

Cloud Computing

Term project

소프트웨어학부

2018038030

이정민

Github Repository: <https://github.com/jeongmin99/CloudComputing>

AWS SDK를 위한 IAM 설정

AWS IAM 페이지에서 IAM 사용자 추가

[IAM](#) > [사용자](#) > jeongmin

jeongmin

정보

삭제

요약

ARN arn:aws:iam::952173121372:user/jeongmin	콘솔 액세스 비활성화된	액세스 키 1 액세스 키 만들기
생성됨 November 28, 2023, 15:14 (UTC+09:00)	마지막 콘솔 로그인 -	

권한

그룹

태그

보안 자격 증명

액세스 관리자

권한 정책 (0)

사용자에게 직접 연결된 정책을 통해 또는 그룹을 통해 권한을 정의합니다.

검색

필터링 기준 유형
모든 유형

< 1 > ⚙

정책 이름

유형

연결 방식:

사용 사례

☐ Command Line Interface(CLI)

AWS CLI를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☒ 로컬 코드

로컬 개발 환경의 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ AWS 컴퓨팅 서비스에서 실행되는 애플리케이션

Amazon EC2, Amazon ECS 또는 AWS Lambda와 같은 AWS 컴퓨팅 서비스에서 실행되는 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ 서드 파티 서비스

AWS 리소스를 모니터링 또는 관리하는 서드 파티 애플리케이션 또는 서비스에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ AWS 외부에서 실행되는 애플리케이션

이 액세스 키를 사용하여 AWS 리소스에 액세스해야 하는 AWS 외부의 데이터 센터 또는 기타 인프라에서 실행 중인 워크로드를 인증할 것입니다.

☐ 기타

귀하의 사용 사례가 여기에 나열되어 있지 않습니다.

⚠ 권장되는 대안

IAM 사용자 access key 및 secret access key 발급

액세스 키 검색 정보

액세스 키

분실하거나 잊어버린 비밀 액세스 키는 검색할 수 없습니다. 대신 새 액세스 키를 생성하고 이전 키를 비활성화합니다.

액세스 키	비밀 액세스 키
AKIA57MPO75QIAIIVBNWC	s34d4e5H716IN6pW4u4K1F3v414d18C708VX5i 숨기기

액세스 키 모범 사례

- 액세스 키를 일반 텍스트, 코드 리포지토리 또는 코드로 저장해서는 안 됩니다.
- 더 이상 필요 없는 경우 액세스 키를 비활성화하거나 삭제합니다.
- 최소 권한을 활성화합니다.
- 액세스 키를 정기적으로 교체합니다.

액세스 키 관리에 대한 자세한 내용은 [AWS 액세스 키 관리 모범 사례](#)를 참조하세요.

.csv 파일 다운로드

완료

사용자 그룹 생성 및 권한 정책(AdministratorAccess) 설정 후 사용자를 그룹에 추가

사용자 그룹 생성 ×

사용자 그룹을 생성하고 그룹에 연결할 정책을 선택합니다. 그룹을 사용하여 직무, AWS 서비스 액세스 또는 사용자 지정 권한별로 사용자 권한을 관리하는 것이 좋습니다. [자세히 알아보기](#)

사용자 그룹 이름
이 그룹을 식별하는 의미 있는 이름을 입력합니다.

master

최대 128자입니다. 영숫자 및 '+', '@', '_' 문자를 사용하세요.

권한 정책 (1/896)

정책 생성

필터링 기준 유형

모든 ... ▼

< 1 2 3 4 5 6 7 ... 45 >

	정책 이름	유형	다음... ▼	설명
<input checked="" type="checkbox"/>	AdministratorAccess	AWS 관리형 - ...	없음	Provides full access to AWS service...
<input type="checkbox"/>	AdministratorAcce...	AWS 관리형	없음	Grants account administrative per...
<input type="checkbox"/>	AdministratorAcce...	AWS 관리형	없음	Grants account administrative per...
<input type="checkbox"/>	AlexaForBusinessD...	AWS 관리형	없음	Provide device setup access to Alex...
<input type="checkbox"/>	AlexaForBusinessF...	AWS 관리형	없음	Grants full access to AlexaForBusin...
<input type="checkbox"/>	AlexaForBusinessG...	AWS 관리형	없음	Provide gateway execution access t...

