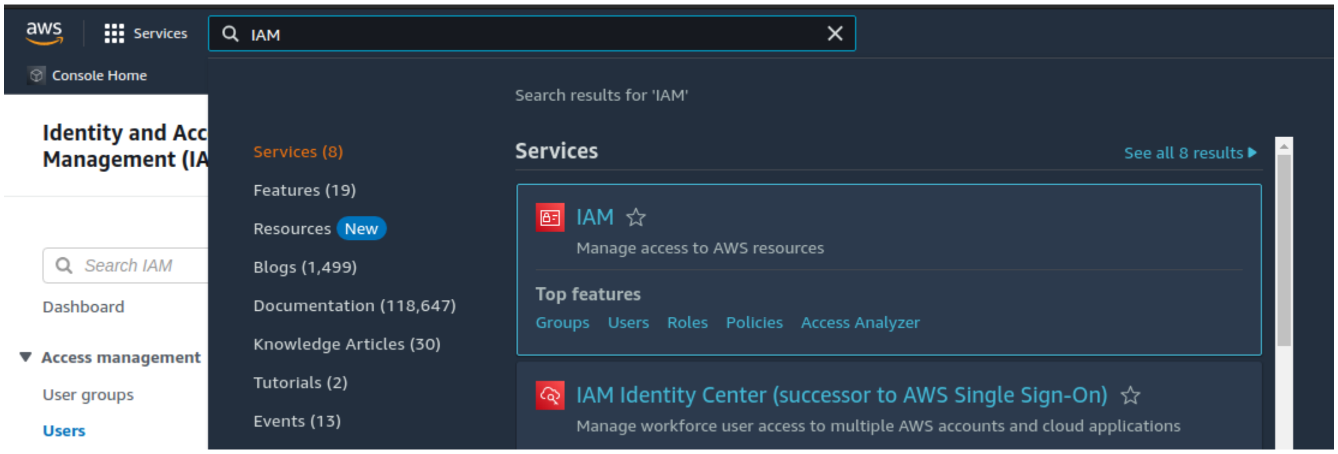
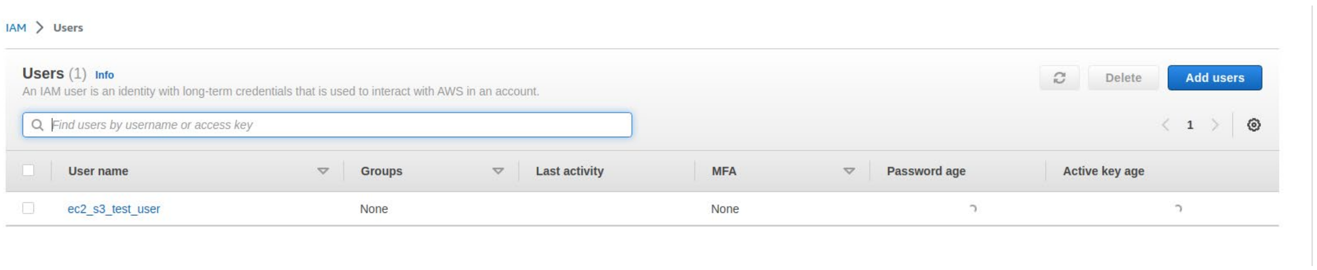
S3 as DFS (naive implementation)

AWS CONSOLE:

