AWS, PYTHON

WORKING WITH EC2 INSTANCES USING BOTO3 IN PYTHON

Kelvin Galabuzi

August 7, 2021 | No Comments

★★★★★ 5(1)

The Amazon EC2 is a cloud service within Amazon Web Services cloud platform that allows building and managing virtual machines to support various application workloads. With Amazon EC2, you can create, resize or decommission instances at any time depending on the business requirements. As a Cloud Automation Engineer, you'll be dealing with a lot of tasks around this topic. So, this article is providing

Table of contents

Prerequisites

- Ensure that you have an AWS user account and programmatic API access with permissions for managing Amazon EC2 service.
- To simplify your cloud automation jorney we suggest you to configure and use the AWS Cloud9 IDE. Alternatively, you can use your local Linux and Windows command-line interfaces. If that's the case, you need to set up your Python environment and the AWS access credentials for your shell environment.

This section of the article will cover the basic operations with EC2 instances using the Boto3 library.

Creating EC2 instance

To create one or more EC2 instances, you need to use the create_instances() method of the EC2 resource. The simplest EC2 instance configuration might include the following arguments:

- `MinCount` – minimum number of EC2 instances to launch
- `MaxCount` – maximum number of EC2 instances to launch
- `ImageId` – the Amazon Machine Image which is used to launch your EC2 instance (Working with Snapshots and AMIs using Boto3 in Python)
- `InstanceType` – Instance Type specifies how much CPU and RAM resources your EC2 instance should have
- `KeyName` – SSH key name, which you're going to use to get remote access to the EC2 instance

Such configuration will launch an EC2 instance in the default VPC:

Creating EC2 instance

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
KEY_PAIR_NAME = 'my-ssh-key-pair'
AMI_ID = 'ami-0cc00ed857256d2b4'

instances = EC2_RESOURCE.create_instances(
    MinCount = 1,
    MaxCount = 1,
    ImageId=AMI_ID,
    InstanceType='t2.micro',
    KeyName=KEY_PAIR_NAME,
    TagSpecifications=[
        {
            'ResourceType': 'instance',
```

```
                    'Value': 'my-ec2-instance'
                },
            ]
        },
    ]
)

for instance in instances:
    print(f'EC2 instance "{instance.id}" has been launched')

    instance.wait_until_running()
    print(f'EC2 instance "{instance.id}" has been started')
```

The `create_instances()` method returns a list of launched instances. You can use the fol-loop to walk through the instances and wait till every instance is up and running if you need to (the `wait_until_running()` method).

Here's an execution output:

Listing EC2 Instances

The best way to list all EC2 instances is to use the `all()` method from the instances collection of the EC2 resource.

Then you can use for-loop to iterate through the returned list of instances to get the information about Instance ID (`id`), Platform (`platform`), Instance Type (`instance_type`), Public IP (`public_ip_address`), Image (`image.id`) and many others by accessing instance object attributes.

Listing EC2 Instances

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)

instances = EC2_RESOURCE.instances.all()
```

```
    print(f'Instance state: {instance.state["Name"]}')
    print(f'Instance AMI: {instance.image.id}')
    print(f'Instance platform: {instance.platform}')
    print(f'Instance type: "{instance.instance_type}')
    print(f'Piblic IPv4 address: {instance.public_ip_address}')
    print('-'*60)
```

The screenshot below shows the resulting sample output from running the script.

Filtering EC2 instances

Filtering EC2 allows you to get a list of EC2 instances based on specified conditions. For example, you can get specific instance information by instance type, a list of instances based on Tags, instance state, and many other conditions.

Filtering EC2 instances by state

Filtering EC2 instances by state

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_STATE = 'running'

instances = EC2_RESOURCE.instances.filter(
    Filters=[
        {
            'Name': 'instance-state-name',
            'Values': [
                INSTANCE_STATE
            ]
        }
    ]
)

print(f'Instances in state "{INSTANCE_STATE}":')

for instance in instances:
    print(f'  - Instance ID: {instance.id}')
```

Here's an execution output:

Filtering EC2 instances by type

To filter EC2 instances by type, you can usethe `filter()` method in the instances collection of the EC2 resource:

Filtering EC2 instances by type

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_TYPE = 't2.micro'

instances = EC2_RESOURCE.instances.filter(
    Filters=[
        {
            'Name': 'instance-type',
            'Values': [
                INSTANCE_TYPE
```

```
)

print(f'Instances of type "{INSTANCE_TYPE}":')

for instance in instances:
    print(f'  - Instance ID: {instance.id}')
```

Here's an execution output:

Filtering EC2 instances by Tag

To filter EC2 instances by type, you can usethe `filter()` method in the instances collection of the EC2 resource:

Filtering EC2 instances by Tag

```
AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_NAME_TAG_VALUE = 'my-ec2-instance'

instances = EC2_RESOURCE.instances.filter(
    Filters=[
        {
            'Name': 'tag:Name',
            'Values': [
                INSTANCE_NAME_TAG_VALUE
            ]
        }
    ]
)

print(f'Instances with Tag "Name={INSTANCE_NAME_TAG_VALUE}":')

for instance in instances:
    print(f'  - Instance ID: {instance.id}')
```

Here's an execution output:

Filtering EC2 instances by instance ID

To filter EC2 instances by Instance ID, you can usethe `filter()` method in the instances collection of the EC2 resource:

Filtering EC2 instances by instance ID

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

instances = EC2_RESOURCE.instances.filter(
    InstanceIds=[
        INSTANCE_ID,
    ],
)
```

Here's an execution output:

Describing EC2 instance properties

To access theEC2 instance properties, you can use thedescribe_instances() method of the EC2 client (gets all properties in the Python dictionary format), or you can use the EC2.Instance class' attributes (provides access to a specific attribute) of the EC2 resource.

Describing all EC2 instance properties

The `describe_instances()` method accepts the Filters and InstanceIds attributes that allow you to find specific instances.

Describing all EC2 instance properties

```
from datetime import date, datetime
import boto3

AWS_REGION = "us-east-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'


# Helper method to serialize datetime fields
def json_datetime_serializer(obj):
    if isinstance(obj, (datetime, date)):
        return obj.isoformat()
    raise TypeError ("Type %s not serializable" % type(obj))


response = EC2_CLIENT.describe_instances(
    InstanceIds=[
        INSTANCE_ID,
    ],
)

print(f'Instance {INSTANCE_ID} attributes:')

for reservation in response['Reservations']:
    print(json.dumps(
            reservation,
            indent=4,
            default=json_datetime_serializer
        )
    )
```

Here's an execution output:

Listing EC2 instance EBS volumes

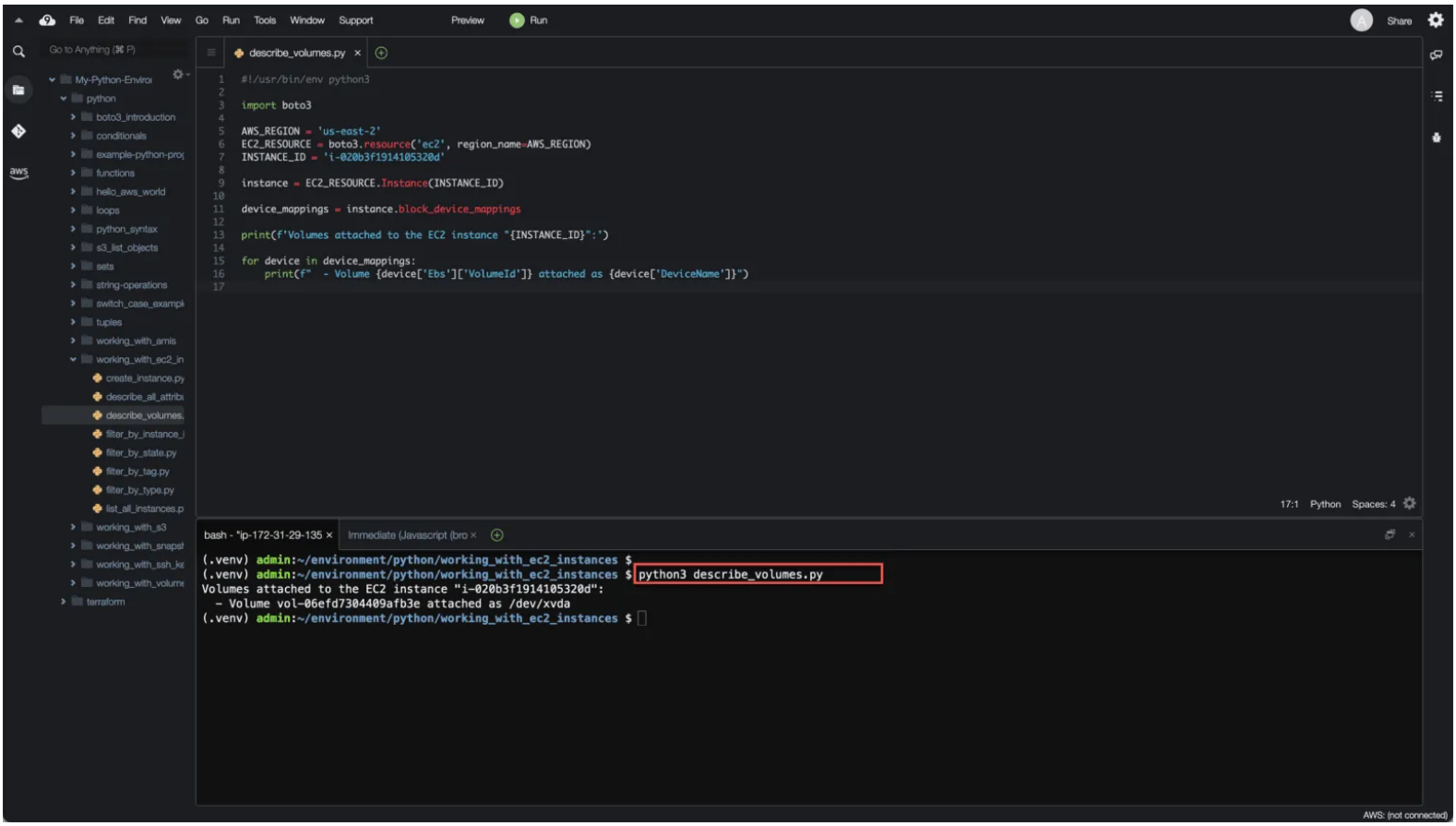
To get EC2 instance EBS volumes, you can use the block_device_mappings list of the EC2.Instance resource:

Listing EC2 instance EBS volumes