취소

사용자 그룹 생성

Master node setting

인스턴스 생성 후 ssh로 접근

Htcondor GPG-KEY 및 레포지토리 다운로드

```
root@ip-172-31-36-244:~  
[root@ip-172-31-36-244 ~]# wget https://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor  
--2023-11-30 05:05:02-- https://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor  
Resolving research.cs.wisc.edu (research.cs.wisc.edu)... 128.105.7.58  
Connecting to research.cs.wisc.edu (research.cs.wisc.edu)|128.105.7.58|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 1752 (1.7K)  
Saving to: 'RPM-GPG-KEY-HTCondor'  
  
100%[=====>] 1,752      --.-K/s   in 0s  
  
2023-11-30 05:05:03 (217 MB/s) - 'RPM-GPG-KEY-HTCondor' saved [1752/1752]  
  
[root@ip-172-31-36-244 ~]# rpm --import RPM-GPG-KEY-HTCondor  
[root@ip-172-31-36-244 ~]#
```

```
root@ip-172-31-36-244:~  
[root@ip-172-31-36-244 ~]# wget https://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo  
--2023-11-30 05:05:51-- https://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo  
Resolving research.cs.wisc.edu (research.cs.wisc.edu)... 128.105.7.58  
Connecting to research.cs.wisc.edu (research.cs.wisc.edu)|128.105.7.58|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 254  
Saving to: 'htcondor-stable-rhel7.repo'  
  
100%[=====>] 254      --.-K/s   in 0s  
  
2023-11-30 05:05:52 (33.9 MB/s) - 'htcondor-stable-rhel7.repo' saved [254/254]  
  
[root@ip-172-31-36-244 ~]# ls  
htcondor-stable-rhel7.repo  RPM-GPG-KEY-HTCondor  
[root@ip-172-31-36-244 ~]# cp htcondor-stable-rhel7.repo /etc/yum.repos.d/  
[root@ip-172-31-36-244 ~]#
```

HTCondor 패키지 설치

```
root@ip-172-31-36-244:~  
[root@ip-172-31-36-244 ~]# yum install condor  
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd  
htcondor-stable | 2.9 kB 00:00  
htcondor-stable/primary_db | 124 kB 00:01  
Resolving Dependencies  
--> Running transaction check  
--> Package condor.x86_64 0:8.8.17-1.el7 will be installed  
--> Processing Dependency: condor-procd = 8.8.17-1.el7 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: condor-external-libs(x86-64) = 8.8.17-1.el7 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: condor-classads = 8.8.17-1.el7 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: libcgrouper >= 0.37 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: policycoreutils-python for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: perl(Archive::Tar) for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: libltdl.so.7()(64bit) for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: libclassad.so.14()(64bit) for package: condor-8.8.17-1.el7.x86_64  
--> Running transaction check
```

Master 노드로 사용할 인스턴스 이므로 해당 인스턴스의 internal hostname과 사용 데몬 (MASTER, SCHEDD, STARTD, COLLECTOR, NEGOTIATOR) 명시

```
root@ip-172-31-36-244:/etc/condor/config.d  
[root@ip-172-31-36-244 ~]# hostname  
ip-172-31-36-244.ap-northeast-2.compute.internal  
[root@ip-172-31-36-244 ~]# cd /etc/condor/config.d/  
[root@ip-172-31-36-244 config.d]# ls  
[root@ip-172-31-36-244 config.d]# vi condor_config.local  
[root@ip-172-31-36-244 config.d]# ^C  
[root@ip-172-31-36-244 config.d]# vi condor_config.local  
[root@ip-172-31-36-244 config.d]# cat condor_config.local  
ALLOW_WRITE = *  
CONDOR_HOST = ip-172-31-36-244.ap-northeast-2.compute.internal  
DAEMON_LIST = MASTER, SCHEDD, STARTD, COLLECTOR, NEGOTIATOR  
[root@ip-172-31-36-244 config.d]#
```

Condor_status 명령을 통해 master 노드가 클러스터에 편입된 것을 확인(STARTD)

```
root@ip-172-31-36-244:/etc/condor/config.d
[root@ip-172-31-36-244 config.d]# condor_status
Name                                     OpSys      Arch      State      Ac
ip-172-31-36-244.ap-northeast-2.compute.internal LINUX      X86_64    Unclaimed  Be

Machines Owner Claimed Unclaimed Matched Preempting Drain
X86_64/LINUX      1      0      0          1      0          0      0
Total             1      0      0          1      0          0      0
[root@ip-172-31-36-244 config.d]#
```

