운영 체제의 실제 프로젝트

20152842 유남규

1. CLSH 설계

선택한 옵션 : 1, 2, 3, 4, 6

사용자가 프롬프트에 입력하면 모든 옵션들이 동시에 고려되도록 설계를 하였습니다. 모든 통신들은 subprocess.Popen함수로 ssh를 구현하여 동작하며, 모든 통신들은 응답이 오는 순서대로 출력을 하기 위해 concurrent 모듈을 통해 구현하였습니다.

옵션 1 (--hostfile옵션을 생략)

'—hostfile' 옵션은 '—host' 옵션이 없는 경우에만 동작하도록 하였습니다. 두 옵션을 동시에 사용하면 '—host'옵션만 동작하도록 하였습니다.

즉, 옵션 1은 '—host'옵션과 '—hostfile'옵션이 둘 다 생략될 경우입니다.

'—hostfile'옵션이 생략되었다면 'CLSH_HOSTS'와 'CLSH_HOSTFILE' 환경 변수를 'os.environ.get' 함수를 사용하여 가져옵니다.

'CLSH_HOSTS'는 '노드1:노드2:노드3'형식의 환경 변수이고 CLSH_HOSTFILE은 'hostfile' 파일이 위치한 경로를 가지고 있는 환경 변수입니다.

CLSH_HOSTS의 환경 변수 값이 존재하면(ex : '노드1:노드2:노드3') 노드1, 노드2, 노드3 에 연결을 진행합니다.

CLSH_HOSTS의 환경 변수 값이 존재하지 않으면 CLSH_HOSTFILE의 값 경로에서 'hostfile'을 찾고 파일이 존재하면 그 파일에서 노드를 읽어온 후 연결을 진행합니다. 'hostfile'의 구성은 각 라인마다 '노드1', '노드2', '노드3' 의 형식으로 되어 있습니다.

위 두 환경 변수의 값이 존재하지 않으면 현재 실행 위치에서 'hostfile'을 찾고 존재하면 파일에서 노드를 읽어온 후 연결을 진행합니다.

만약 위 3가지 조건을 모두 충족하지 못 한다면 에러 메시지를 출력하게 됩니다.

옵션 2 (쉘 리다이렉션 구현)

파이프를 사용하여 프롬프트에 입력할 시 바로 입력이 된다는 점에서 착안하여, 'select'함수를 사용하여 구현하였습니다.

'select 모듈의 select'함수의 반환 값 중 하나인 'read_fds'가 읽기 조건을 반환하는데 이 값이 True 일 때 파이프를 사용하였다고 판단하는 로직으로 구현하였습니다.

위 조건이 충족되면 읽어온 값을 로컬 특정 경로의 파일에 저장하고 ssh를 통해 연결할 때 그 파일을 열어 입력으로 전달합니다.

옵션 3(출력 옵션 구현)

입력 옵션 '—out'과 '—err'는 각자 사용이 가능하며 ssh 통신시 'subprocess.Popen'을 사용하였기에 'Popen'에 'commuincate'함수를 적용하여 'stdout'과 'stderr' 리턴 값을 out과 err 경로에 지정된 파일 형식으로 값을 넣었습니다.
옵션이 입력될 때 옵션의 인자 경로에 폴더가 없다면 폴더를 생성 후에 out과 err의 저장 시도를 합니다. 또한 out과 err 각각 값이 비어 있는지 판단하고 비어 있다면 저장하지 않게 됩니다.

옵션 4(Interactive Mode 구현)

모든 옵션이 입력이 되지 않거나 '—i' 옵션만 입력이 되면 interactive 모드로 구동을 합니다.
'—I'옵션이 입력이 되면 통신할 하나의 노드를 선택하거나 전체 노드와 통신하는 지를 선택할 수 있습니다. 한 개의 노드를 입력하면 그 노드와 개별적으로 통신을 하고, 빈칸을 제출하면 모든 노드와 통신을 하게 됩니다.
옵션을 입력하지 않는다면 실행하는 명령어를 로컬에서 입력을 받아 그 입력을 다시 각 노드들에게 전달하여 실행을 하게 됩니다.