```
#!/usr/bin/env python3

import boto3

AWS_REGION = 'us-east-2'
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

instance = EC2_RESOURCE.Instance(INSTANCE_ID)

device_mappings = instance.block_device_mappings

print(f'Volumes attached to the EC2 instance "{INSTANCE_ID}":')
```

For additional information on working with EBS volumes, check out the Working with EBS volumes in Python using the Boto3 article.

Here's an execution output:



Getting EC2 instance state

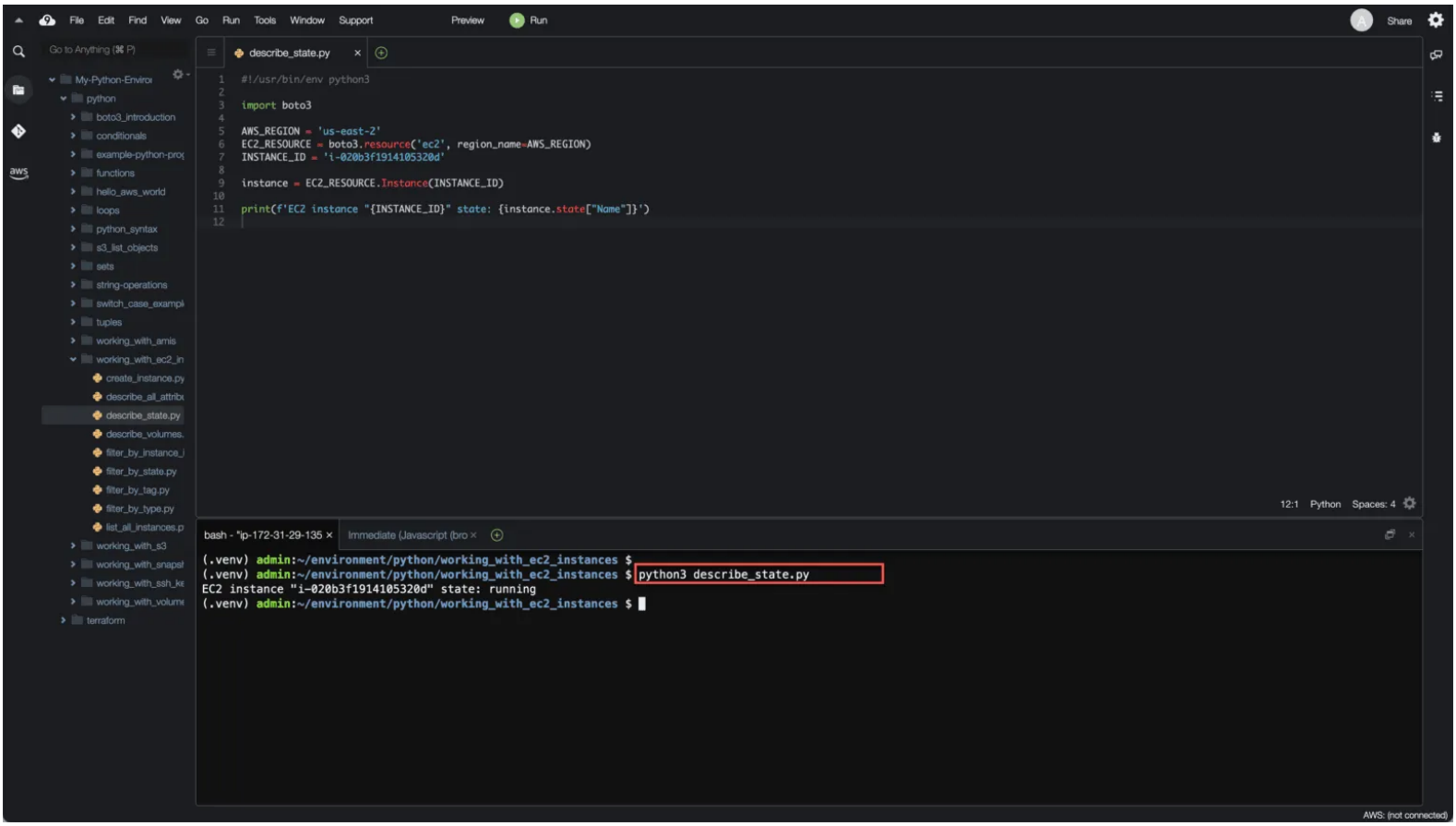
To get the EC2 instance state, you can use the state list of the EC2.Instance resource:

Getting EC2 instance state

```
#!/usr/bin/env python3

import boto3
```

```
instance = EC2_RESOURCE.Instance(INSTANCE_ID)

print(f'EC2 instance "{INSTANCE_ID}" state: {instance.state["Name"]}')
```

If you'd like to get a state of multiple EC2 instances, here's an example of using the filter() method.



Managing EC2 instance Tags

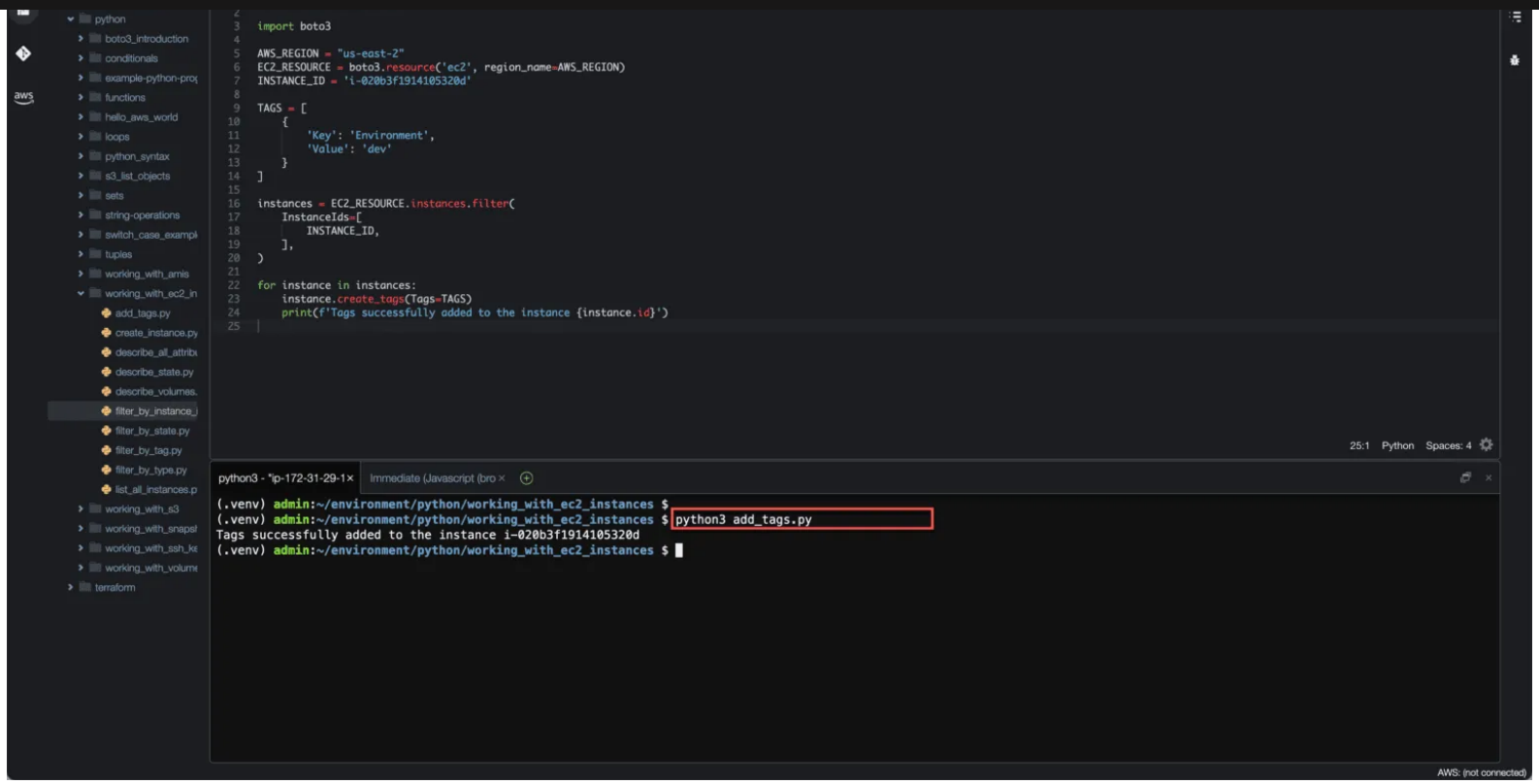
Tags allow you to group your resources according to your organization and project structure, and they usually help organize resources in groups for management or billing reporting purposes.

Adding Tags to EC2 instance

To add Tags to EC2 instances, you can use the create_tags() method of the EC2.Instance resource:

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

TAGS = [
    {
        'Key': 'Environment',
        'Value': 'dev'
    }
]

instances = EC2_RESOURCE.instances.filter(
    InstanceIds=[
        INSTANCE_ID,
    ],
)

for instance in instances:
    instance.create_tags(Tags=TAGS)
    print(f'Tags successfully added to the instance {instance.id}')
```

Here's an execution output:



Listing EC2 instance Tags

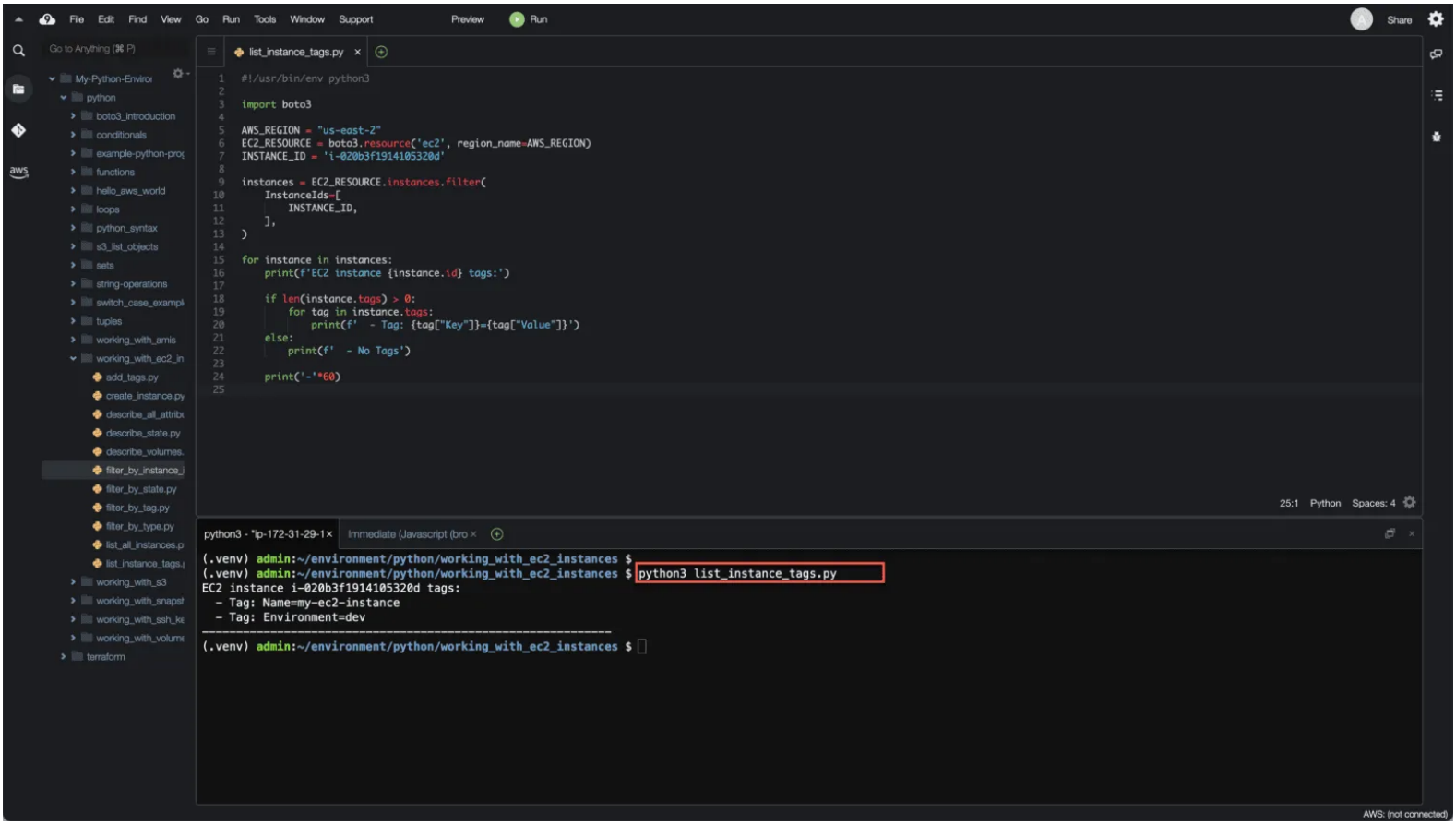
To list all tags that are associated with the EC2 instances, you can use a for loop to iterate through the list of instance.tags (EC2 resource):

Listing EC2 instance Tags

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

instances = EC2_RESOURCE.instances.filter(
    InstanceIds=[
        INSTANCE_ID,
    ],
)
```