Condor 데몬 enable & start

```
root@ip-172-31-36-244:/etc/condor/config.d
[root@ip-172-31-36-244 config.d]# systemctl enable condor
Created symlink from /etc/systemd/system/multi-user.target.wants/condor.service
to /usr/lib/systemd/system/condor.service.
[root@ip-172-31-36-244 config.d]# systemctl is-enabled condor
enabled
[root@ip-172-31-36-244 config.d]# systemctl status condor
● condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/usr/lib/systemd/system/condor.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Thu 2023-11-30 05:16:58 UTC; 51s ago
     Main PID: 3518 (condor_master)
    Status: "All daemons are responding"
     CGroup: /system.slice/condor.service
            └─3518 /usr/sbin/condor_master -f
               └─3561 condor_procd -A /var/run/condor/procd_pipe -L /var/log/cond...
                  └─3562 condor_shared_port -f
                     └─3563 condor_collector -f
                        └─3564 condor_schedd -f
                           └─3567 condor_startd -f
                              └─3570 condor_negotiator -f

Nov 30 05:16:58 ip-172-31-36-244.ap-northeast-2.compute.internal systemd[1]: ...
Nov 30 05:16:58 ip-172-31-36-244.ap-northeast-2.compute.internal htcondor[3523]:
...
```

```
root@ip-172-31-33-56:~  
login as: ec2-user  
Authenticating with public key "htcondor"  
  
#  
~\####_ Amazon Linux 2  
~~\_#####\  
~~\_###| AL2 End of Life is 2025-06-30.  
~~\_#/_____  
~~ V~' '->  
~~~~ /  
~~.-_- /  
~~/_/_/_ /  
_/_/'-' /  
A newer version of Amazon Linux is available!  
  
Amazon Linux 2023, GA and supported until 2028-03-15.  
https://aws.amazon.com/linux/amazon-linux-2023/  
  
[ec2-user@ip-172-31-33-56 ~]$ sudo su -  
[root@ip-172-31-33-56 ~]#
```

```
root@ip-172-31-33-56:~  
[root@ip-172-31-33-56 ~]# wget https://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor  
--2023-11-30 05:21:27-- https://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor  
Resolving research.cs.wisc.edu (research.cs.wisc.edu)... 128.105.7.58  
Connecting to research.cs.wisc.edu (research.cs.wisc.edu)|128.105.7.58|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 1752 (1.7K)  
Saving to: 'RPM-GPG-KEY-HTCondor'  
  
100%[=====>] 1,752 --.-K/s in 0s  
  
2023-11-30 05:21:28 (231 MB/s) - 'RPM-GPG-KEY-HTCondor' saved [1752/1752]  
  
[root@ip-172-31-33-56 ~]# rpm --import RPM-GPG-KEY-HTCondor  
[root@ip-172-31-33-56 ~]# wget https://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo  
--2023-11-30 07:06:36-- https://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo  
Resolving research.cs.wisc.edu (research.cs.wisc.edu)... 128.105.7.58  
Connecting to research.cs.wisc.edu (research.cs.wisc.edu)|128.105.7.58|:443... connected.  
HTTP request sent, awaiting response... 200 OK
```

Slave 노드 HTCondor 패키지 설치

```
root@ip-172-31-33-56:~  
[root@ip-172-31-33-56 ~]# cp htcondor-stable-rhel7.repo /etc/yum.repos.d/  
[root@ip-172-31-33-56 ~]# yum install condor  
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd  
amzn2-core | 3.6 kB 00:00  
htcondor-stable | 2.9 kB 00:00  
htcondor-stable/primary_db | 124 kB 00:01  
Resolving Dependencies  
--> Running transaction check  
--> Package condor.x86_64 0:8.8.17-1.el7 will be installed  
--> Processing Dependency: condor-procd = 8.8.17-1.el7 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: condor-external-libs(x86-64) = 8.8.17-1.el7 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: condor-classads = 8.8.17-1.el7 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: libcgrouper >= 0.37 for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: policycoreutils-python for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: perl(Archive::Tar) for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: libltdl.so.7()(64bit) for package: condor-8.8.17-1.el7.x86_64  
--> Processing Dependency: libclassad.so.14()(64bit) for package: condor-8.8.17-1.el7.x86_64
```

Condor 설정에 master 노드의 hostname 명시, 데몬은 MASTER와 STARTD 사용

Condor 데몬 enable & start