Create IAM user for authorizing use of AWS CLI on EC2-Instance



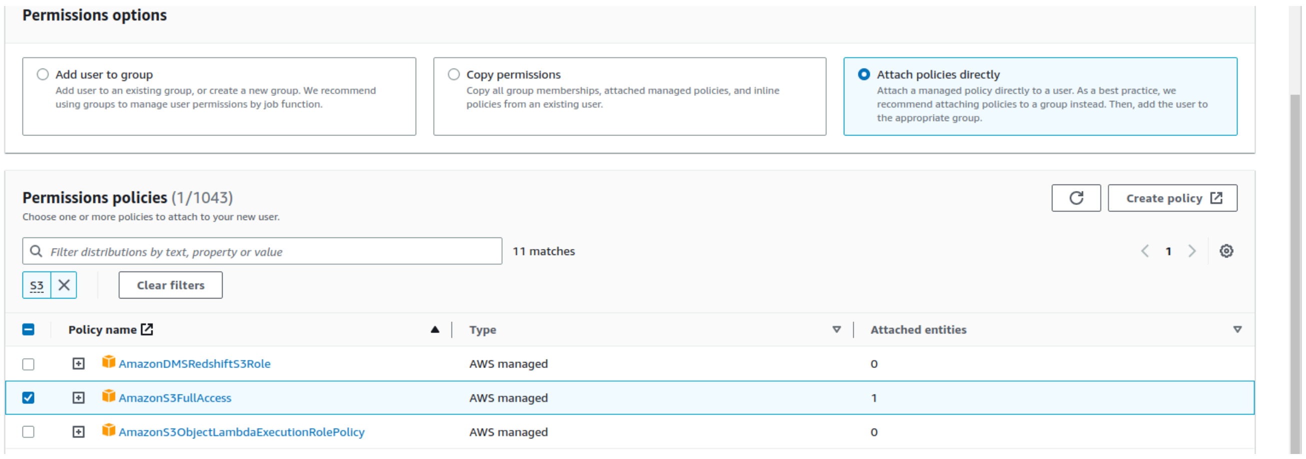
Select Add User



An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Groups	Last activity	MFA	Password age	Active key age
ec2_s3_test_user	None		None		

Attach a Policy with Appropriate Permissions (S3FullAccess/S3WriteOnly)


Permissions options
Permissions policies (1/1043)
Create policy
Filter distributions by text, property or value
11 matches
S3
Clear filters
1

Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

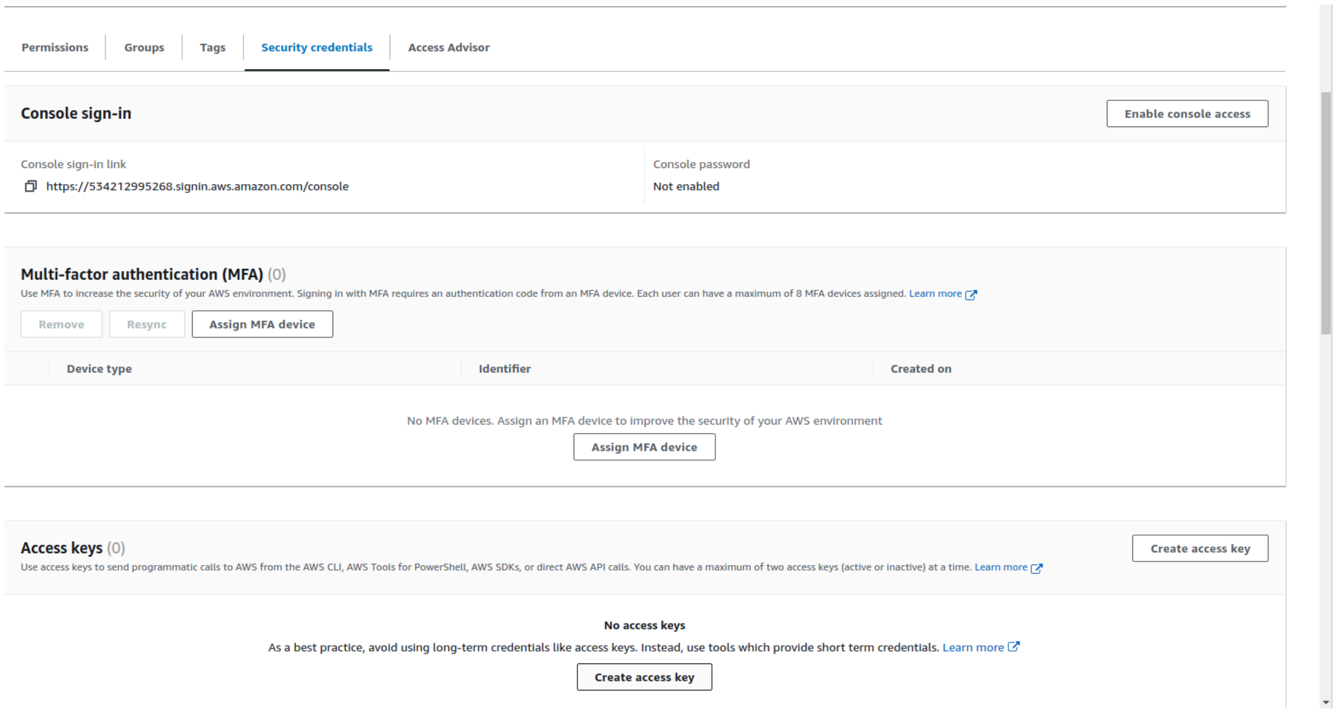
Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

[✓ Attach policies directly]
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Choose one or more policies to attach to your new user.