```
    if len(instance.tags) > 0:
        for tag in instance.tags:
            print(f'  - Tag: {tag["Key"]}={tag["Value"]}')
    else:
        print(f'  - No Tags')

    print('-'*60)
```

Here's an execution output:



Updating EC2 instance Tags

To update Tags of EC2 instances, you can use the create_tags() method of the EC2.Instance resource that not only creates but also overrides Tags (previous example)

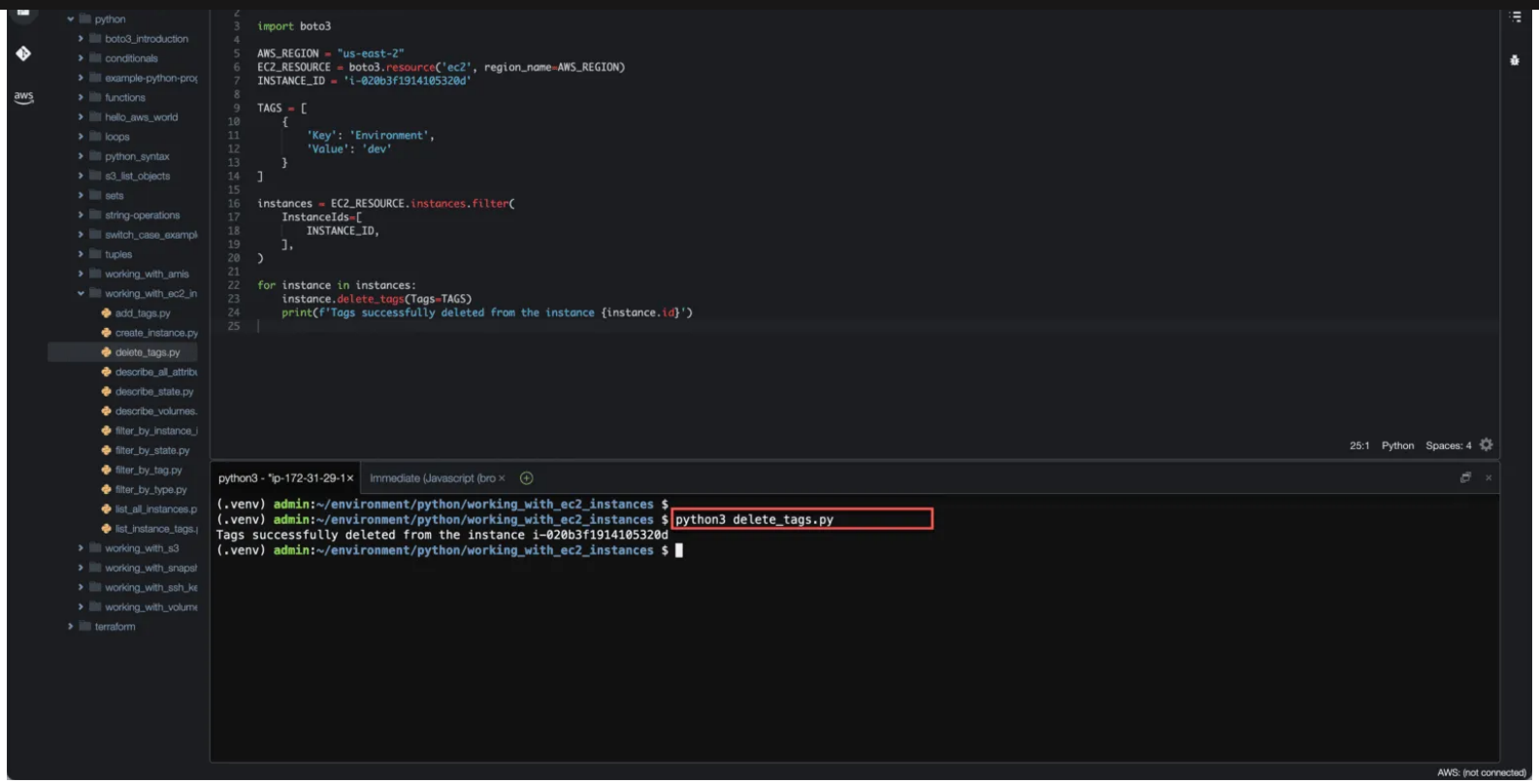
Deleting EC2 instance Tags

Note: you can delete the EC2 instance Tag based on its `Key` (required) and optional `Value`. If you specify a Tag `Key` without a Tag `Value`, the `delete_tags()` method will delete any Tag with the specified `Key` regardless of its value. If you specify a Tag `Key` with the `Value` that equals an empty string,the `delete_tags()` method will delete the Tag only if its value is an empty string.

Deleting EC2 instance Tags

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

TAGS = [
    {
        'Key': 'Environment',
        'Value': 'dev'
    }
]

instances = EC2_RESOURCE.instances.filter(
    InstanceIds=[
        INSTANCE_ID,
    ],
)

for instance in instances:
    instance.delete_tags(Tags=TAGS)
    print(f'Tags successfully deleted from the instance {instance.id}')
```

Here's an execution output:



EC2 instance advanced monitoring

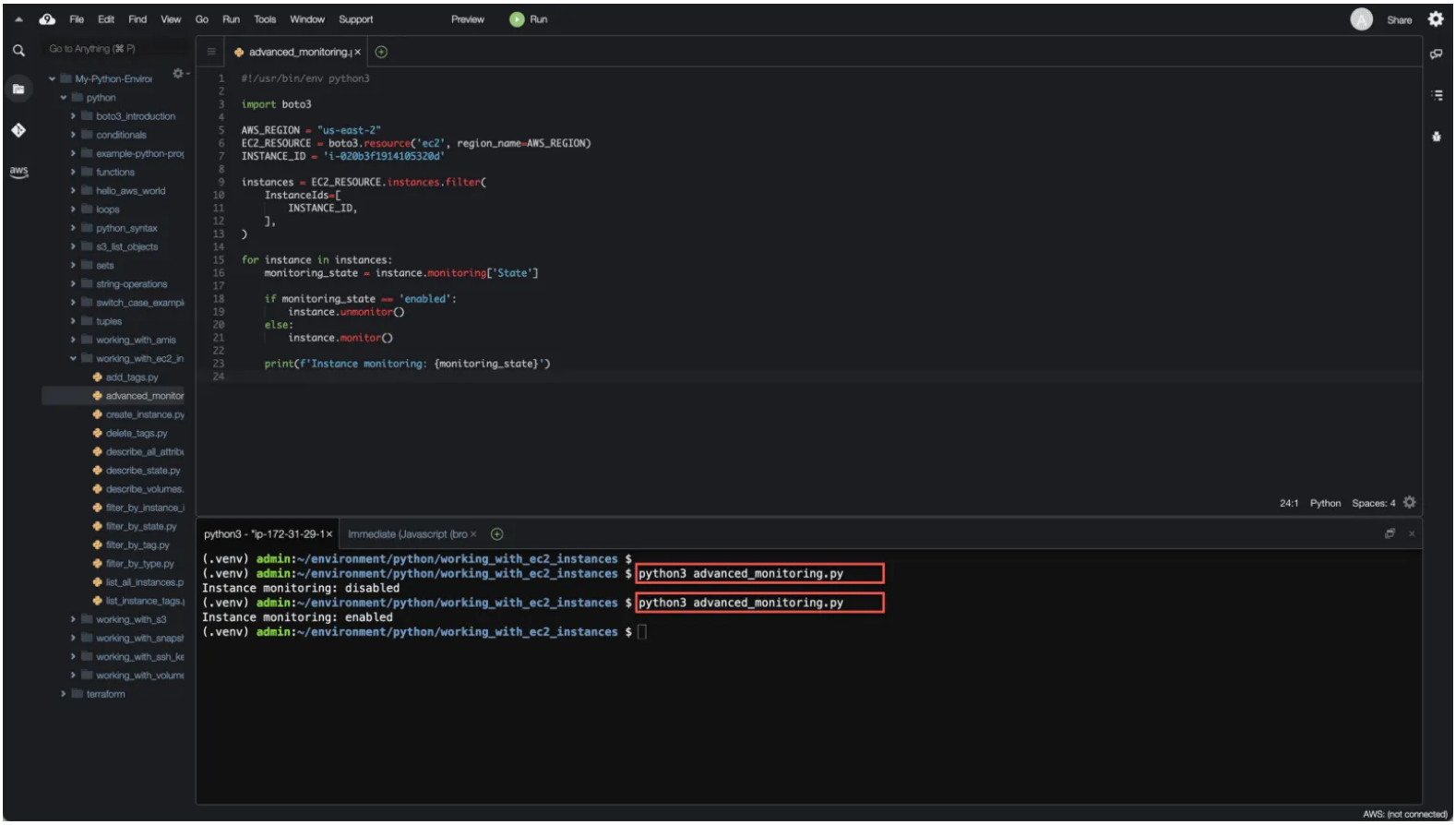
To enable advanced monitoring for the EC2 Instance, use the monitor() method to turn the monitoring on and the unmonitor() method to turn the monitoring off (EC2 resource):

Enable and disable advanced monitoring

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

instances = EC2_RESOURCE.instances.filter(
    InstanceIds=[
        INSTANCE_ID,
    ],
)
```

```
    if monitoring_state == 'enabled':
        instance.unmonitor()
    else:
        instance.monitor()

    print(f'Instance monitoring: {monitoring_state}')
```

Here's an execution output:



Managing EC2 instance state

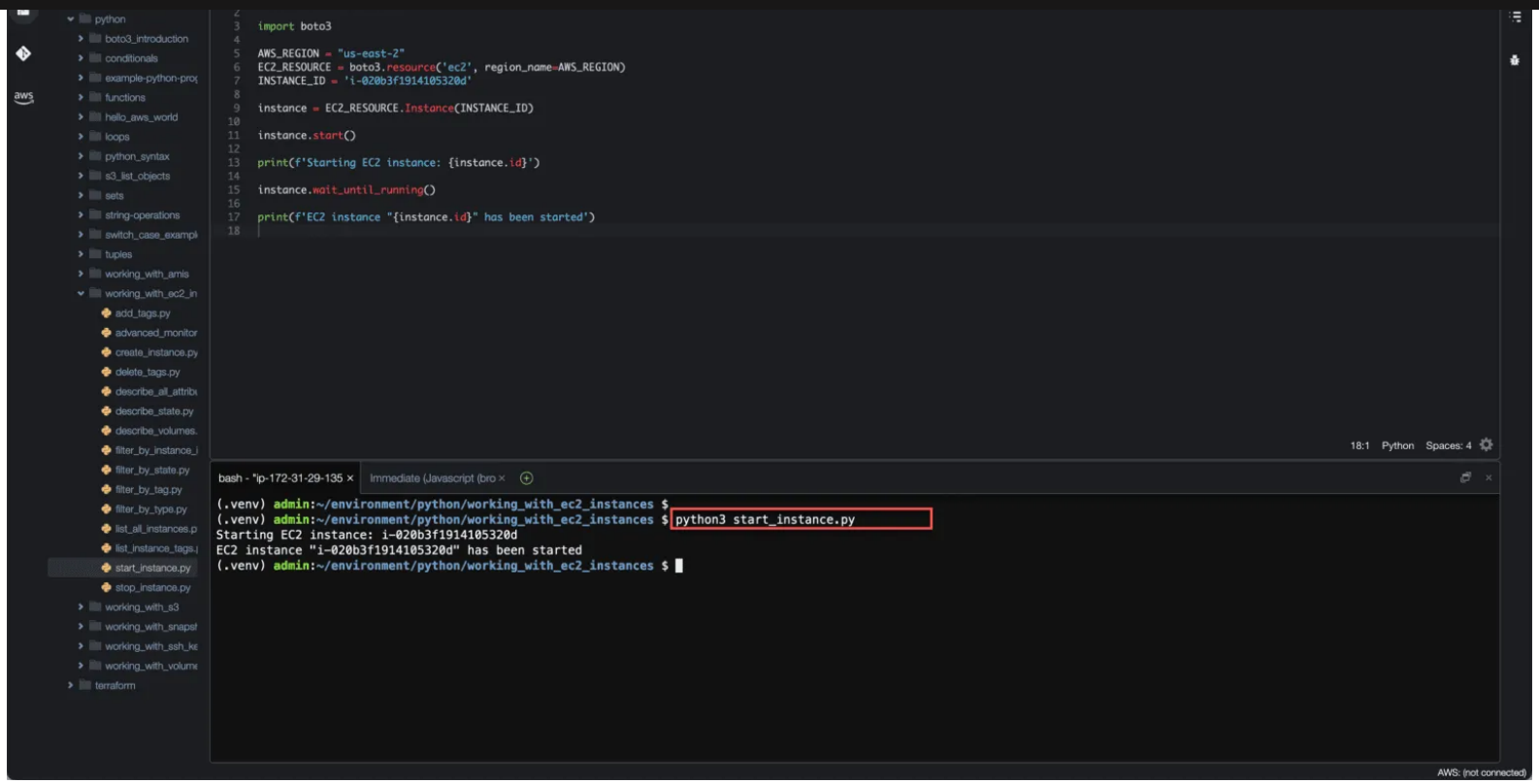
In this section of the article, we'll cover the basic method allowing to manage EC2 instance state: `start()`, `stop()`, `reboot()`, and `terminate()`.

Starting EC2 instance

Starting EC2 instance

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

instance = EC2_RESOURCE.Instance(INSTANCE_ID)

instance.start()

print(f'Starting EC2 instance: {instance.id}')

instance.wait_until_running()

print(f'EC2 instance "{instance.id}" has been started')
```

Thewait_until_running() waiter method allows you to wait till the EC2 instance is up and running.