```
root@ip-172-31-33-56:/etc/condor/config.d  
[root@ip-172-31-33-56 config.d]# cat condor_config.local  
ALLOW_WRITE = *  
CONDOR_HOST = ip-172-31-36-244.ap-northeast-2.compute.internal  
DAEMON_LIST= MASTER, STARTD  
[root@ip-172-31-33-56 config.d]# systemctl enable condor  
Created symlink from /etc/systemd/system/multi-user.target.wants/condor.service to /usr/lib/systemd/system/condor.service.  
[root@ip-172-31-33-56 config.d]# systemctl is-enabled condor  
enabled  
[root@ip-172-31-33-56 config.d]# systemctl status condor  
● condor.service - Condor Distributed High-Throughput-Computing  
   Loaded: loaded (/usr/lib/systemd/system/condor.service; enabled; vendor preset: disabled)  
   Active: inactive (dead)  
[root@ip-172-31-33-56 config.d]# systemctl start condor  
[root@ip-172-31-33-56 config.d]# systemctl status condor  
● condor.service - Condor Distributed High-Throughput-Computing  
   Loaded: loaded (/usr/lib/systemd/system/condor.service; enabled; vendor preset: disabled)  
   Active: active (running) since Thu 2023-11-30 07:10:28 UTC; 970ms ago  
     Main PID: 32546 (condor_master)  
       Tasks: 2 (limit: 4194303)  
      Memory: 1.6M  
     CGroup: /system.slice/condor.service
```


보안 그룹 설정

Master 노드의 보안 그룹과 Slave 노드가 속하는 보안 그룹 간 모든 트래픽에 대해 통신이 가능하도록 설정

EC2 > 보안 그룹 > sg-08ffcd7b60082e7fe5 - launch-wizard-5 > 인바운드 규칙 편집

인바운드 규칙 편집 정보

인바운드 규칙은 인스턴스에 도달하도록 허용된 수신 트래픽을 제어합니다.

보안 그룹 규칙 ID	유형 정보	프로토콜 정보	포트 범위 정보	소스 정보	설정 - 선택 사항 정보
sg-0949b1a373c42b1a9	HTTP	TCP	80	사용자 지정	Q, 0.0.0.0/0 ✕ 삭제
sg-079da7575f17ee2f	HTTPS	TCP	443	사용자 지정	Q, 0.0.0.0/0 ✕ 삭제
sg-0522051b9ed11e6fb	SSH	TCP	22	사용자 지정	Q, 0.0.0.0/0 ✕ 삭제
-	모든 트래픽	전체	전체	사용자 지정	Q, sg-038024d654c7f9fa5 ✕ CIDR 블록 보안 그룹 launch-wizard-6 sg-038024d654c7f9fa5 인두사 적록 삭제

규칙 추가

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses.

취소 변경 사항 미리 보기 규칙 저장

EC2 > 보안 그룹 > sg-038024d654c7f9fa5 - launch-wizard-6 > 인바운드 규칙 편집

인바운드 규칙 편집 정보

인바운드 규칙은 인스턴스에 도달하도록 허용된 수신 트래픽을 제어합니다.

보안 그룹 규칙 ID	유형 정보	프로토콜 정보	포트 범위 정보	소스 정보	설정 - 선택 사항 정보
sg-0f84371ec37d91934	HTTP	TCP	80	사용자 지정	Q, 0.0.0.0/0 ✕ 삭제
sg-021e82ca57594b98	SSH	TCP	22	사용자 지정	Q, 0.0.0.0/0 ✕ 삭제
sg-07c84c2b08ada2b91	HTTPS	TCP	443	사용자 지정	Q, 0.0.0.0/0 ✕ 삭제
-	모든 트래픽	전체	전체	사용자 지정	Q, sg-038024d654c7f9fa5 ✕ CIDR 블록 보안 그룹 launch-wizard-6 sg-038024d654c7f9fa5 인두사 적록 삭제

규칙 추가

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses.

취소 변경 사항 미리 보기 규칙 저장

Condor_status 명령 시 slave 인스턴스가 클러스터에 편입된 것을 확인

```
[root@ip-172-31-33-56 ~]# condor_status
Name                                     OpSys      Arch      State      Ac
ip-172-31-33-56.ap-northeast-2.compute.internal  LINUX      X86_64    Unclaimed  Id
ip-172-31-36-244.ap-northeast-2.compute.internal  LINUX      X86_64    Unclaimed  Id