2. CLSH 구현

A. 기본 구현

```
385
386 if __name__ == '__main__':
387     # attr = ['host', "hostfile", "out", "err"]
388     args, command = makeParser()
389     all_Nodes = get_node_info()
390
391     option2()
392
393     # Basic
394     for attr in vars(args):
395         if attr == "host" and error_AttrAndNone(args, attr):
396             basic_host()
397         if attr == "hostfile" and error_AttrAndNone(args, attr):
398             basic_hostfile()
399     # Option 1 : hostfile 생략
400     option1()
401
402
```

```python
def makeParser():
    parser = argparse.ArgumentParser(description="Input options")

    parser.add_argument("--host")
    parser.add_argument("--hostfile")
    parser.add_argument("--out")
    parser.add_argument("--err")
    parser.add_argument("-i", action="store_true")

    args, command = parser.parse_known_args()

    combined = ' '.join(command)
    return args, combined
```

```python
def get_node_info():
    container_details = []
    # 컨테이너 정보에서 호스트 노드의 컨테이너 이름과 포트 가져오기
    container_info = get_container_info()
    for container in container_info:
        container_name = container[0]

        # 컨테이너 이름을 이용하여 컨테이너의 IP 주소 가져오기
        ip_result = subprocess.run(["docker", "inspect", "-f",
                                    "{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}", container_name],
                                   capture_output=True, text=True)
        container_ip = LOCALHOST

        # 포트 가져오기
        container_port = container[-1]

        container_details.append((container_name.decode('utf-8').split('-')[1],
                                  container_ip, container_port.decode('utf-8').split(':')[-1].split('->')[0]))

    return container_details

def get_container_info():
    # Docker Compose 명령어 실행하여 컨테이너 정보 가져오기
    result = subprocess.run(["docker-compose", "ps"], capture_output=True, text=False)
    output_lines = result.stdout.splitlines()
    container_info = [line.split() for line in output_lines[1:]]  # 컨테이너 정보 파싱
    return container_info
```

```python
def basic_host():
    nodes = args.host.split(",")
    target_nodes = [find_by_attr(all_Nodes, node) for node in nodes]
    check_null_nodes(target_nodes)
    simple_CLSH(target_nodes, command)
    quit()

def basic_hostfile():
    hostfile_nodes = read_hostfile(args.hostfile)
    target_nodes = get_nodes_from_hostfile(all_Nodes, hostfile_nodes)
    check_null_nodes(target_nodes)
    simple_CLSH(target_nodes, command)
    quit()
```

```python
if not(args.i):
  with concurrent.futures.ThreadPoolExecutor(max_workers=len(nodes)) as executor:
      results = {executor.submit(ssh_command, node[1], node[2], command): node for node in nodes}
      for future in concurrent.futures.as_completed(results):
          node = results[future]
          # try:
          output, error = future.result()
          # Option 3
          option3(output, error, node[0])
          print(f"{node[0]}: {output.decode('utf-8')}")
      quit()
```

```python
def ssh_command(container_ip, container_port, command):
  ssh_cmd = subprocess.Popen(['ssh', container_ip, '-p', container_port, command],
                             stdin=subprocess.PIPE,
                             stdout=subprocess.PIPE,
                             stderr=subprocess.PIPE)
  stdout, stderr = ssh_cmd.communicate()
  return stdout, stderr
```

기본 구현인 '—host'와 '—hostfile' 옵션 사용 시의 코드입니다.

파일을 실행하면 makeParser()함수로 필요한 옵션들과 커맨드를 파싱합니다. 그리고 get_node_info()함수로 'docker ps' 명령어를 입력하여 현재 동작 중인 컨테이너들의 정보를 가져옵니다.

파싱한 옵션들 중 '—host' 옵션이 존재하면 'basic_host'로, '—host' 옵션이 존재하지 않고 '—hostfile' 옵션이 존재하면 'basic_hostfile'로 이동합니다.

basic_host는 커맨드 라인에 입력한 노드가 현재 가동중인 노드에 있는지 확인하고, basic_hostfile은 파일에 있는 노드가 현재 가동중인 노드에 있는지 확인합니다.

ssh연결은 'subprocess.Popen'함수를 통해 구현하였습니다. 출력과 에러 출력은 commnuicate를 통해 받아와서 따로 종료를 하지 않아도 자동으로 종료되게 하였습니다.

응답이 오는 순서대로 출력하기 위해 concurrent를 사용하여 구현하였습니다.