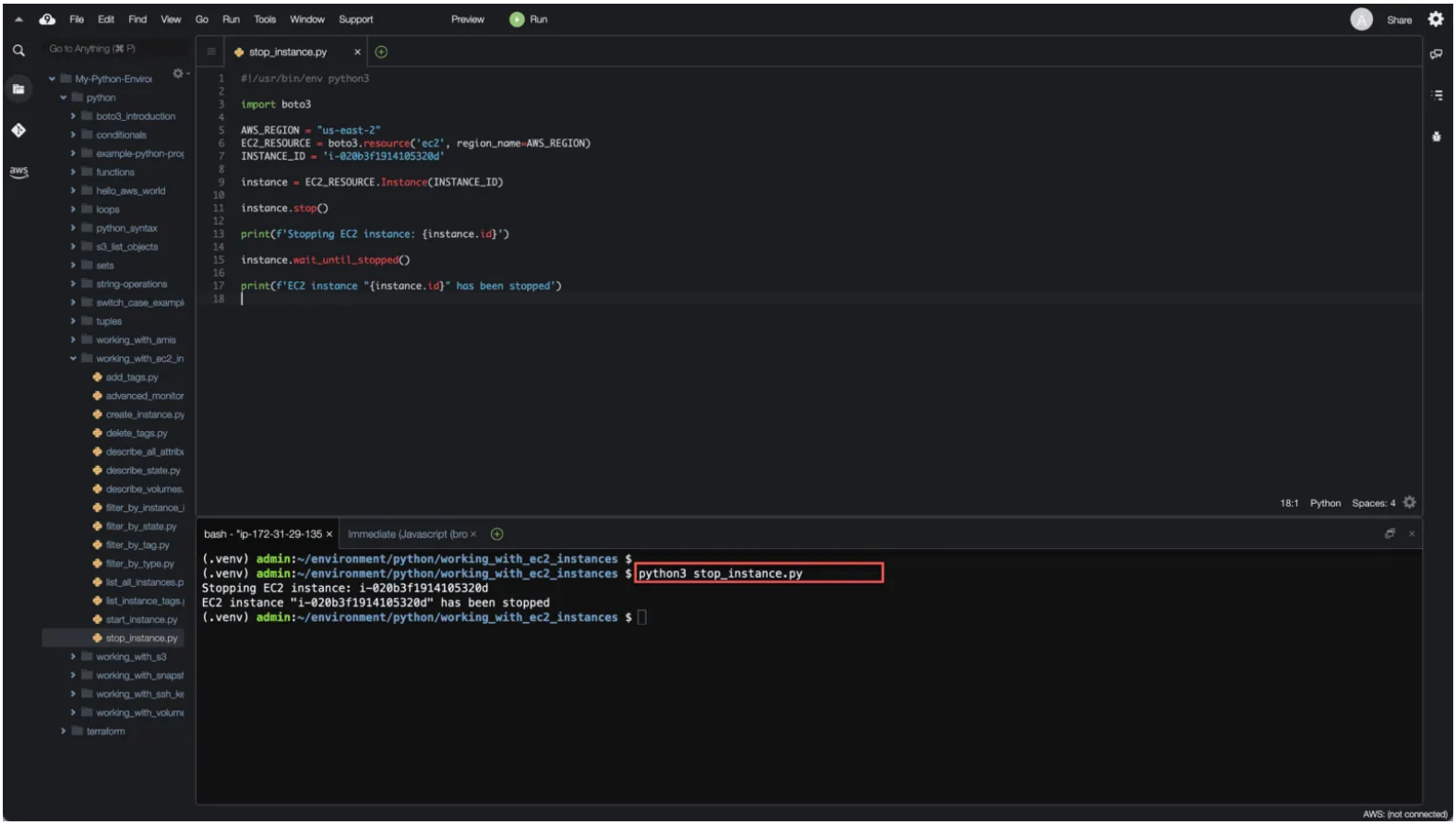
Stopping EC2 instance

To stop instances, you can use the stop() method of the EC2.Instance object:

Stopping EC2 instance

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'

instance = EC2_RESOURCE.Instance(INSTANCE_ID)

instance.stop()

print(f'Stopping EC2 instance: {instance.id}')

instance.wait_until_stopped()
```

Thewait_until_stopped() waiter method allows you to wait till the EC2 instance is completely stopped.



Rebooting EC2 instance

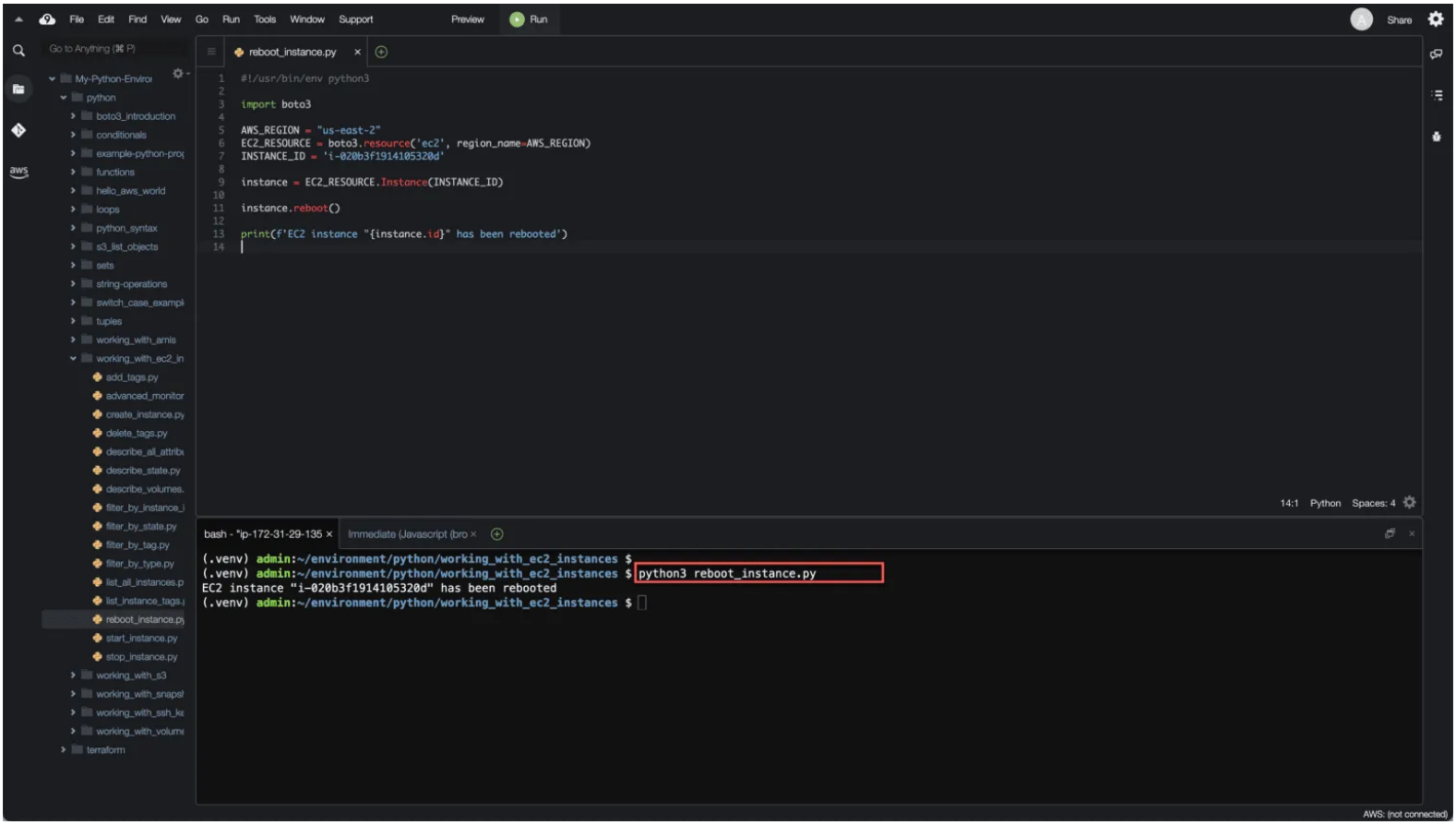
To stop instances, you can use the reboot() method of the EC2.Instance object:

Rebooting EC2 instance

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-020b3f1914105320d'
```

```
print(f'EC2 instance "{instance.id}" has been rebooted')
```

Here's an execution output:



Terminating EC2 instance

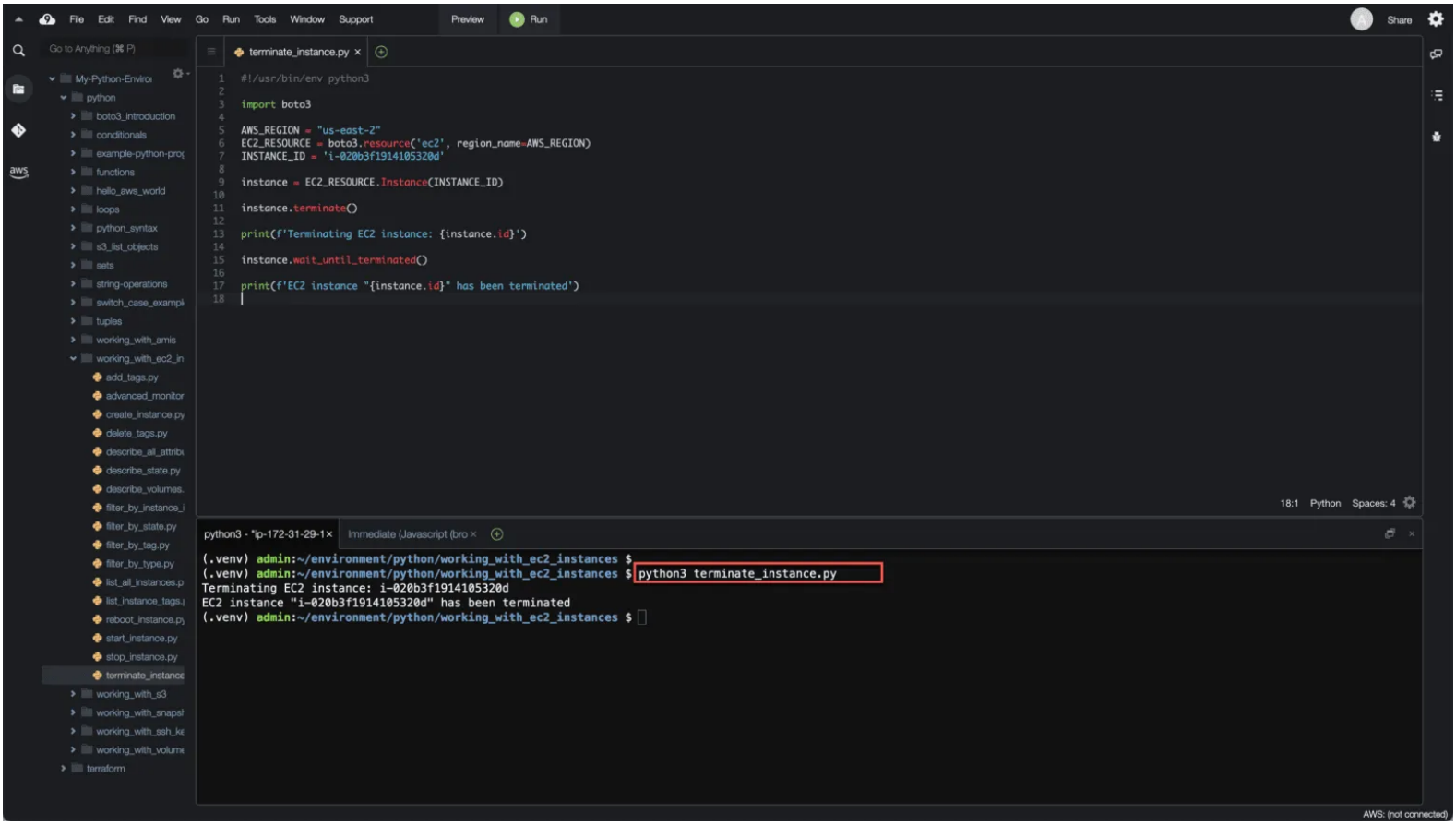
To terminate the EC2 instance, you can use the terminate() method of the EC2.Instance object:

Terminate EC2 instances using Boto3

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
```

```
instance = EC2_RESOURCE.Instance(INSTANCE_ID)

instance.terminate()

print(f'Terminating EC2 instance: {instance.id}')

instance.wait_until_terminated()

print(f'EC2 instance "{instance.id}" has been terminated')
```

Thewait_until_terminated() waiter method allows you to wait till the EC2 instance is completely terminated.



Modifying EC2 instance attributes using Boto3

To change EC2 instance attributes, you can use the `modify_attribute()` method of the EC2 resource.

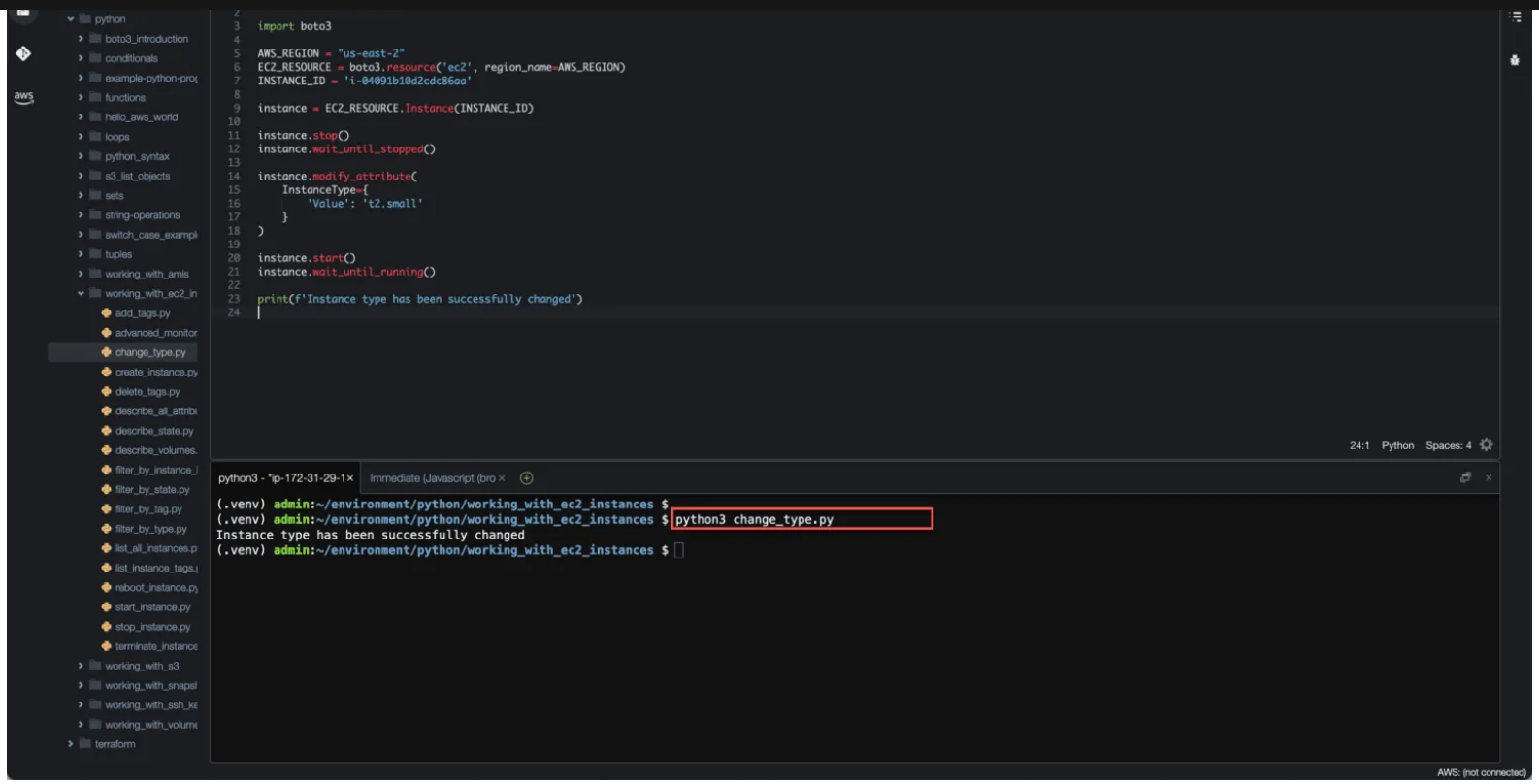
Changing EC2 instance type

To modify the EC2 instance type, you can use the modify_attribute() method of the EC2 client:

Changing EC2 instance type using Boto3

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-04091b10d2cdc86aa'

instance = EC2_RESOURCE.Instance(INSTANCE_ID)

instance.stop()
instance.wait_until_stopped()

instance.modify_attribute(
    InstanceType={
        'Value': 't2.small'
    }
)

instance.start()
instance.wait_until_running()

print(f'Instance type has been successfully changed')
```

Here's an execution output:



How to manage SSH keys using Boto3?

Before creating an EC2 instance using Boto3, you have to set up an SSH key in your account. You must have an SSH key during the EC2 instance launch if you're not using AWS Systems Manager and are willing to have remote access to your EC2 instance. In addition to that, you'll need an SSH key to get the Windows EC2 instance password.

You can create an SSH key manually using AWS Web Console or automatically by using the Boto3 library. This section of the article will describe how to use the Boto3 library to manage SSH keys.

For more information about SSH keys, we recommend you look at the Top 10 SSH Features You MUST Know To Be More Productive article.

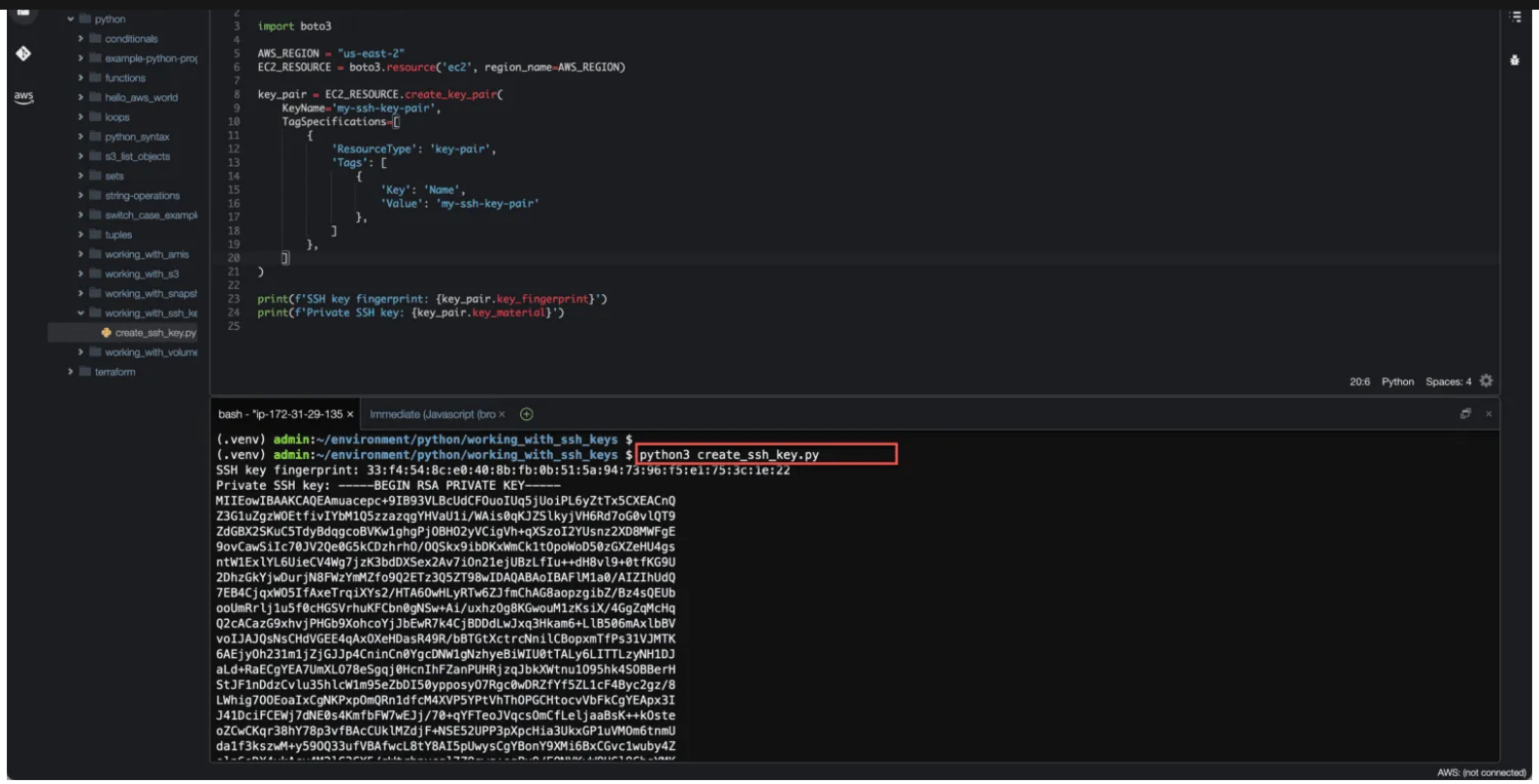
Creating SSH key

To create an SSH key pair, you have to use the create_key_pair() method of the EC2 resource. This method will generate a new SSH key pair and let you save the private SSH key.

Generating SSH Key pair using Boto3 client

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)

key_pair = EC2_RESOURCE.create_key_pair(
    KeyName='my-ssh-key-pair',
    TagSpecifications=[
        {
            'ResourceType': 'key-pair',
            'Tags': [
                {
                    'Key': 'Name',
                    'Value': 'my-ssh-key-pair'
                },
            ]
        },
    ]
)