Machines  Owner  Claimed  Unclaimed  Matched  Preempting  Drain
X86_64/LINUX      2      0      0      2      0      0
Total            2      0      0      2      0      0
[root@ip-172-31-33-56 ~]#
```

Slave 인스턴스 기반 이미지를 생성

이미지(AMI라고도 함)는 ECU 전스원스를 시작할 때 설정되는 프로그램 실행을 정의합니다. 기본 전스원스의 구성에서 이미지를 정정할 수 있습니다.

인스턴스 ID

i-099d7ae91ac17927b (htcondor-slave)

이미지 이름

htcondor-slave-image

최대 127자. 생성 후에는 수정할 수 없습니다.

이미지 설명 - 선택 사항

이미지 설명

최대 255자

재부팅 안 함

☐ 활성화

인스턴스 볼륨

스토리지 유형

디바이스

스냅샷

크기

볼륨 유형

IOPS

처리량

종료 시 삭제

암호화됨

EBS

/dev/...

볼륨에서 새 스냅샷 생성

8

EBS 범용 SSD - gp2

100

☒ 활성화

☐ 암호화

볼륨 추가

이미지 생성 프로세스 중에 Amazon EC2는 위의 각 볼륨의 스냅샷을 생성합니다.

태그 - 선택 사항

태그는 사용자가 AWS 리소스에 할당하는 레이블입니다. 각 태그는 키와 값(선택 사항)으로 구성됩니다. 태그를 사용하여 리소스를 검색 및 필터링하거나 AWS 비용을 추적할 수 있습니다.

이미지와 스냅샷을 함께 태그 지정

이미지와 스냅샷에 동일한 태그를 지정합니다.

이미지와 스냅샷을 별도로 태그 지정

이미지와 스냅샷에 다른 태그를 지정합니다.

리소스에 연결된 태그가 없습니다.

Amazon Machine Images(AMI) (1) 정보

내 소유

AMI를 속성 또는 태그로 찾기

AMI로 인스턴스 시작

1

	Name	AMI 이름	AMI ID	원본	소유자	표시 여부	상태	생성 날짜
<input type="checkbox"/>	htcondor-slave-image	ami-0825246f21ab11a8c	952173121372/htcondor-slave-image	952173121372	프라이빗	<input checked="" type="checkbox"/>	사용 가능	2023/11/30 16:00

인스턴스에 IP, pem key 없이 명령을 내릴 수 있는 AWS SSM을 사용하기 위한 IAM 역할 생성


[illegible]

Master 인스턴스에 생성한 IAM 역할 부여

[EC2](#) > [인스턴스](#) > [i-0fa386b594da20771](#) > IAM 역할 수정

IAM 역할 수정 정보



IAM 역할을 인스턴스에 연결합니다.

인스턴스 ID
 **i-0fa386b594da20771** (htcondor-master)

IAM 역할

인스턴스에 연결할 IAM 역할을 선택하거나 역할이 생성되어 있지 않다면 새 역할을 생성합니다. 선택한 역할이 현재 인스턴스에 연결된 모든 역할을 대체합니다.

ec2-ssm-role ▼

 [새 IAM 역할 생성](#) 



[취소](#) [IAM 역할 업데이트](#)

IAM 역할에 SSM 권한 부여



ec2-ssm-role 정보 [삭제](#)

Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.

요약 [편집](#)


생성 날짜 November 30, 2023, 22:37 (UTC+09:00)	ARN  arn:aws:iam::952173121372:role/ec2-ssm-role	인스턴스 프로필 ARN  arn:aws:iam::952173121372:instance-profile/ec2-ssm-role
마지막 활동 -	최대 세션 지속 시간 1시간	




[권한](#) | [신뢰 관계](#) | [태그](#) | [엑세스 관리자](#) | [세션 취소](#)

권한 정책 (2) 정보  [시뮬레이션](#)  [제거](#) [권한 추가 ▼](#)

최대 10개의 관리형 정책을 연결할 수 있습니다.

필터링 기준 유형
모든 유형 ▼

< 1 > 

<input type="checkbox"/>	정책 이름 	▲	유형 ▼	연결된 엔터티 ▼
<input type="checkbox"/>	 AmazonEC2RoleforSSM		AWS 관리형	2
<input type="checkbox"/>	 AmazonSSMManagedInstanceCore		AWS 관리형	2

List instance

인스턴스 id, 이미지 id, type, state, monitoring state를 출력한다.

```
def listInstances():
    print("Listing instances...")
    done = False
    while done==False:
        list=resource.instances.all()
        for instance in list:
            print("[id] %s, [AMI] %s, [type] %s, [state] %10s, [monitoring state] %s" %
(instance.instance_id,instance.image_id,instance.instance_type,instance.state[
'Name'],instance.monitoring['State']))

        done=True
```

```
-----
Amazon AWS Control Panel using SDK
-----
1. list instance          2. available zones
3. start instance         4. available regions
5. stop instance          6. create instance
7. reboot instance        8. list images
9. condor_status          10. scaling
11. condor_q              99. exit
-----
Enter an integer: 1
Listing instances...
[id] i-03e639c4290101b11, [AMI] ami-0f52ba4acb7f8f76a, [type] t2.micro, [state] stopped, [monitoring state] disabled
[id] i-0fa386b594da20771, [AMI] ami-05e02e6210658716f, [type] t2.micro, [state] running, [monitoring state] disabled
[id] i-099d7ae91ac17927b, [AMI] ami-05e02e6210658716f, [type] t2.micro, [state] stopped, [monitoring state] disabled
```

Available zones

가용한 zone의 정보를 출력한다.

```
def availableZones():
    print("Available zones...")
    done = False
    while done==False:
        res=ec2.describe_availability_zones()
        list=res['AvailabilityZones']
        for zone in list:
            print("[id] %s [region] %15s [zone] %15s" % (zone['ZoneId'],
zone['RegionName'], zone['ZoneName']))

        done=True
```

Amazon AWS Control Panel using SDK

- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |
-

Enter an integer: 2

Available zones...

[id]	apne2-az1	[region]	ap-northeast-2	[zone]	ap-northeast-2a
[id]	apne2-az2	[region]	ap-northeast-2	[zone]	ap-northeast-2b
[id]	apne2-az3	[region]	ap-northeast-2	[zone]	ap-northeast-2c
[id]	apne2-az4	[region]	ap-northeast-2	[zone]	ap-northeast-2d

Start instance

인스턴스의 id를 입력 받고, id에 해당하는 인스턴스를 시작한다.

