

# ACCESSING DYNAMOD OF AWS SERVICE USING THIRD-PARTY SERVICE

- First create an Amazon AWS root user: <https://shorturl.at/dIJOW>
- Then navigate to IAM from the search bar
- Inside IAM select the **users** from the left navbar
- Click Add New User and create an IAM user with two policies:
  - DynamoDB full access
  - Administrator access

The screenshot shows the AWS IAM 'Create new user' console. The user name is 'Python'. The 'Provide user access to the AWS Management Console' checkbox is checked. The 'User type' section has 'I want to create an IAM user' selected. The 'Console password' section has 'Custom password' selected, with a password field containing '\*\*\*\*\*'. The 'Users must create a new password at next sign-in' checkbox is also checked.

Python

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ \_ - (hyphen)

☒ Provide user access to the AWS Management Console - *optional*  
If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

**Are you providing console access to a person?**

User type

☐ Specify a user in Identity Center - Recommended  
We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

☒ I want to create an IAM user  
We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Console password

☐ Autogenerated password  
You can view the password after you create the user.

☒ Custom password  
Enter a custom password for the user.

\*\*\*\*\*

- Must be at least 8 characters long
- Must include at least three of the following mix of character types: uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and symbols ! @ # \$ % ^ & \* ( ) \_ + - (hyphen) = [ ] { } ' "

☐ Show password

☐ Users must create a new password at next sign-in - Recommended  
Users automatically get the IAMUserChangePassword policy to allow them to change their own password.

The screenshot shows the 'Permissions options' step of the 'Create new user' console. The 'Attach policies directly' option is selected. Below, a table lists available policies, with 'AdministratorAccess' selected.

**Permissions options**

☐ 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.

**Permissions policies (2/1136)**

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

Search

Filter by Type: All types

	Policy name	Type	Attached entities
<input type="checkbox"/>	AccessAnalyzerServiceRolePolicy	AWS managed	0
<input checked="" type="checkbox"/>	AdministratorAccess	AWS managed - job function	0
<input type="checkbox"/>	AdministratorAccess-Amplify	AWS managed	0
<input type="checkbox"/>	AdministratorAccess-AWSElasticBea...	AWS managed	0
<input type="checkbox"/>	AlexaForBusinessDeviceSetup	AWS managed	0
<input type="checkbox"/>	AlexaForBusinessFullAccess	AWS managed	0
<input type="checkbox"/>	AlexaForBusinessGatewayExecution	AWS managed	0

This screenshot shows the 'Review and create' step of the AWS IAM console. The user name is 'Python'. The console password type is 'Custom password', and 'Require password reset' is set to 'No'. The permissions summary shows two policies: 'AdministratorAccess' (AWS managed - job function) and 'AmazonDynamoDBFullAccess' (AWS managed), both used as 'Permissions policy'. There are no tags associated with the resource. The bottom of the page has 'Cancel', 'Previous', and 'Create user' buttons.

**User details**

User name	Console password type	Require password reset
Python	Custom password	No

**Permissions summary**

Name	Type	Used as
<a href="#">AdministratorAccess</a>	AWS managed - job function	Permissions policy
<a href="#">AmazonDynamoDBFullAccess</a>	AWS managed	Permissions policy

**Tags - optional**

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

[Cancel](#) [Previous](#) [Create user](#)

- Then return to the user's page and select the newly created user
- Inside that click create access key and click on “**Third Party access**”.

This screenshot shows the 'Create access key' page in the AWS IAM console. The breadcrumb trail is 'IAM > Users > Python > Create access key'. The page title is 'Access key best practices & alternatives'. The 'Use case' section has five options, with 'Third-party service' selected. The other options are 'Command Line Interface (CLI)', 'Local code', 'Application running on an AWS compute service', and 'Application running outside AWS'.