B. 옵션 구현

i. –hostfile 옵션 생략

```python
if __name__ == '__main__':
  # attr = ['host', "hostfile", "out", "err"]
  args, command = makeParser()
  all_Nodes = get_node_info()

  option2()

  # Basic
  for attr in vars(args):
    if attr == "host" and error_AttrAndNone(args, attr):
      basic_host()
    if attr == "hostfile" and error_AttrAndNone(args, attr):
      basic_hostfile()
  # Option 1 : hostfile 생략
  option1()
```

옵션 1은 host와 hostfile 옵션 둘 다 없어야 하므로 main에서 마지막에 위치한 option1()에서 실행됩니다.

```
def option1():
  if args.host == None and args.hostfile == None:
    CLSH_HOSTS = os.environ.get('CLSH_HOSTS')
    CLSH_HOSTFILE = os.environ.get('CLSH_HOSTFILE')
    if(CLSH_HOSTS != None):
      print("CLSH_HOSTS env exist.")
      # export CLSH_HOSTS=work01:work02:work03:work04
      nodes = get_nodes_from_CLSH_HOSTS(CLSH_HOSTS)
      target_nodes = [find_by_attr(all_Nodes, node) for node in nodes]

      check_null_nodes(target_nodes)
      simple_CLSH(target_nodes, command)

    elif(CLSH_HOSTFILE != None):
      print("CLSH_HOSTS env doesn't exist. Finding CLSH_HOSTFILE env...")
      path = get_hostfile_path(CLSH_HOSTFILE)
      nodes = read_hostfile(path)
      target_nodes = [find_by_attr(all_Nodes, node) for node in nodes]

      check_null_nodes(target_nodes)
      simple_CLSH(target_nodes, command)
```

```
  elif not(os.path.isfile(HOSTFILE_CURRENTDIR)):
    print("CLSH_HOSTS and CLSH_HOSTFILE env don't exist. Finding hostfile...")
    hostfile_nodes = read_hostfile(HOSTFILE_CURRENTDIR)
    target_nodes = get_nodes_from_hostfile(all_Nodes, hostfile_nodes)

    check_null_nodes(target_nodes)
    simple_CLSH(target_nodes, command)
  else:
    print("--hostfile 옵션이 제공되지 않았습니다")
    quit()
```

조건으로 host와 hostfile이 없을 때에 실행이 됩니다.

실행이 되면 CLSH_HOSTS와 CLSH_HOSTFILE 환경 변수 값을 불러옵니다.

만약 CLSH_HOSTS의 값이 존재하면 위 기본 구현한 방식에 노드를 전달하여 동작합니다. 값이 없다면 CLSH_HOSTFILE의 값으로 'hostfile'파일을 찾습니다. 파일을 찾으면 파일 안의 노드 값들을 위의 기본 구현한 방식에 전달하여 동작합니다. CLSH_HOSTFILE의 값도 없다면 현재 위치에서 'hostfile'을 찾고 파일을 찾으면 파일 안의 노드 값들을 위의 기본 구현한 방식에 전달하여 동작합니다. 위의 조건들이 모두 일치하지 않으면 hostfile 옵션 에러 메시지를 출력하고 종료합니다.

ii. 쉘 리다이렉션

```
385
386 if __name__ == '__main__':
387     # attr = ['host', "hostfile", "out", "err"]
388     args, command = makeParser()
389     all_Nodes = get_node_info()
390
391     option2()
392
393     # Basic
394     for attr in vars(args):
395         if attr == "host" and error_AttrAndNone(args, attr):
396             basic_host()
397         if attr == "hostfile" and error_AttrAndNone(args, attr):
398             basic_hostfile()
399     # Option 1 : hostfile 생략
400     option1()
401
402
```

프롬프트에 커맨드를 입력할 때 파이프를 사용하게 되면 sys가 입력을 바로 받게 되므로 명령어에 파이프가 존재하는지에 대한 판별을 가장 먼저 하기 위해 option2()의 함수를 가장 먼저 동작하도록 하였습니다.

```
def option2():
    # print("Finding PIPE...")
    # 입력이 있는지 확인하기 위해 select 사용
    pipe_input, _, _ = select.select([sys.stdin], [], [], 0)

    if pipe_input:
        file_path = os.getenv('PIPE_INPUT_FILE', './pipe_input.txt')
        save_pipe_input_to_file(file_path, pipe_input[0].read())

        for attr in vars(args):
            if attr == "host" and error_AttrAndNone(args, attr):
                basic_host_Redirection(file_path)
            if attr == "hostfile" and error_AttrAndNone(args, attr):
                basic_hostfile_Redirection(file_path)
        # Option 1 : hostfile 생략
        option1_Redirection(file_path)
        quit()
    else:
        print("No PIPE. Passing...")
```

Select 함수를 통해 입력 조건이 True이면 명령어에 파이프를 사용하였다고 판별하였습니다.

파이프가 존재하면 출력 값을 저장하기 위해 현재 위치에 'pipe_input.txt' 파일을 만들고 그 파일에 파이프의 출력 값을 넣었습니다.