```
def startInstance(id):
    print("Starting .... %s" % id)
    done= False
    while done==False:
        res=ec2.start_instances(InstanceIds=[id])
        instance=resource.Instance(id)
        #인스턴스 시작 시 까지 대기
        instance.wait_until_running(Filters=[{'Name': 'instance-
id', 'Values': [id]}])
        print("Successfully started instance %s" % id)
        done=True
```

Amazon AWS Control Panel using SDK

- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |
-

Enter an integer: 3

Enter instance id: i-099d7ae91ac17927b

Starting i-099d7ae91ac17927b

Successfully started instance i-099d7ae91ac17927b

Available regions

가용 리전의 정보를 출력한다.

```
def availableRegions():
    print("Available regions...")
    done = False
    while done==False:
        res=ec2.describe_regions()
        list=res['Regions']
        for region in list:
            print("[region] %15s, [endpoint] %s" %
                (region['RegionName'],region['Endpoint']))

        done=True
```

Amazon AWS Control Panel using SDK

- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |
-

Enter an integer: 4

Available regions...

[region]	ap-south-1,	[endpoint]	ec2.ap-south-1.amazonaws.com
[region]	eu-north-1,	[endpoint]	ec2.eu-north-1.amazonaws.com
[region]	eu-west-3,	[endpoint]	ec2.eu-west-3.amazonaws.com
[region]	eu-west-2,	[endpoint]	ec2.eu-west-2.amazonaws.com
[region]	eu-west-1,	[endpoint]	ec2.eu-west-1.amazonaws.com
[region]	ap-northeast-3,	[endpoint]	ec2.ap-northeast-3.amazonaws.com
[region]	ap-northeast-2,	[endpoint]	ec2.ap-northeast-2.amazonaws.com
[region]	ap-northeast-1,	[endpoint]	ec2.ap-northeast-1.amazonaws.com
[region]	ca-central-1,	[endpoint]	ec2.ca-central-1.amazonaws.com
[region]	sa-east-1,	[endpoint]	ec2.sa-east-1.amazonaws.com
[region]	ap-southeast-1,	[endpoint]	ec2.ap-southeast-1.amazonaws.com
[region]	ap-southeast-2,	[endpoint]	ec2.ap-southeast-2.amazonaws.com
[region]	eu-central-1,	[endpoint]	ec2.eu-central-1.amazonaws.com
[region]	us-east-1,	[endpoint]	ec2.us-east-1.amazonaws.com
[region]	us-east-2,	[endpoint]	ec2.us-east-2.amazonaws.com
[region]	us-west-1,	[endpoint]	ec2.us-west-1.amazonaws.com
[region]	us-west-2,	[endpoint]	ec2.us-west-2.amazonaws.com

Stop instance

instance id에 해당하는 인스턴스를 정지한다.

```
def stopInstance(id):
    print("Stopping .... %s" % id)
    done= False
    while done==False:
        res=ec2.stop_instances(InstanceIds=[id])
        instance=resource.Instance(id)
        #인스턴스 종료 시 까지 대기
        instance.wait_until_stopped(Filters=[{'Name': 'instance-
id', 'Values': [id]}])
        print("Successfully stopped instance %s" % id)
        done=True
```

Amazon AWS Control Panel using SDK

-
- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |
-

```
Enter an integer: 5
Enter instance id: i-0d61aa9f733857194
Stopping .... i-0d61aa9f733857194
Successfully stopped instance i-0d61aa9f733857194
-----
```

Create instance

AMI id를 입력 받고, 해당 ami를 기반으로 인스턴스를 생성한다. 이 때 생성되는 인스턴스는 condor pool에 포함되어야 하므로, master와 통신이 가능하도록 설정해야 한다. 따라서 condor slave 노드 용 보안그룹으로 설정한다.

```
def createInstance(id):
    print("Creating...")
    done= False
    while done==False:
        #htcondor-slave-image 기반 생성, 보안그룹 설정
        res=resource.create_instances(ImageId=id, InstanceType='t2.micro', MaxCo
unt=1, MinCount=1, SecurityGroupIds=[ 'sg-038024dd64c7f9fa5' ])
        instance=resource.Instance(res[0].instance_id)
```



```

        #인스턴스 실행 시 까지 대기
        instance.wait_until_running(Filters=[{'Name': 'instance-
id', 'Values': [res[0].instance_id]}])
        print("Successfully started EC2 instance %s based on AMI %s" %
(res[0].instance_id, id))
        done=True

```

Amazon AWS Control Panel using SDK

- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |

Enter an integer: 6
Enter ami id: ami-0825246f21ab11a8c
Creating...
Successfully started EC2 instance i-0d61aa9f733857194 based on AMI ami-0825246f21ab11a8c

Reboot instance

실행 중인 인스턴스의 id를 입력 받고, 해당 id를 가진 인스턴스를 재시작 한다.