print(f'SSH key fingerprint: {key_pair.key_fingerprint}')
print(f'Private SSH key: {key_pair.key_material}')
```

Here's an execution output:



Listing SSH keys

To list all SSH keys using Boto3, you need to use the `all()` method of the key_pairs collection of the EC2 resource and the for-loop. The most useful properties of the returned KeyPairInfo object are:

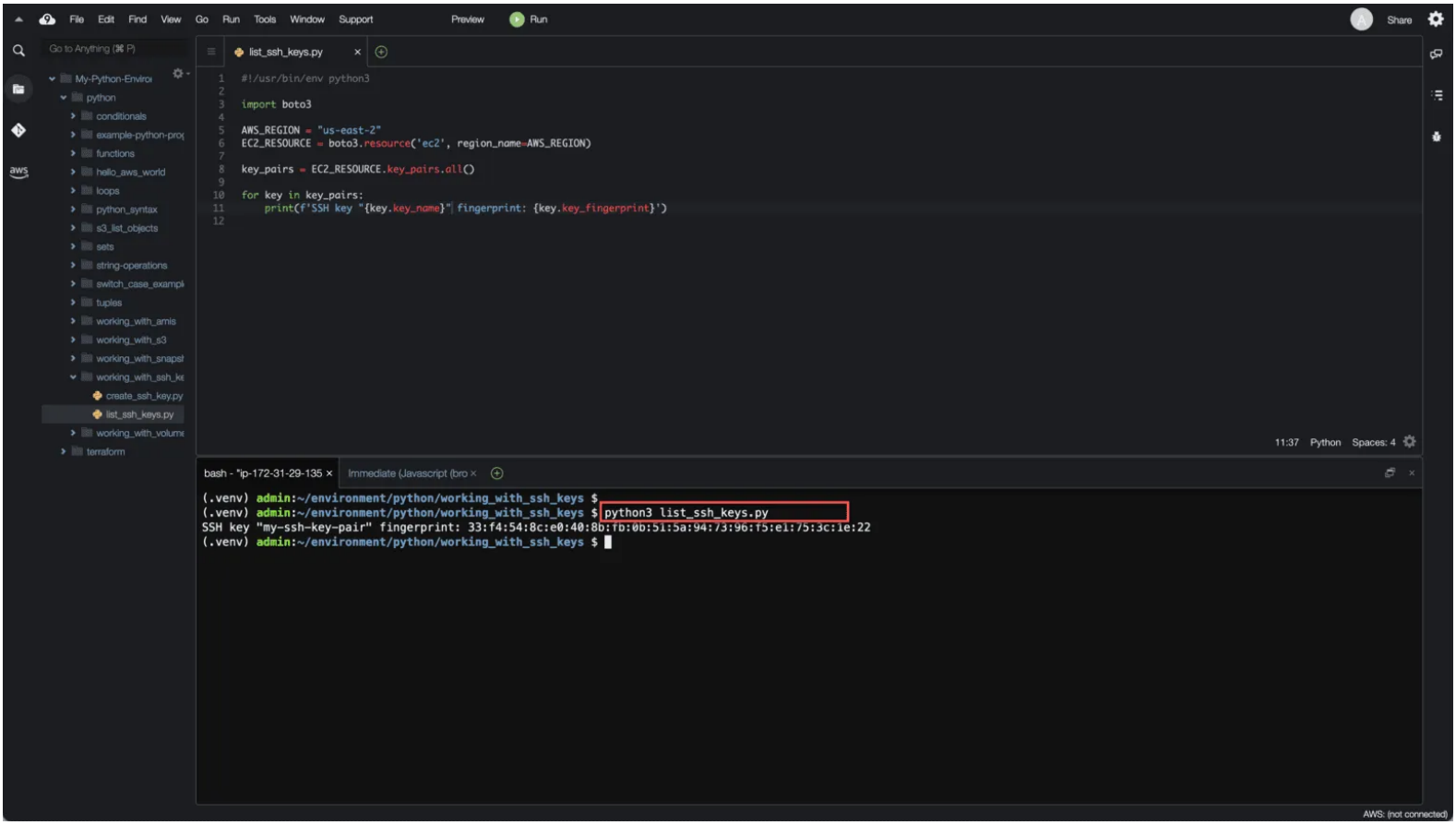
- `key_name`
- `key_fingerprint`
- `tags`

Listing SSH keys

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
```

```
for key in key_pairs:
    print(f'SSH key "{key.key_name}" fingerprint: {key.key_fingerprint}')
```

Here's an example output:



Searching for SSH keys

The key_pairs collection of the EC2 resource allows you to use the `filter()` method to search for specific SSH keys by key id, name, or tag.

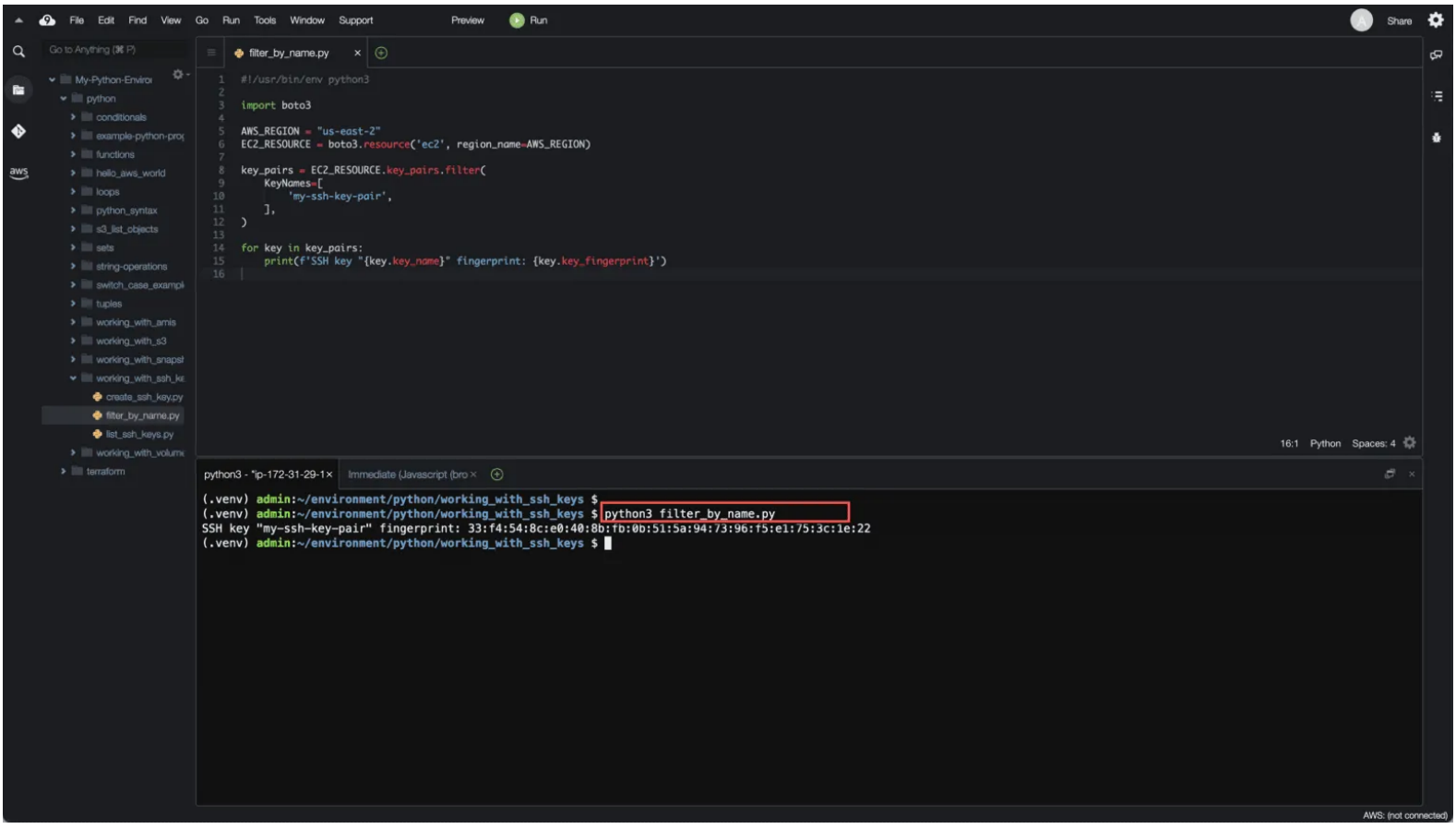
Searching for SSH key by name

Here's an example of filtering SSH key by its name:

Searching for SSH key by name

```
AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)

key_pairs = EC2_RESOURCE.key_pairs.filter(
    KeyNames=[
        'my-ssh-key-pair',
    ],
)

for key in key_pairs:
    print(f'SSH key "{key.key_name}" fingerprint: {key.key_fingerprint}')
```

Here's an execution output:

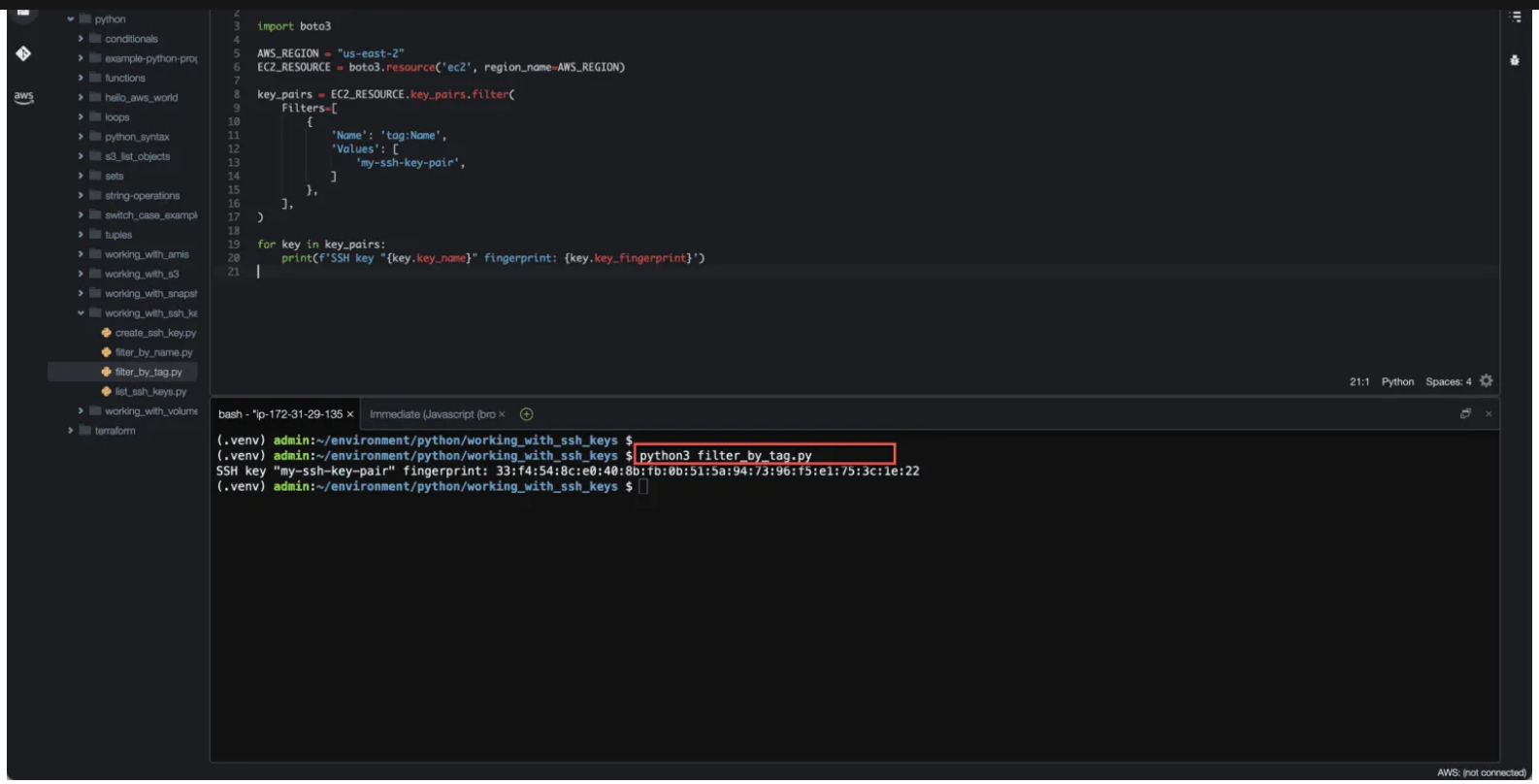


Searching for SSH key by tag

Searching for SSH key by Tag

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)

key_pairs = EC2_RESOURCE.key_pairs.filter(
    Filters=[
        {
            'Name': 'tag:Name',
            'Values': [
                'my-ssh-key-pair',
            ]
        },
    ],
)