그 후의 동작들은 기본 구현과 한 부분 빼고 전부 동일합니다. 다른 부분은 아래와 같습니다.

```python
def ssh_command(container_ip, container_port, command):
    ssh_cmd = subprocess.Popen(['ssh', container_ip, '-p', container_port, command],
                               stdin=subprocess.PIPE,
                               stdout=subprocess.PIPE,
                               stderr=subprocess.PIPE)
    stdout, stderr = ssh_cmd.communicate()
    return stdout, stderr

def ssh_command_Redirection(container_ip, container_port, command, file_path):
    ssh_cmd = subprocess.Popen(['ssh', container_ip, '-p', container_port, command],
                               stdin=read_pipe_input_to_file(file_path),
                               stdout=subprocess.PIPE,
                               stderr=subprocess.PIPE,
                               )
    stdout, stderr = ssh_cmd.communicate()
    return stdout, stderr
```

위의 함수가 기본 구현에 쓰이는 함수이고 아래 Redirection이 파이프 사용시에 쓰이는 함수입니다.

```python
def save_pipe_input_to_file(file_path, pipe_input):
    # 파일에 입력 저장
    with open(file_path, 'w') as file:
        file.write(pipe_input)

def read_pipe_input_to_file(file_path):
    # 파일로부터 입력을 읽어옴
    return open(file_path, 'r')
```

파이프 사용시에 출력 값을 저장한 파일을 읽어온 것을 open 하여 stdin의 값으로 전달하여 동작하도록 하였습니다.

iii. err, out 옵션

err와 out 구현은 위의 구현들과 호환되도록 작성하였습니다. 그렇기에 위의 기본 구현을 기반으로 동작합니다.

```python
if not(args.i):
    with concurrent.futures.ThreadPoolExecutor(max_workers=len(nodes)) as executor:
        results = {executor.submit(ssh_command, node[1], node[2], command): node for node in nodes}
        for future in concurrent.futures.as_completed(results):
            node = results[future]
            # try:
            output, error = future.result()
            # Option 3
            option3(output, error, node[0])
            print(f"{node[0]}: {output.decode('utf-8')}")
        quit()
```

이 부분에서 ssh의 결과를 받아올 때, 리턴 값인, stdout, stderr를 output과 error 매칭하여 option3()이 동작하도록 합니다.

```
if args.out is not None:
    if not os.path.exists(args.out) :
        os.makedirs(args.out)
    output_file_path = args.out + node + ".out"
else:
    output_file_path = None

if args.err is not None:
    if not os.path.exists(args.err):
        os.makedirs(args.err)
    error_file_path = args.err + node + ".err"
else:
    error_file_path = None

output_len = len(output.decode())
error_len = len(error.decode())

if output_len != 0 and error_len != 0 and output_file_path is not None
    and error_file_path is not None:
    # 표준 출력과 표준 에러 모두 파일에 저장
    with open(output_file_path, 'w') as out_file, open(error_file_path, 'w') as err_file:
        out_file.write(output.decode())
        err_file.write(error.decode())
elif output_len != 0 and error_len == 0 and output_file_path is not None:
    # 표준 출력만 파일에 저장
    with open(output_file_path, 'w') as out_file:
        out_file.write(output.decode())
elif output_len == 0 and error_len != 0 and error_file_path is not None:
    # 표준 에러만 파일에 저장
    with open(error_file_path, 'w') as err_file:
        err_file.write(error.decode())
```

option3()함수가 실행이 되면 out과 err 옵션이 사용되었는지 판별을 합니다. 사용이 되었다면 out옵션에 인자로 전달된 저장 경로들의 파일이 존재하는지 판별하여 없다면 파일을 생성하고 저장될 파일의 이름을 완성합니다. err옵션도 동일하게 동작합니다.

그리고 받아온 output과 error의 길이로 비어 있는지 아닌지를 판단합니다. 즉, out옵션을 사용하고 출력의 결과가 있을 때 에만 결과를 파일로 저장합니다. Err 옵션도 out옵션과 동일하게 동작합니다.

iv. -I 옵션

-I 옵션은 ssh 통신 직전에 판별하도록 구현하였습니다.

```
if args.i:
    print("Enter 'quit' to leave this interactive mode")
    print("Working with nodes: " + node_names_join)
    print("If you want to communicate with a single container, please enter its name.")
    print("If you leave it blank, communication will be established with all containers.")
    node_input = input("Node Input : ")
    while True:
        if node_input is not None and find_by_attr(nodes, node_input):
            command = input("clsh > ")
            if command == "quit":
                quit()
            node_info = find_node_by_name(nodes, node_input)
            output, error = ssh_command(node_info[1], node_info[2], command)

            option3(output, error, node_info[0])
            print(f"{node_info[0]}: {output.decode('utf-8')}")

        elif len(node_input) != 0 and not(find_by_attr(nodes, node_input)):
            print("Input node just one exactly")
            node_input = input("Node Input : ")

        elif len(node_input) == 0:
```

```
command = input("clsh > ")
if command == "quit":
    quit()
with concurrent.futures.ThreadPoolExecutor(max_workers=len(nodes)) as executor:
    results = {executor.submit(ssh_command, node[1], node[2], command): node for node in nodes}
    for future in concurrent.futures.as_completed(results):
        node = results[future]
        output, error = future.result()
        option3(output, error, node[0])
        print(f"{node[0]}: {output.decode('utf-8')}")
```