	Policy name	Type	Attached entities
☐	AmazonDMSRedshiftS3Role	AWS managed	0
✓	AmazonS3FullAccess	AWS managed	1
☐	AmazonS3ObjectLambdaExecutionRolePolicy	AWS managed	0

Create Access Key


Permissions | Groups | Tags | Security credentials | Access Advisor

Console sign-in

Enable console access

Console sign-in link
https://534212995268.signin.aws.amazon.com/console

Console password
Not enabled

Multi-factor authentication (MFA) (0)
Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. Learn more

Remove | Resync | Assign MFA device

Device type	Identifier	Created on

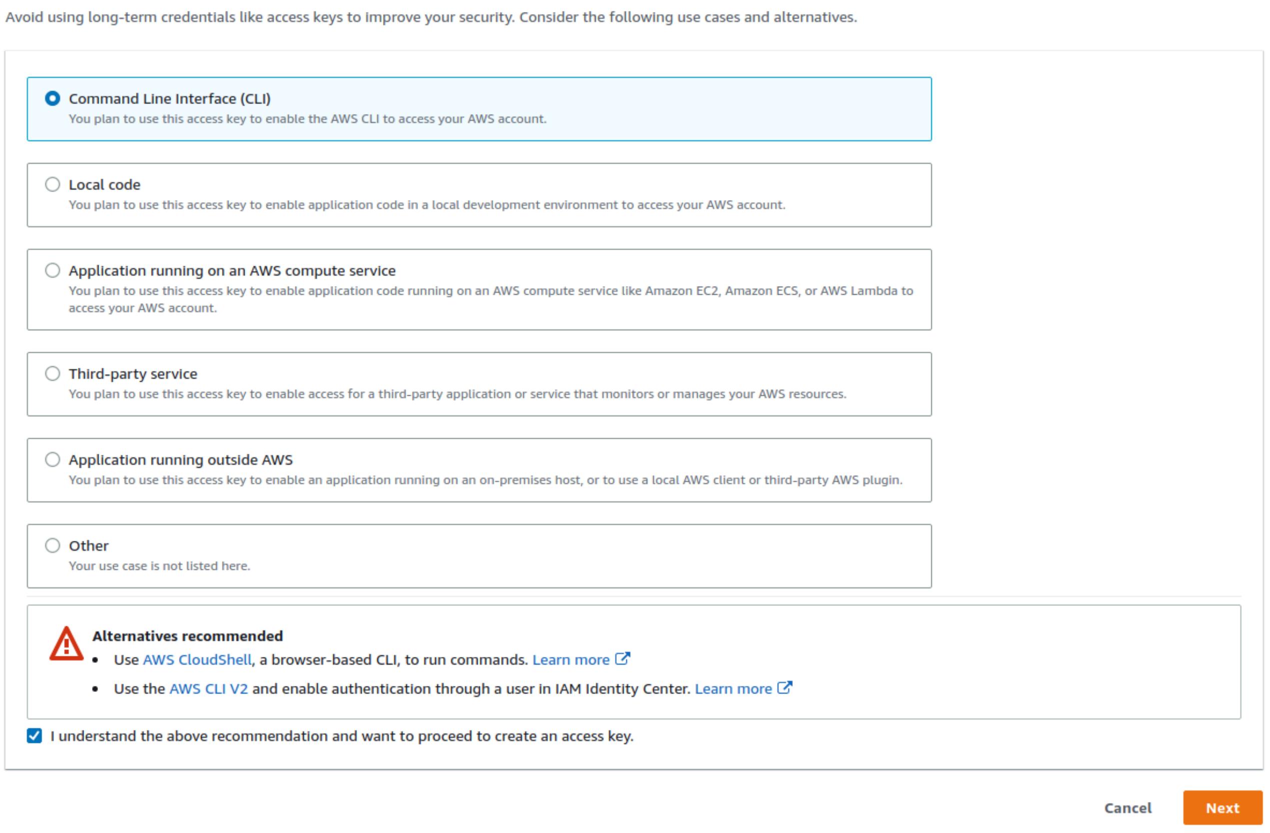
No MFA devices. Assign an MFA device to improve the security of your AWS environment
Assign MFA device

Access keys (0)
Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more

Create access key

No access keys
As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. Learn more
Create access key

Choose for CLI


Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

- [✓ Command Line Interface (CLI)]
  You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code
  You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service
  You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- Third-party service
  You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- Application running outside AWS
  You plan to use this access key to enable an application running on an on-premises host, or to use a local AWS client or third-party AWS plugin.
- Other
  Your use case is not listed here.