```

def rebootInstance(id):
    print("Rebooting .... %s" % id)
    done= False
    while done==False:
        res=ec2.reboot_instances(InstanceIds=[id])
        instance=resource.Instance(id)
        #인스턴스 실행 시 까지 대기
        instance.wait_until_running(Filters=[{'Name': 'instance-
id', 'Values': [id]}])
        print("Successfully rebooted instance %s" % id)
        done=True

```

Amazon AWS Control Panel using SDK

- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |

Enter an integer: 7
Enter instance id: i-0d61aa9f733857194
Rebooting i-0d61aa9f733857194
Successfully rebooted instance i-0d61aa9f733857194

List image

가상머신 이미지 리스트를 출력한다. 본 프로그램에서는 htcondor-slave-image라는 이름을 가진 이미지의 정보만 출력한다.

```
def listImages():
    print("Listing images ....")
    done = False
    while done==False:
        #이름이 htcondor-slave-image 인 이미지만 출력
        res=ec2.describe_images(Filters=[{'Name':'name','Values':['htcondor-
slave-image']}]])
        list=res['Images']
        for image in list:
            print("[ImageID] %s, [Name] %s, [Owner] %s" %
(image['ImageId'],image['Name'],image['OwnerId']))

        done=True
```

Amazon AWS Control Panel using SDK

-
- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |
-

Enter an integer: 8

Listing images

[ImageID] ami-0825246f21ab11a8c, [Name] htcondor-slave-image, [Owner] 952173121372

Condor_status

HTCondor 클러스터의 상태를 출력한다. 본 프로그램에서는 IP 명시와 pem key 없는 접근을 위해 ssh가 아닌 AWS SSM을 사용하여 master 노드에 명령을 수행한다. Condor_status 명령을 master 노드에 전달하여 그 결과를 터미널에 출력한다.

```
def condor_status():
    #ssm module 로 통신
    res=ssm.send_command(InstanceIds=['i-
0fa386b594da20771'],DocumentName='AWS-RunShellScript',Parameters={'commands':
['condor_status']})
    command_id = res['Command']['CommandId']

    #커맨드 실행 결과 받을때 까지 대기
    waiter = ssm.get_waiter("command_executed")
```

```

try:
    waiter.wait(
        CommandId=command_id,
        InstanceId='i-0fa386b594da20771',
    )
except WaiterError as ex:
    logging.error(ex)
    return

#커맨드 실행 결과 출력
print(ssm.get_command_invocation(CommandId=command_id, InstanceId='i-0fa386b594da20771')['StandardOutputContent'])

```

```

-----
1. list instance           2. available zones
3. start instance         4. available regions
5. stop instance          6. create instance
7. reboot instance        8. list images
9. condor_status          10. scaling
11. condor_q              99. exit
-----
Enter an integer: 9
Name                                     OpSys      Arch   State   Activity LoadAv Mem   ActvtyTime
ip-172-31-33-56.ap-northeast-2.compute.internal LINUX      X86_64 Unclaimed Idle     0.000 952 0+02:04:46
ip-172-31-34-36.ap-northeast-2.compute.internal LINUX      X86_64 Unclaimed Idle     0.000 952 0+00:04:47
ip-172-31-36-244.ap-northeast-2.compute.internal LINUX      X86_64 Unclaimed Idle     0.000 952 0+03:47:56

Machines Owner Claimed Unclaimed Matched Preempting Drain
X86_64/LINUX      3      0      0      3      0      0      0
Total             3      0      0      3      0      0      0
-----

```

Scaling

현재 HTCondor queue의 상태를 확인하여, 현재 클러스터에 존재하는 slot의 개수와 큐에 존재하는 작업의 개수를 비교하여, slot보다 작업이 많은 경우 scale out, 그 반대의 경우에는 scale in을 수행한다. 또한 이미 작업이 배치된 노드의 삭제를 방지하기 위해, 해당 노드들의 hostname을 가져와 인스턴스 마다 비교하여 scale in을 수행한다. AWS SSM이 명령 결과를 빠르게 가져오지 못해 slot, job 개수와 작업이 배치된 노드의 hostname을 가져오는 명령을 셸 스크립트를 통해 작성하고, 해당 셸 스크립트를 실행하는 명령을 SSM으로 수행한다. 또한 동일한 명령을 thread programming을 통해 비동기적으로 수행하는 autoscaling 기능도 구현하였다.