옵션에 I 가 입력이 된다면 현재 연결 가능한 노드들을 나열합니다. 그 중 하나만 입력을 하면 그 노드와 ssh 통신을 하고 입력을 빈칸으로 하게 된다면 모든 노드들과 ssh 통신을 합니다.

올바른 노드가 입력되지 않는다면 올바르게 입력할 때 까지 반복되고, 연결 후 명령어가 'quit'이 입력되기 전까지 동작합니다.

```
if not (args.host or args.hostfile or args.out or args.err or args.i):
  print("Enter 'quit' to leave this interactive mode")
  print("Working with nodes: " + node_names_join)
  while True:
    command = input("clsh local > ")
    if command == "quit":
      quit()
    # local_command_stdout = local_command(command)
    with concurrent.futures.ThreadPoolExecutor(max_workers=len(nodes)) as executor:
        results = {executor.submit(ssh_command, node[1], node[2], command): node for node in nodes}
        for future in concurrent.futures.as_completed(results):
            node = results[future]
            # try:
            output, error = future.result()
            # Option 3
            print(f"{node[0]}: {output.decode('utf-8')}")
```

아무 옵션도 입력하지 않는다면 자동적으로 옵션 2의 방식처럼 연결할 노드를 찾아 ssh 연결을 하게 됩니다.

3. CLSH 결과

A. 설치하기

i. 기본 구현 되어있던 도커의 설정에 아래의 그림과 같이 설정을 추가한다.

```
# 로컬의 rsa를 컨테이너에 저장
COPY ./id_rsa.pub /root/id_rsa.pub
RUN ssh-keygen -t rsa -b 4096 -N "" -f /root/.ssh/id_rsa \
        && touch /root/.ssh/authorized_keys \
        && chmod 644 /root/.ssh/authorized_keys \
        && cat /root/id_rsa.pub >> /root/.ssh/authorized_keys \
        && rm /root/id_rsa.pub
```

그 후에 로컬에서 'ssh-keygen' 명령어를 실행하여 id-rsa키를 생성하고 도커를 가동한다. 각 컨테이너에 /root/.ssh/authorized_keys 가 생성되고 그 안에 값이 로컬의 id_rsa.pub 내용과 같은 지 확인한다.

ii. 'apt-get install python3'를 통해 파이썬을 설치후에 'python3 clsh.py [options] 명령어를 통해 실행한다.

B. 빠르게 시작하기

i. 기본 구현

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py --host work01,work02 cat /proc/loadavg
No PIPE. Passing...
work01: 0.00 0.01 0.00 1/842 834

work02: 0.00 0.01 0.00 1/839 464
```

option : --host, 노드1/노드2/노드3, 원격에 실행할 명령어

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py --hostfile ./hostfile cat /proc/loadavg
No PIPE. Passing...
work04: 0.00 0.00 0.00 2/848 620

work03: 0.00 0.00 0.00 5/847 550

work01: 0.00 0.00 0.00 1/843 837

work02: 0.00 0.00 0.00 1/841 467
```

option : --hostfile, 'hostfile' 파일 경로, 원격에 실행할 명령어

ii. 옵션 1(hostfile 생략 옵션)

```
root@DESKTOP-9S2ICBK:/CLSH# env | grep CLSH
PWD=/CLSH
CLSH_HOSTS=work01:work03:work04
CLSH_HOSTFILE=/root/CLSH/hostfile
```

위 CLSH_HOSTS, CLSH_HOSTFILE 두 환경 변수 설정을 한 후 명령어 입력.