for key in key_pairs:
    print(f'SSH key "{key.key_name}" fingerprint: {key.key_fingerprint}')
```

Here's an execution output:

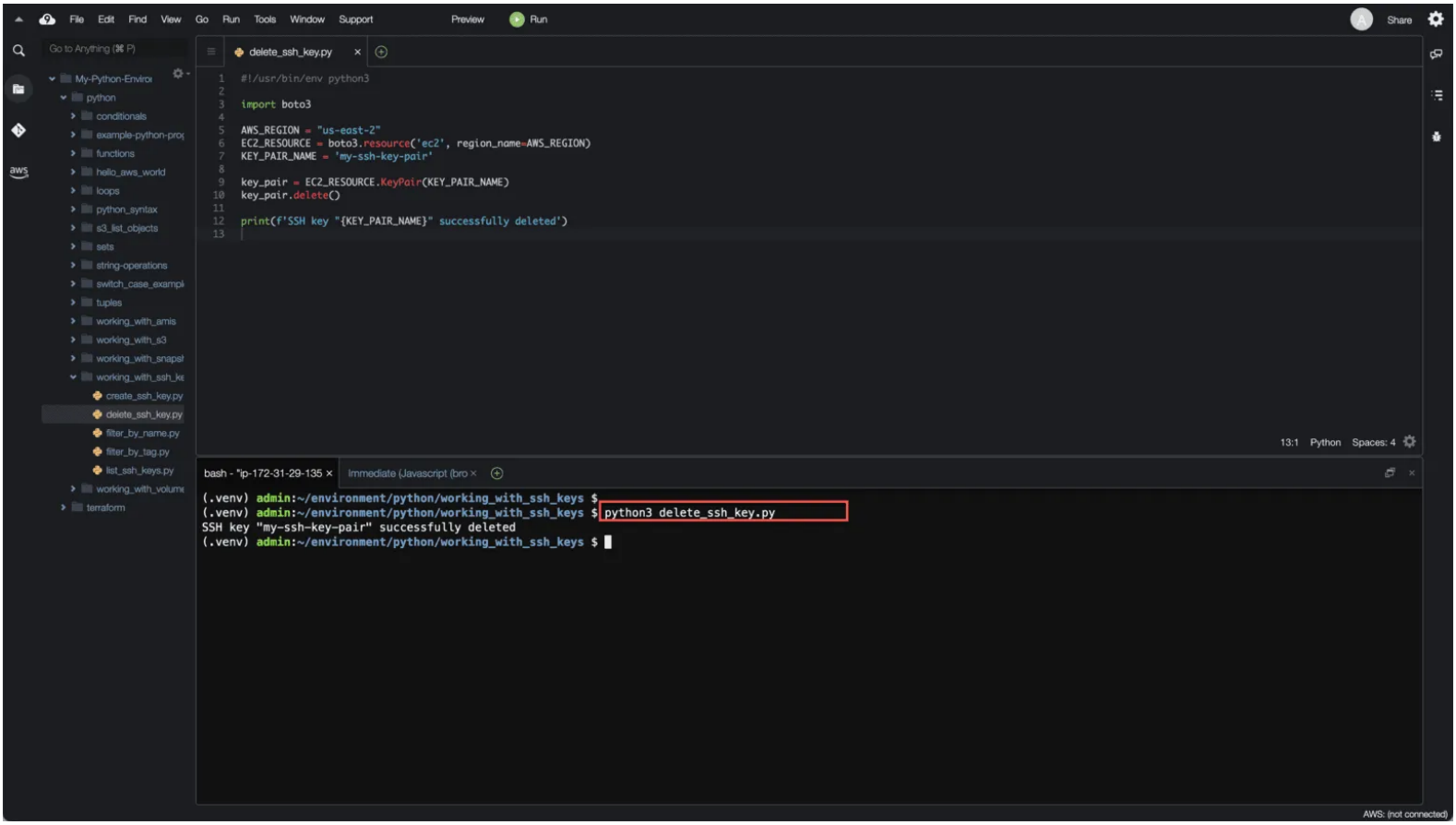


Deleting SSH key

To delete an SSH key pair, you have to use the delete() method of the KeyPair class of the EC2 resource:

Deleting SSH key

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
KEY_PAIR_NAME = 'my-ssh-key-pair'

key_pair = EC2_RESOURCE.KeyPair(KEY_PAIR_NAME)
key_pair.delete()

print(f'SSH key "{KEY_PAIR_NAME}" successfully deleted')
```



Managing Security Groups using Boto3

Security groups control inbound and outbound traffic of the EC2 instance network interface. Each Security Group consists of one or many Security Group Rules. This section of the article will cover how to manage Security Groups and use them with EC2 instances.

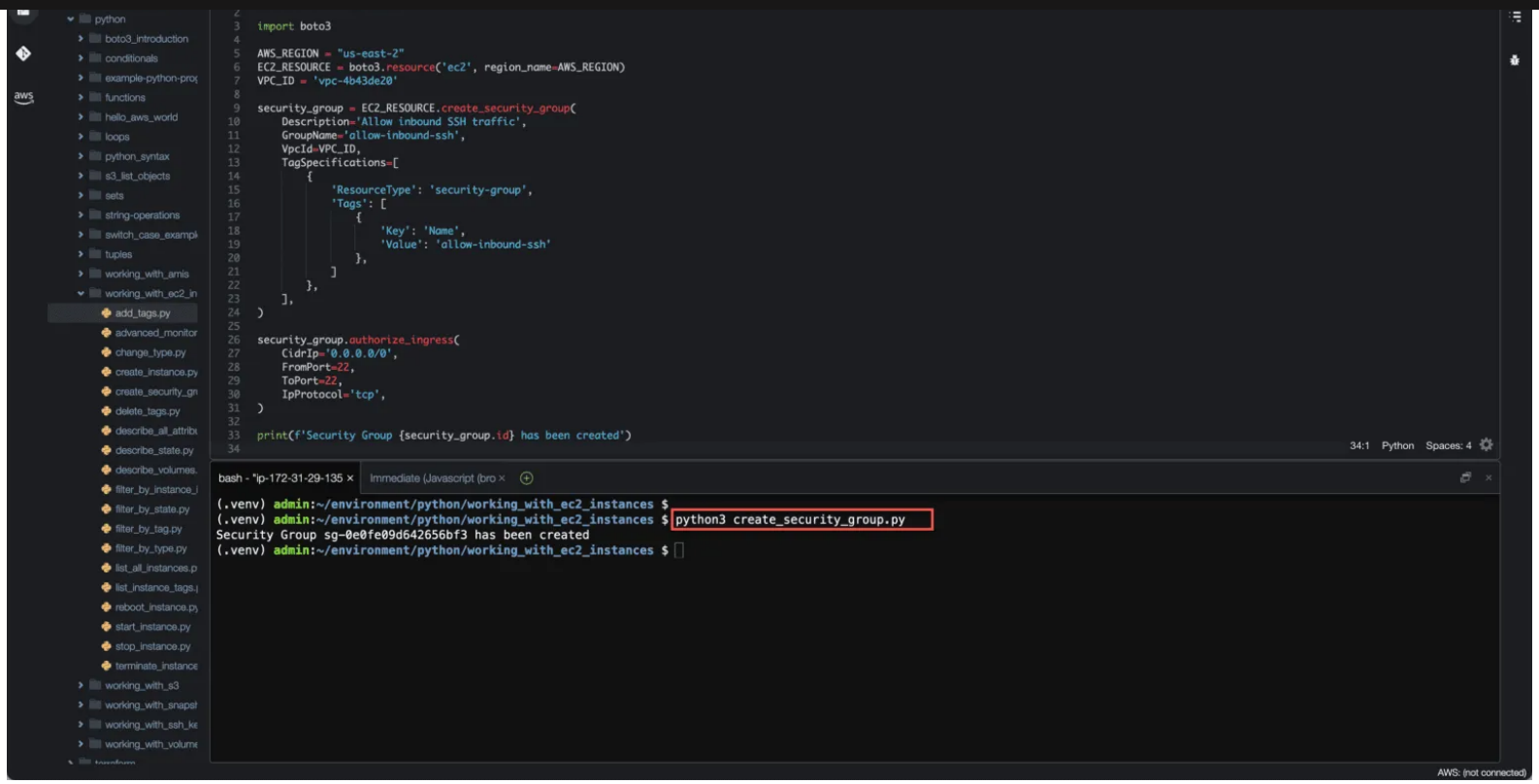
Note: every EC2 instance must have at least one Security Group associated with it. If no Security Group has been specified during the EC2 instance launch, the default Security Group of the default VPC is associated with the instance.

Creating a Security Group

To define a Security Group, you can use the `create_security_group()` of the EC2 resource. To control inbound and outbound traffic, the authorize_ingress() and the authorize_egress() methods are used:

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
VPC_ID = 'vpc-4b43de20'

security_group = EC2_RESOURCE.create_security_group(
    Description='Allow inbound SSH traffic',
    GroupName='allow-inbound-ssh',
    VpcId=VPC_ID,
    TagSpecifications=[
        {
            'ResourceType': 'security-group',
            'Tags': [
                {
                    'Key': 'Name',
                    'Value': 'allow-inbound-ssh'
                },
            ]
        },
    ],
)

security_group.authorize_ingress(
    CidrIp='0.0.0.0/0',
    FromPort=22,
    ToPort=22,
    IpProtocol='tcp',
)

print(f'Security Group {security_group.id} has been created')
```

Here's an execution output:

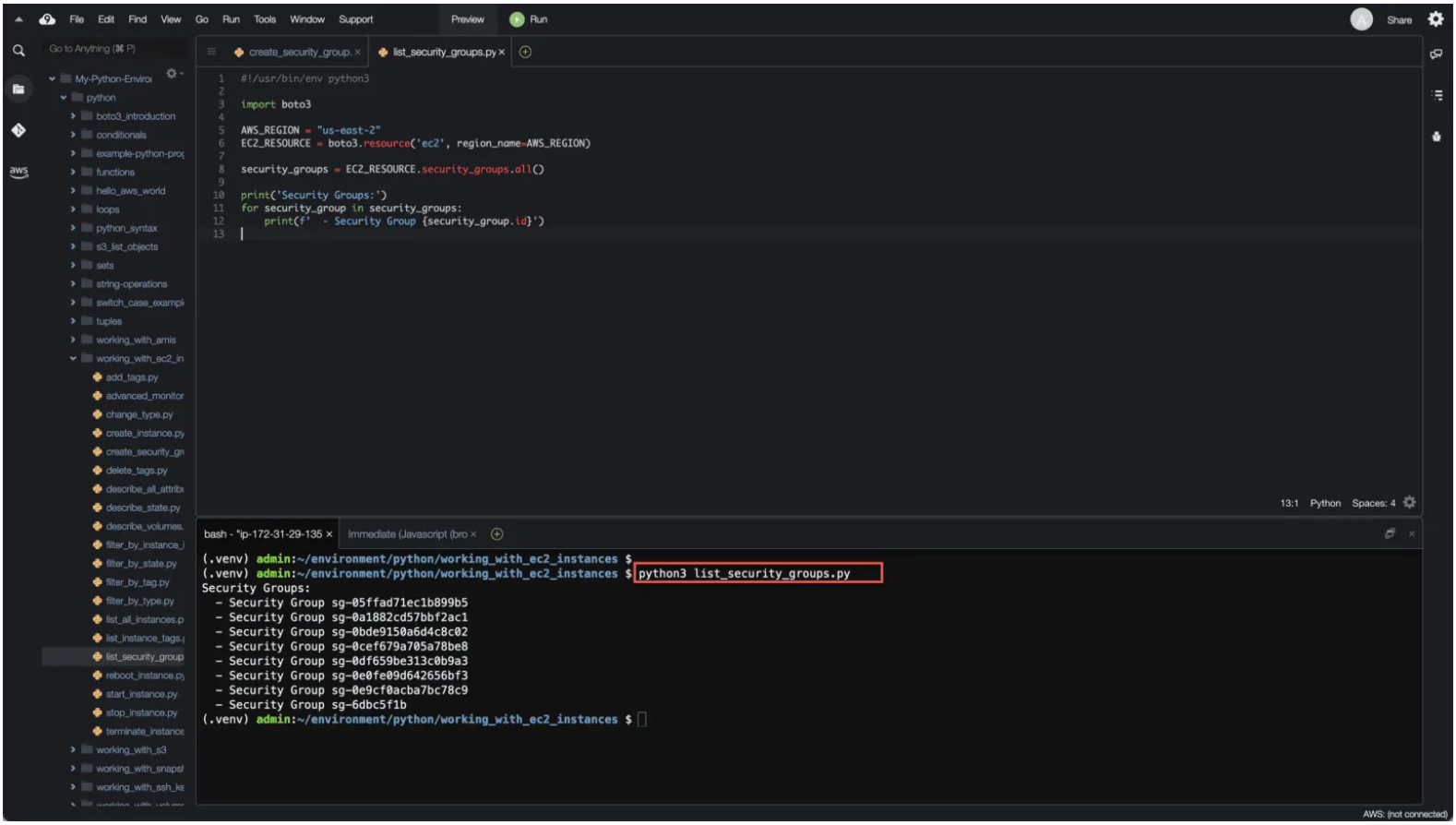


Listing all Security Groups

To list all Security Groups, you can use the `all()` method of the security_groups collection of the EC2 resource:

Listing all Security Groups

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)

security_groups = EC2_RESOURCE.security_groups.all()

print('Security Groups:')
for security_group in security_groups:
    print(f'  - Security Group {security_group.id}')
```



Searching Security Groups

To find Security Groups by specified conditions, you can use the `filter()` method of the security_groups collection of the EC2 resource.

