clsh        ssh-key.pem
❯ ./clsh --user ec2-user cat /proc/loadavg
Note: use hostfile '.hostfile' (default)
13.124.15.148: 0.00 0.00 0.00 1/130 5844
3.38.165.191: 0.00 0.00 0.00 1/126 9631
43.201.19.126: 0.00 0.00 0.00 1/131 11260
```

4. 위의 모든 것을 실패한다면 에러처리 합니다

```
❯ ./clsh --user ec2-user cat /proc/loadavg
--hostfile 옵션이 제공되지 않았습니다
```

옵션3: 출력 옵션 구현

기본적으로 결과나 에러를 콘솔로 출력하지만, 이를 파일로 출력할수도 있습니다.

```
$ ./clsh --user ec2-user --out=out --err=err [command]
```

```
❯ ./clsh --user ec2-user --out=out --err=err cat /proc/loadavg
Note: use hostfile '.hostfile' (default)
❯ cat out
13.124.15.148: 0.00 0.00 0.00 1/132 6009
3.38.165.191: 0.00 0.00 0.00 1/129 9756
43.201.19.126: 0.25 0.06 0.02 1/135 11376
```

```
❯ ./clsh --user ec2-user --out=out --err=err dsfajdfwrongcommand
Note: use hostfile '.hostfile' (default)
❯ cat err
13.124.15.148: -bash: line 1: dsfajdfwrongcommand: command not found
3.38.165.191: -bash: line 1: dsfajdfwrongcommand: command not found
43.201.19.126: -bash: line 1: dsfajdfwrongcommand: command not found
```

옵션4: Interactive Mode 구현

Interactive 모드를 사용하여 내부 쉘을 통해 각 노드와 통신할 수 있습니다. quit을 입력해 종료합니다. !명령어를 입력하면 로컬 쉘에서 실행합니다.

```
$ ./clsh -i
```

```
❯ ./clsh --user ec2-user -i
Note: use hostfile '.hostfile' (default)
Enter 'quit' to leave this interactive mode
Working with nodes: 13.124.15.148, 3.38.165.191, 43.201.19.126
clsh> uname
------------------------
13.124.15.148: Linux
3.38.165.191: Linux
43.201.19.126: Linux
------------------------
clsh> pwd
------------------------
13.124.15.148: /home/ec2-user
3.38.165.191: /home/ec2-user
43.201.19.126: /home/ec2-user
------------------------
clsh> abcde
------------------------
13.124.15.148: -bash: line 1: abcde: command not found
3.38.165.191: -bash: line 1: abcde: command not found
43.201.19.126: -bash: line 1: abcde: command not found
------------------------
clsh> !uname
------------------------
LOCAL: Darwin
------------------------
clsh> !ls
------------------------
LOCAL: clsh
err
out
ssh-key.pem
------------------------
clsh> quit
```

옵션5: 종료

프로그램은 시그널을 받으면 해당 시그널에 맞는 행동을 하도록 설계되었습니다. 2장 5절에서 확인할 수 있습니다.

옵션6: 예외

SSH 연결중 일부 연결이 끊어지면 프로그램은 즉시 알아채고 에러메세지를 출력한 후, 다른 모든 연결을 종료합니다.

```
❯ ./clsh --user ec2-user -i
Note: use hostfile '.hostfile' (default)
Enter 'quit' to leave this interactive mode
Working with nodes: 113.124.145.147, 3.38.165.191, 43.201.19.126
clsh> ls
------------------------
ERROR: 113.124.145.147 connection lost
```