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py cat /proc/loadavg
No PIPE. Passing...
CLSH_HOSTS and CLSH_HOSTFILE env don't exist. Finding hostfile...
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work02, work03, work04
clsh local > ls -al
work03: total 24
drwx------ 1 root root 4096 Dec  7 19:10 .
drwxr-xr-x 1 root root 4096 Dec  7 05:29 ..
-rw-r--r-- 1 root root 3106 Oct 15  2021 .bashrc
-rw-r--r-- 1 root root  161 Jul  9  2019 .profile
drwx------ 2 root root 4096 Dec  7 05:29 .ssh
-rw------- 1 root root  679 Dec  7 19:10 .viminfo
```

option : 원격에 실행할 명령어

iii. 옵션 2 (쉘 리다이렉션)

```
root@DESKTOP-9S2ICBK:/CLSH# ls /etc/*.conf | python3 clsh.py --hostfile ./hostfile cat
work03: /etc/adduser.conf
/etc/ca-certificates.conf
/etc/debconf.conf
/etc/deluser.conf
/etc/e2scrub.conf
/etc/fuse.conf
/etc/gai.conf
/etc/hdparm.conf
/etc/host.conf
/etc/ld.so.conf
```

option : '파이프에 실행할 명령어', '|', 원격에서 실행할 명령어

iv. 옵션 3 (출력 옵션 구현)

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py --out ./ --err ./ cat /proc/loadavg
No PIPE. Passing...
CLSH_HOSTS and CLSH_HOSTFILE env don't exist. Finding hostfile...
work04: 0.00 0.00 0.00 1/848 638

work01: 0.00 0.00 0.00 4/848 855

work02: 0.00 0.00 0.00 3/848 485

work03: 0.00 0.00 0.00 4/848 568

root@DESKTOP-9S2ICBK:/CLSH# ls
Dockerfile  docker-compose.yml  hostfileDir  pipe_input.txt  work01.out  work03.out
clsh.py     hostfile            id_rsa.pub   temp_file.txt   work02.out  work04.out
```

option : '—out' '노드이름.out 파일이 저장될 경로', '—err', '노드이름.err 이름이 저장될 경로', 원격에서 실행될 명령어

v. 옵션 4 (Interactive Mode)

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py -i
No PIPE. Passing...
CLSH_HOSTS and CLSH_HOSTFILE env don't exist. Finding hostfile...
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work02, work03, work04
If you want to communicate with a single container, please enter its name.
If you leave it blank, communication will be established with all containers.
Node Input : work01
clsh >
```

option: '-I',

vi.

```
root@DESKTOP-9S2ICBK:/CLSH# unset CLSH_HOSTS
root@DESKTOP-9S2ICBK:/CLSH# unset CLSH_HOSTFILE
root@DESKTOP-9S2ICBK:/CLSH# env | grep CLSH
PWD=/CLSH
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py cat /proc/loadavg
No PIPE. Passing...
--hostfile 옵션이 제공되지 않았습니다
```

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py
No PIPE. Passing...
CLSH_HOSTS env exist.
Note : use CLSH_HOSTS env
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work03, work04
clsh local > ls -al
work01: total 32
drwx------ 1 root root 4096 Dec  7 19:10 .
drwxr-xr-x 1 root root 4096 Dec  7 05:29 ..
-rw------- 1 root root   95 Dec  7 18:30 .bash_history
-rw-r--r-- 1 root root 3106 Oct 15  2021 .bashrc
-rw-r--r-- 1 root root  161 Jul  9  2019 .profile
drwx------ 1 root root 4096 Dec  7 17:57 .ssh
-rw------- 1 root root 1769 Dec  7 19:10 .viminfo

work04: total 28
drwx------ 1 root root 4096 Dec  7 19:10 .
drwxr-xr-x 1 root root 4096 Dec  7 05:29 ..
-rw------- 1 root root   13 Dec  7 18:49 .bash_history
-rw-r--r-- 1 root root 3106 Oct 15  2021 .bashrc
-rw-r--r-- 1 root root  161 Jul  9  2019 .profile
drwx------ 2 root root 4096 Dec  7 05:29 .ssh
-rw------- 1 root root  679 Dec  7 19:10 .viminfo
```

'python3 clsh.py' 명령어를 입력하면 가장 빠르게 실행이 가능하다. (옵션4의

C. 사용 방법

모든 명령어의 시작은 'python3 clsh.py'로 시작한다. (옵션 2 쉘 리다이렉션은 제외)

i. 기본 구현

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py --host work01,work02 cat /proc/loadavg
No PIPE. Passing...
work02: 0.00 0.00 0.00 3/844 491

work01: 0.00 0.00 0.00 2/844 861
```

입력할 명령어 : --host '노드' '명령어'

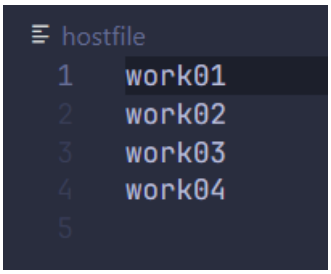
노드의 입력 형식은 연결된 컨테이너 이름이고 띄어쓰기 없이 쉼표로 구분하여 입력한다.

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py --hostfile ./hostfile cat /proc/loadavg
No PIPE. Passing...
work01: 0.00 0.00 0.00 7/848 864

work02: 0.00 0.00 0.00 3/847 494

work04: 0.00 0.00 0.00 3/844 641

work03: 0.00 0.00 0.00 2/844 571
```