```

# get slot, job, hostname
#!/bin/bash
tmp=$(condor_status -l | grep '^Cpus')
uc=$(echo "$tmp" | wc -l)

```

```
jobs=$(condor_q | grep 'Total for all users' | awk -F "jobs" '{ print $0}' | awk -F ' ' '{print $5}')
ud=$(echo "$jobs")
```

```
sched=$(condor_status | grep 'internal' | grep 'Claimed' | awk -F " " '{print $1}')
echo "$uc""$ud""$sched"
```

```
def scaling():
    #ssm 을 통해 master 의 shell script 실행
    # queue 의 job 개수, slot 개수, job 이 배치된 instance private-ip
    가져오기
    res=ssm.send_command(InstanceIds=['i-
0fa386b594da20771'],DocumentName='AWS-RunShellScript',Parameters={'commands':
['. /home/ec2-user/autoscaling.sh']})
    command_id = res['Command']['CommandId']
    #커맨드 실행 대기
    waiter = ssm.get_waiter("command_executed")
    try:
        waiter.wait(
            CommandId=command_id,
            InstanceId='i-0fa386b594da20771',
        )
    except WaiterError as ex:
        logging.error(ex)
    return

    #결과 가져오기
    result=ssm.get_command_invocation(CommandId=command_id, InstanceId='i-
0fa386b594da20771')['StandardOutputContent']
    slot=result[0] #슬롯 갯수
    jobs=result[1] #큐 작업 갯수
    sched=result[2] # 작업이 배치된 노드 private ip
    sched=sched.split('\n') # private ip string list split

    if slot<jobs: #슬롯보다 큐 작업이 많으면
        print('scale out')
        createInstance('ami-0825246f21ab11a8c') #slave image 기반 인스턴스
        생성

    elif slot>jobs: #슬롯이 큐 작업보다 많으면
        print('scale in')

        #실행 중인 인스턴스 목록 가져오기
        res=list(resource.instances.filter(Filters=[{'Name':'instance-
state-name', 'Values':['running']}]))
        result=[]

        #작업 배치된 슬롯이 아니면 append
```

```

for i in res:
    for j in range(0, len(sched)-1):
        if i.private_ip_address != sched[j]:
            result.append(i)

#모든 노드가 작업 배치된 것이 아니면
if len(result) != 0:
    #리스트 맨 마지막 인스턴스
    id=result[-1].instance_id
    #master 노드가 아니면
    if id != 'i-0fa386b594da20771':
        stopInstance(id) #인스턴스 종료
else:
    pass

# 큐 작업 == 슬롯 갯수
else:
    print('stable')

```

Scale out

 Amazon AWS Control Panel using SDK

- | | |
|--------------------|----------------------|
| 1. list instance | 2. available zones |
| 3. start instance | 4. available regions |
| 5. stop instance | 6. create instance |
| 7. reboot instance | 8. list images |
| 9. condor_status | 10. scaling |
| 11. condor_q | 99. exit |
-

Enter an integer: 10

scale out

Creating...

Successfully started EC2 instance i-0bf8542ed1289001c based on AMI ami-0825246f21ab11a8c

Scale in

```
-----
                Amazon AWS Control Panel using SDK
-----
1. list instance           2. available zones
3. start instance         4. available regions
5. stop instance          6. create instance
7. reboot instance        8. list images
9. condor_status          10. scaling
11. condor_q              99. exit
-----

Enter an integer: 10
scale in
Stopping .... i-0bf8542ed1289001c
Successfully stopped instance i-0bf8542ed1289001c
```

Condor_q

HTCondor 클러스터 queue의 상태를 출력한다. Condor_status와 마찬가지로 AWS SSM을 통해 master 노드에 명령을 전달하고, 결과값을 출력한다.

```
def condor_q():
    #ssm module 로 통신
    res=ssm.send_command(InstanceIds=['i-
0fa386b594da20771'],DocumentName='AWS-RunShellScript',Parameters={'commands':
['condor_q']})
    command_id = res['Command']['CommandId']

    #커맨드 실행 결과 받을때 까지 대기
    waiter = ssm.get_waiter("command_executed")
    try:
        waiter.wait(
            CommandId=command_id,
            InstanceId='i-0fa386b594da20771',
        )
    except WaiterError as ex:
        logging.error(ex)
        return

    #커맨드 실행 결과 출력
    print(ssm.get_command_invocation(CommandId=command_id, InstanceId='i-
0fa386b594da20771')['StandardOutputContent'])
```

```
-----
Amazon AWS Control Panel using SDK
-----
1. list instance          2. available zones
3. start instance        4. available regions
5. stop instance         6. create instance
7. reboot instance       8. list images
9. condor_status         10. scaling
11. condor_q             99. exit
-----
```

Enter an integer: 11

```
-- Schedd: ip-172-31-36-244.ap-northeast-2.compute.internal : <172.31.36.244:5629> @ 12/04/23 02:01:58
OWNER BATCH_NAME      SUBMITTED  DONE   RUN    IDLE  HOLD   TOTAL JOB_IDS
```

```
Total for query: 0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
Total for all users: 0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
```