Alternatives recommended
- Use AWS CloudShell, a browser-based CLI, to run commands. Learn more
- Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. Learn more

[✓] I understand the above recommendation and want to proceed to create an access key.

Cancel | Next

Step Two:

i. Install AWS CLI on EC2-Target Machine

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

ii. Run `aws configure`

iii. Enter Access Key created during IAM setup along with Private Key

   a. Enter corresponding region name (likely us-east-1)

   b. Enter default values for other entries.

Step Three:

Add provided python code for s3Agent.py

Step Four:

Run agent in the background with:

```
nohup python3 s3Agent.py 1 & //for initial run

nohup python3 s3Agent.py 0 & //for any run after initial run
```

S3Agent.py

```
import sys
import os
import time
import hashlib

sys.stderr = open("/home/ec2-user/s3_stderr.txt", "w")
sys.stdout = open("/home/ec2-user/s3_stdout.txt", "w")
HOSTNAME = "TEST-PC-1-LINUX"
S3_ROOT_URI = "s3://ccdc-test-s3/"
disallowed = ["/aws/dist/awscli/examples"]
allowed = ["/home/", "/etc/", "/var/"]

def genCheckSum(fp):
    if not any(path in fp for path in allowed) or any(path in fp for path in disal-
lowed):
        return None
    hash = hashlib.md5()
    try:
        with open(fp, "rb") as f:
            for chunk in iter(lambda: f.read(4096), b""):
                hash.update(chunk)
        return hash.digest()
    except:
        return None

def retFileChanges(init_dict, cur_dict):
    nf = []
    rf = []
    for f in cur_dict:
        try:
            init_dict[f]
        except:
            nf.append(f)
    for f in init_dict:
        try:
            cur_dict[f]
        except:
           rf.append(f)
    return nf, rf

def list_files(startpath):
    sys_dict = {}
    for root, dirs, files in os.walk(startpath):
        level = root.replace(startpath, '').count(os.sep)
        for f in files:
           if (os.path.exists(os.path.join(root, f))):
               check_sum = genCheckSum(os.path.join(root, f))
               if check_sum is not None:
                   sys_dict[os.path.join(root, f)] = check_sum
    return sys_dict

def updateS3(nf, rf, mf):
    print("Adding new files:", nf)
    for f in nf:
        print("aws s3 cp {S3_ROOT_URI}{HOSTNAME}{f}")
        os.system("aws s3 cp {S3_ROOT_URI}{HOSTNAME}{f}")
    print("Removing old files:", rf)
    for f in rf:
        print("aws s3 rm {S3_ROOT_URI}{HOSTNAME}{f}")
        os.system("aws s3 rm {S3_ROOT_URI}{HOSTNAME}{f}")
    print("Updating modified files:", mf)
    for f in mf:
        print(f"aws s3 cp {f} {S3_ROOT_URI}{HOSTNAME}{f}")
        os.system(f"aws s3 cp {f} {S3_ROOT_URI}{HOSTNAME}{f}")

def performFullBackup(init_dict):
    i = 0
    for f in init_dict:
        with open("/home/ec2-user/out.txt", "a") as oLog:
            oLog.write(f"aws s3 cp {f} {S3_ROOT_URI}{HOSTNAME}{f}")
        print(f"aws s3 cp {f} {S3_ROOT_URI}{HOSTNAME}{f}")
        os.system(f"aws s3 cp {f} {S3_ROOT_URI}{HOSTNAME}{f}")

if __name__ == "__main__":

    init_dict = list_files('/')

    if int(sys.argv[1]) == 1 :
        print(len(init_dict))
        performFullBackup(init_dict)

    while(True):
        nf, rf, mf = [] ,[], []
        cur_dict = list_files('/')
        if len(cur_dict) != len(init_dict):
            nf, rf = retFileChanges(init_dict, cur_dict)
        for f in init_dict:
            if init_dict[f] is not None:
                try:
                    if init_dict[f] != cur_dict[f]:
                        mf.append(f)
                except:
                    continue
        init_dict =  cur_dict
        print(len(nf), len(rf), len(mf))
        if nf or rf or mf:
           updateS3(nf, rf, mf)
```