입력할 명령어 : --hostfile '파일 경로' '명령어'

Hostfile 옵션의 인자로 사용할 파일 경로를 적어 넣는다. Hostfile의 내용은 아

래 그림과 같이 한 라인마다 노드를 적어 넣는 방식으로 구분하여 작성한다.



ii. 옵션 1 (hostfile 생략)

```
root@DESKTOP-9S2ICBK:/CLSH# env | grep CLSH
PWD=/CLSH
CLSH_HOSTS=work01:work03:work04
CLSH_HOSTFILE=/root/CLSH/hostfile
```

사용하기 전에 'export env '환경변수이름'='값'' 설정을 통해 환경 변수의 값을 미리 지정해 주어야 한다. CLSH_HOSTS 에는 ':'로 구분된 노드 이름을 집어넣고, CLSH_HOSTFILE에는 노드 이름이 저장되어 있는 파일의 경로를 집어넣는다. 경로는 기본은 상대 경로로 처리하고 '/'로 경로 값을 시작하면 절대 경로로 판단한다. 둘 다 값이 존재하지 않으면 에러를 출력하고 종료하게 된다.

```
root@DESKTOP-9S2ICBK:/CLSH# export CLSH_HOSTS=work01:work02
root@DESKTOP-9S2ICBK:/CLSH# env | grep CLSH
PWD=/CLSH
CLSH_HOSTS=work01:work02
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py cat /proc/loadavg
No PIPE. Passing...
CLSH_HOSTS env exist.
Note : use CLSH_HOSTS env
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work02
clsh local >
```

첫 번째로 CLSH_HOSTS의 값만 설정할 경우이다.

```
root@DESKTOP-9S2ICBK:/CLSH# unset CLSH_HOSTS
root@DESKTOP-9S2ICBK:/CLSH# export CLSH_HOSTFILE=./hostfile
root@DESKTOP-9S2ICBK:/CLSH# env | grep CLSH
PWD=/CLSH
CLSH_HOSTFILE=./hostfile
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py cat /proc/loadavg
No PIPE. Passing...
CLSH_HOSTS env doesn't exist. Finding CLSH_HOSTFILE env...
Note : use hostfile ./hostfile(CLSH_HOSTFILE env)
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work02, work03, work04
```

두 번째로 CLSH_HOSTFILE의 값만 설정할 경우이다.

```
root@DESKTOP-9S2ICBK:/CLSH# env | grep CLSH
PWD=/CLSH
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py cat /proc/loadavg
No PIPE. Passing...
CLSH_HOSTS and CLSH_HOSTFILE env don't exist. Finding hostfile...
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work02, work03, work04
clsh local >
```

마지막으로 위 두 환경 변수의 값이 지정되지 않았으면 현재 디렉토리에서 'hostfile'을 탐색하여 값을 가져온다.

```
root@DESKTOP-9S2ICBK:/CLSH# env | grep CLSH
PWD=/CLSH
CLSH_HOSTS=work01:work02
CLSH_HOSTFILE=./hostfile
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py cat /proc/loadavg
No PIPE. Passing...
CLSH_HOSTS env exist.
Note : use CLSH_HOSTS env
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work02
clsh local >
```

만약 위 옵션 3개가 동시에 존재하면 CLSH_HOST 값, CLSH_HOSTFILE 값, 현재 디렉토리의 hostfile 값, 이 순서대로 탐색하여 존재하는 값을 사용한다.

iii. 옵션 2 (쉘 리다이렉션)

옵션 2는 유일하게 명령어의 시작이 다르다.

명령어의 시작을 현재 로컬의 프롬프트에서 동작할 명령어를 입력 후 '|' 문자를 넣고 'python3 clsh.py'를 쓰고 후에 원하는 옵션을 설정하면 된다. 이 옵션의 동작은 로컬에서 동작하는 명령어의 실행 결과가 마지막에 입력하는 커맨드의 입력 값으로 들어간다.

```
root@DESKTOP-9S2ICBK:/CLSH# ls /etc/*.conf | python3 clsh.py --hostfile ./hostfile cat
work02: /etc/adduser.conf
/etc/ca-certificates.conf
/etc/debconf.conf
/etc/deluser.conf
/etc/e2scrub.conf
/etc/fuse.conf
/etc/gai.conf
/etc/hdparm.conf
/etc/host.conf
/etc/ld.so.conf
```

지금 예시는 로컬의 프롬프트에 'ls /etc/*.conf' 명령어를 실행하고 그 명령어의 출력물을 'cat' 명령어의 인자로 전달한 것이다.