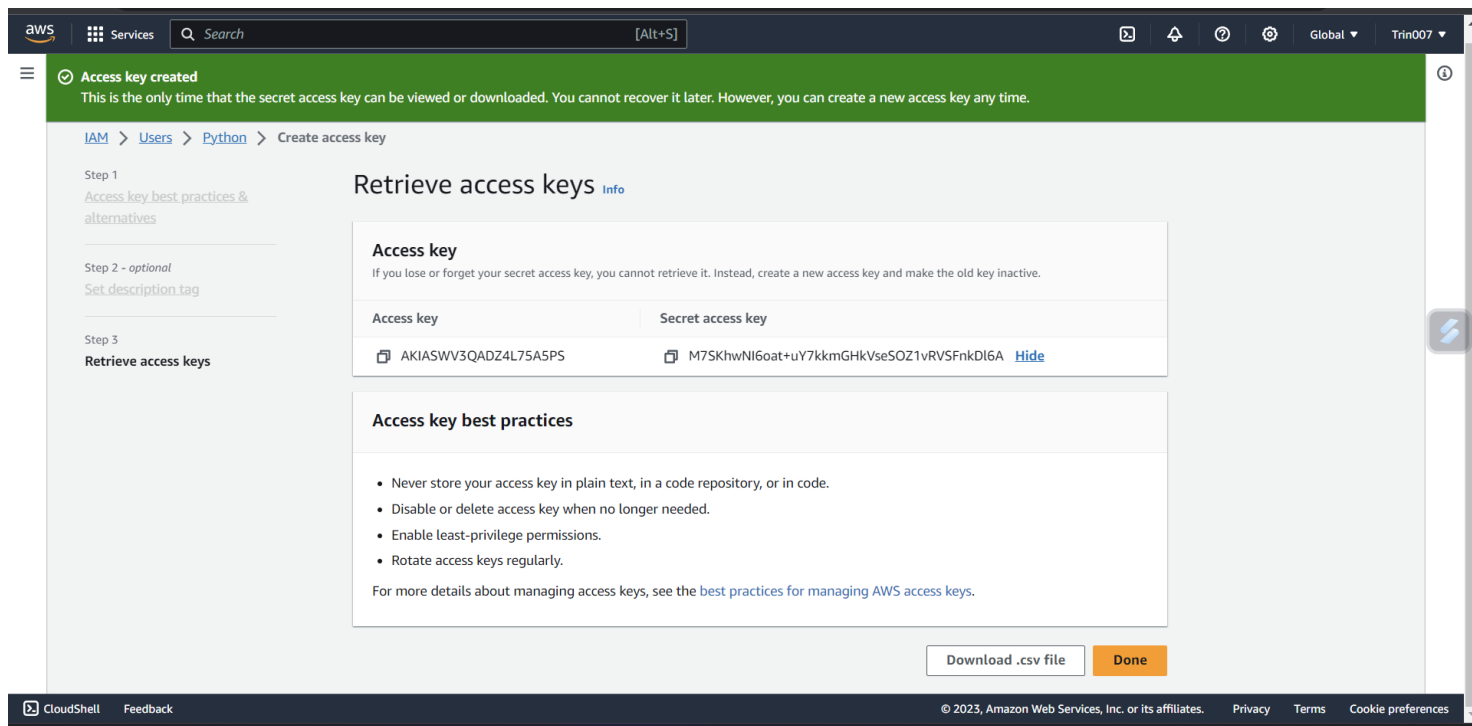
**Access key best practices & alternatives**

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

**Use case**

- ☐ 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 authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.

- Give a tag (optional) and create an access key. Note down the Access Key and Secret Access Key before returning to the user's page.



- Now go to search and type “**DynamoDB**” and click on that
- Then on the top right near your username set the region as Mumbai(ap-south-1)
- Now create a Python file and write the program with the access key, secret access key, and region-name.

## Python Program

```
import boto3
Access_Key="AKIASWV3QADZRAVW2XOK"
Secret_Access_Key="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"
client = boto3.client(
    'dynamodb',
    aws_access_key_id=Access_Key,
    aws_secret_access_key=Secret_Access_Key,
    region_name="ap-south-1",
)
dynamodb = boto3.resource(
    'dynamodb',
    aws_access_key_id=Access_Key,
    aws_secret_access_key=Secret_Access_Key,
    region_name="ap-south-1",
)
ddb_exceptions = client.exceptions
try:
    table = client.create_table(
```

```

TableName='ISS_locations',
KeySchema=[
    {
        'AttributeName': 'timestamp',
        'KeyType': 'HASH'
    }
],
AttributeDefinitions=[
    {
        'AttributeName': 'timestamp',
        'AttributeType': 'N'
    }
],
ProvisionedThroughput={
    'ReadCapacityUnits': 10,
    'WriteCapacityUnits': 10
}
)
print("Creating table")
waiter = client.get_waiter('table_exists')
waiter.wait(TableName='ISS_locations')
print("Table created")

except ddb_exceptions.ResourceInUseException:
    print("Table exists")

import requests
import time
def call_ISS_API():
    print("Calling ISS")
    time.sleep(1)
    r = requests.get('http://api.open-notify.org/iss-now.json')
    json = r.json()
    json['latitude'] = json['iss_position']['latitude']
    json['longitude'] = json['iss_position']['longitude']
    del json['iss_position']
    del json['message']
    return json

api_calls = {}
for i in range(5):
    response = call_ISS_API()
    api_calls[response['timestamp']] = response

print("Putting items")

```

```
for response in api_calls:
    dynamodb.Table('ISS_locations').put_item(
        Item=api_calls[response]
    )

print("Scanning table")
response = dynamodb.Table('ISS_locations').scan()
for i in response['Items']:
    print(i)
```

Replace the Access\_Key and Secret\_Access\_key with your own in the above program.  
This program will create a table called "ISS\_Location" and add 5 items in it.