-I 옵션만을 제외하고 모든 옵션들을 사용 가능하다.

iv. 옵션 3 (출력 옵션)

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py --host work01,work02 --out=/tmp/run/ --err=/tmp cat /proc/loadavg
No PIPE. Passing...
work02: 0.13 0.09 0.02 3/842 532

work01: 0.13 0.09 0.02 2/842 902
```

입력할 명령어 : '—out='원하는 경로', '-err=원하는 경로'

out이나 err옵션에 값에 원하는 경로를 넣으면 경로가 없을 시에는 경로를 생

성하고 그 밑에 원격 실행한 명령의 출력 결과는 '노드 이름.out', 에러 출력 결과는 '노드 이름.err'로 저장 된다.

v. 옵션4 (Interactive mode)

이 옵션은 -I 옵션으로만 구분이 되어 -I 옵션을 입력한 것과 하지 않은 것으로 구분된다. 다른 옵션을 입력하면 동작하지 않는다.

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py -i
No PIPE. Passing...
CLSH_HOSTS env exist.
Note : use CLSH_HOSTS env
Enter 'quit' to leave this interactive mode
Working with nodes: work01, work02
If you want to communicate with a single container, please enter its name.
If you leave it blank, communication will be established with all containers.
Node Input : work12
Input node just one exactly
Node Input : work01
clsh > ls -al
work01: total 32
drwx------ 1 root root 4096 Dec  7 19:10 .
drwxr-xr-x 1 root root 4096 Dec  7 05:29 ..
-rw------- 1 root root   95 Dec  7 18:30 .bash_history
-rw-r--r-- 1 root root 3106 Oct 15  2021 .bashrc
-rw-r--r-- 1 root root  161 Jul  9  2019 .profile
drwx------ 1 root root 4096 Dec  7 17:57 .ssh
-rw------- 1 root root 1769 Dec  7 19:10 .viminfo
```

1. -I 옵션을 입력하였을 경우

   -I 옵션을 입력하였을 경우 '옵션 2'에서 구현한 'hostfile'옵션 생략이 자동으로 동작한다. 동작은 하지만 아직 연결은 되지 않았다. 즉, 환경 변수의 값이나 파일의 값으로 연결될 노드들을 보여준다.

   'working with nodes :' 이 보여지는 라인에 현재 연결 가능한 노드들의 이름을 보여준다. 연결될 단 하나의 노드 이름을 입력하고 원하는 명령어를 입력하면 그 노드와만 통신을 한다. 노드 이름을 잘못 입력하면 올바른 노드 이름을 입력할 때 까지 반복된다.

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py -i
No PIPE. Passing...
CLSH_HOSTS env exist.
Note : use CLSH_HOSTS env
Enter ‘quit’ to leave this interactive mode
Working with nodes: work01, work02
If you want to communicate with a single container, please enter its name.
If you leave it blank, communication will be established with all containers.
Node Input :
clsh > ls -a
work02: .
..
.bashrc
.profile
.ssh

work01: .
..
.bash_history
.bashrc
.profile
.ssh
.viminfo

clsh >
```

노드를 입력할 때 아무것도 입력하지 않고 제출을 하면 연결 가능한 모든 노드들과 통신을 하게 된다.

위의 통신들은 계속 이어지며 ‘quit’을 입력하면 프로그램을 종료한다.

```
work01: .
..
.bash_history
.bashrc
.profile
.ssh
.viminfo

clsh > quit
root@DESKTOP-9S2ICBK:/CLSH#
```

2. -I 옵션을 입력하지 않았을 경우

```
root@DESKTOP-9S2ICBK:/CLSH# python3 clsh.py
No PIPE. Passing...
CLSH_HOSTS env exist.
Note : use CLSH_HOSTS env
Enter ‘quit’ to leave this interactive mode
Working with nodes: work01, work02
clsh local > quit
root@DESKTOP-9S2ICBK:/CLSH#
```

I 옵션을 입력하지 않았다면 자동으로 모든 노드들과 통신을 하게 된다. 그러나 -I 옵션을 입력하였을 때와의 차이점은 입력을 로컬에서 받아 각 노드들의 입력으로 다시 전달이 된다. 위와 동일하게 ‘quit’을 입력하면 종료된다.