Searching Security Groups by ID

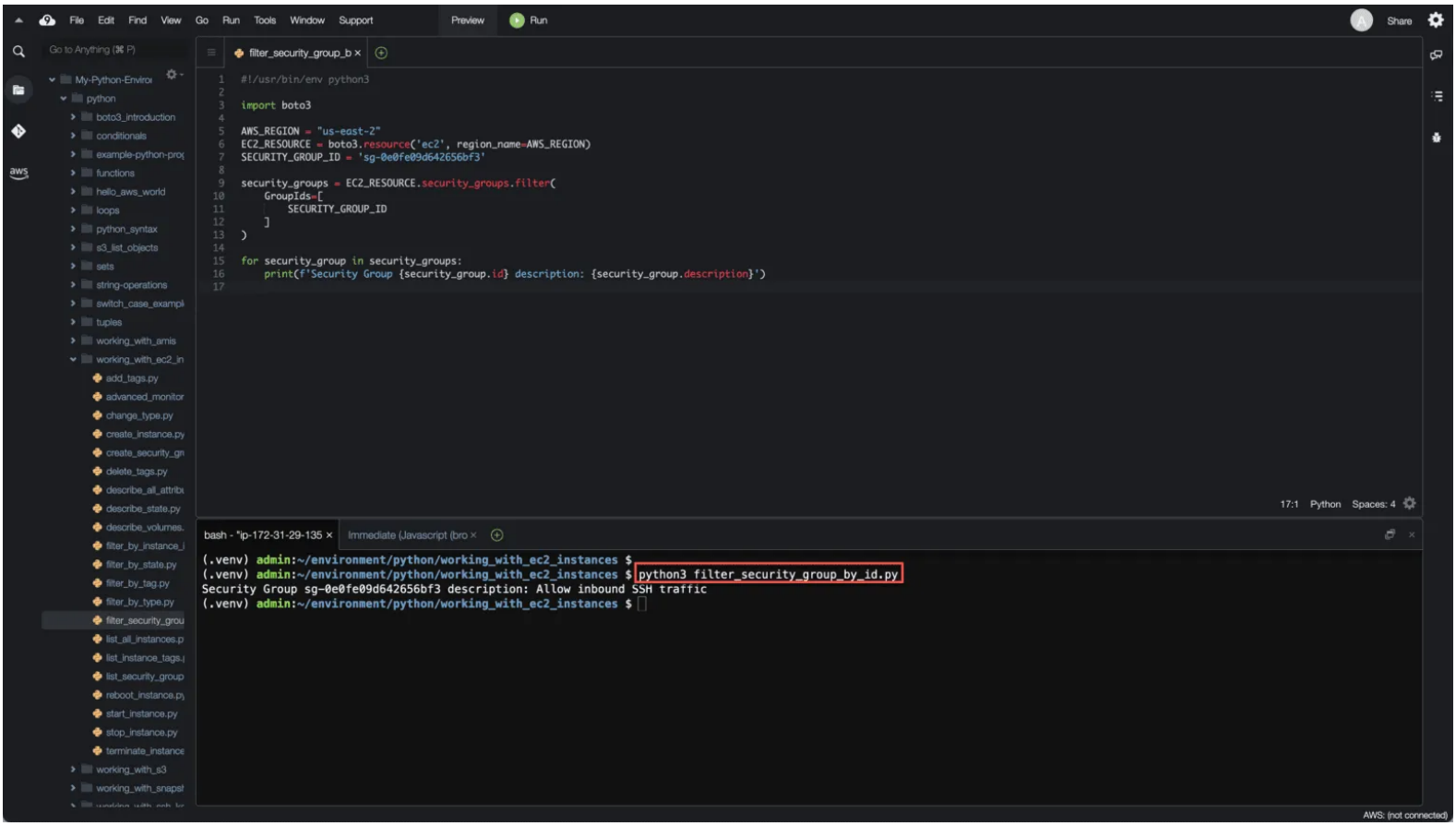
To find Security Groups by the Security Group ID, you can use the `filter()` method of the security_groups collection of the EC2 resource:

Searching Security Groups by ID

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
```

```
security_groups = EC2_RESOURCE.security_groups.filter(
    GroupIds=[
        SECURITY_GROUP_ID
    ]
)

for security_group in security_groups:
    print(f'Security Group {security_group.id} description: {security_group.description}')
```

Here's an execution output:

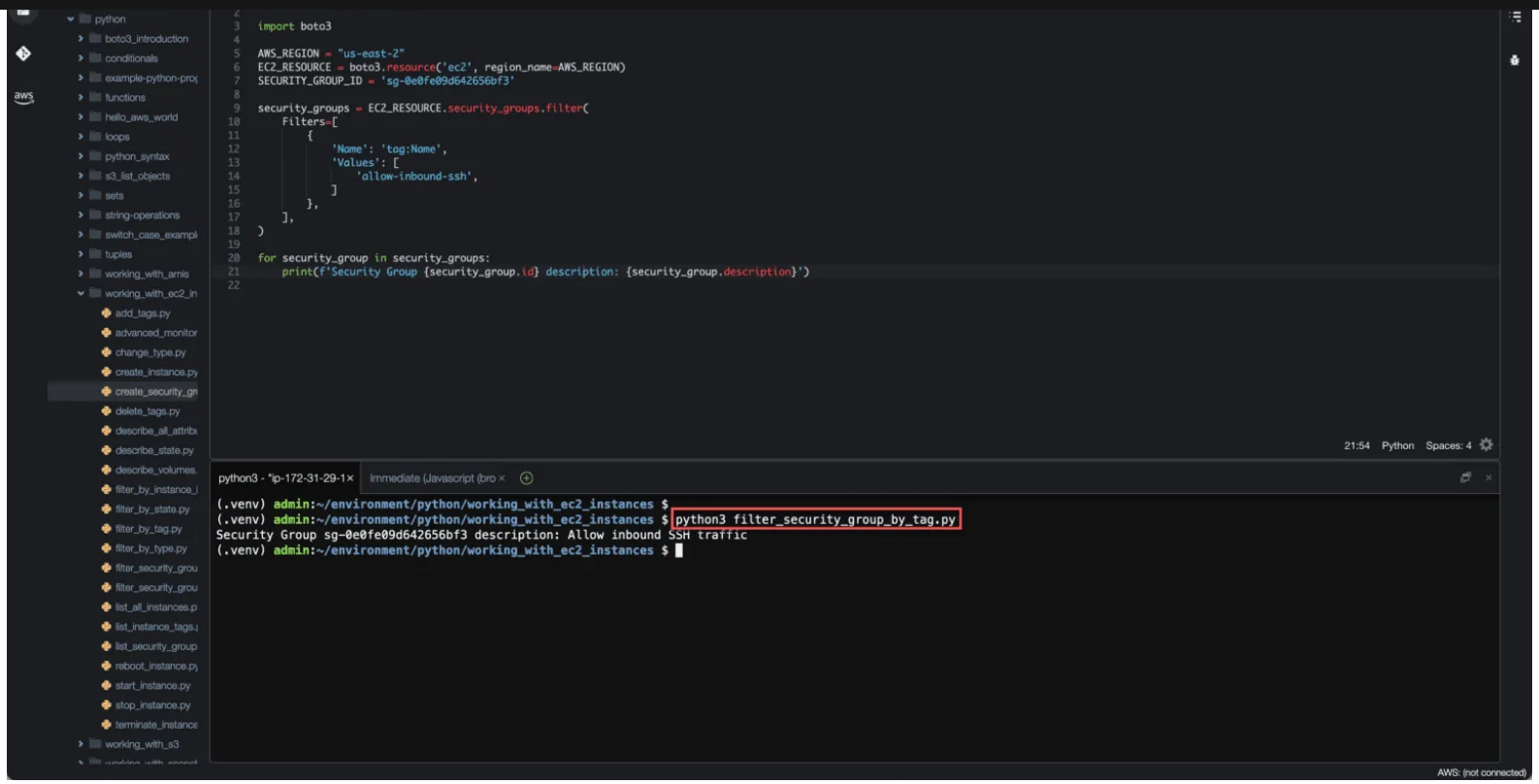


Searching Security Groups by Tag

To find Security Groups by the Tag, you can use the `filter()` method of the security_groups collection of the EC2 resource:

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
SECURITY_GROUP_ID = 'sg-0e0fe09d642656bf3'

security_groups = EC2_RESOURCE.security_groups.filter(
    Filters=[
        {
            'Name': 'tag:Name',
            'Values': [
                'allow-inbound-ssh',
            ]
        },
    ],
)

for security_group in security_groups:
    print(f'Security Group {security_group.id} description: {security_group.description}')
```

Here's an execution output:



Describing Security Groups

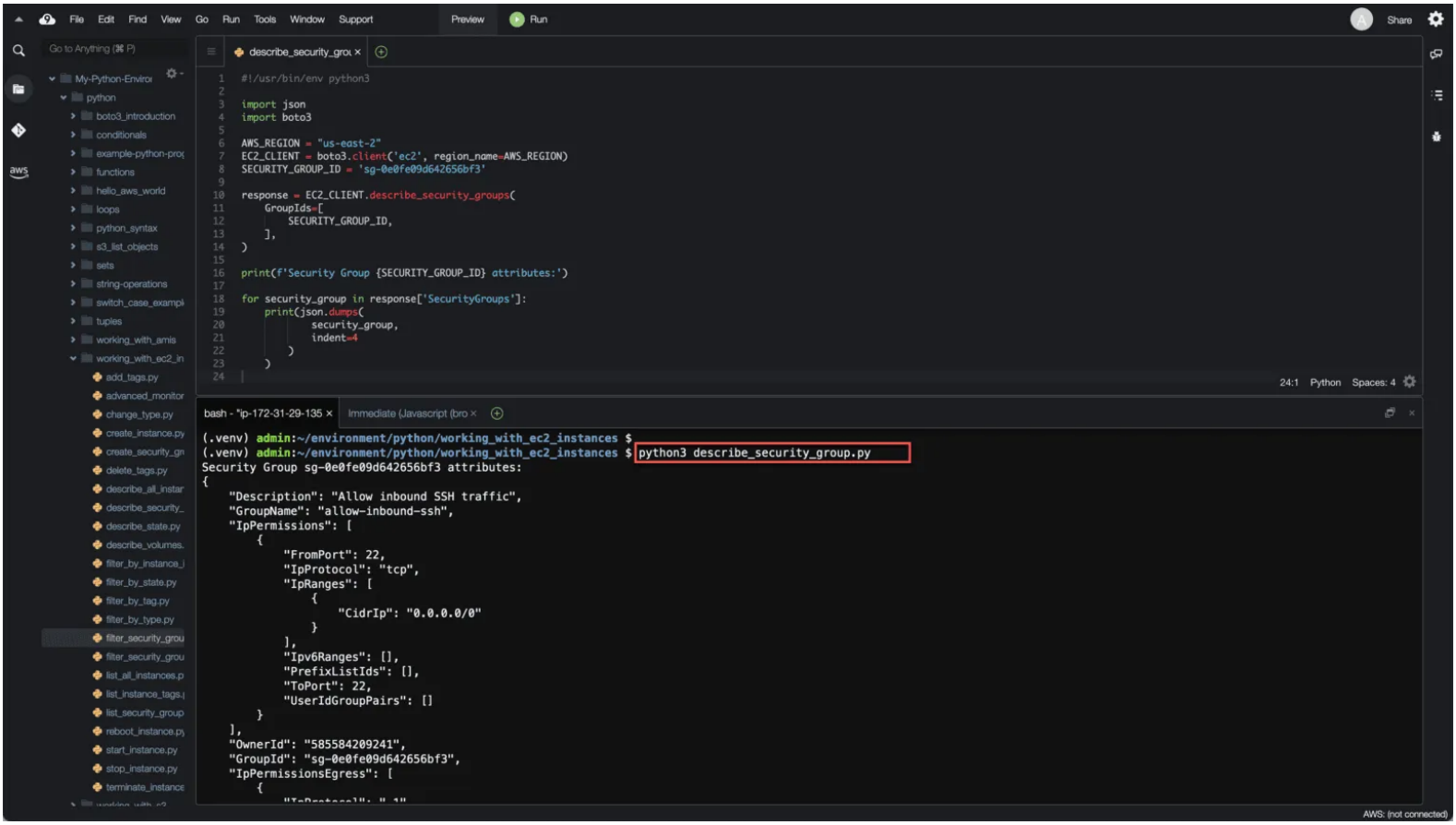
To list all Security Group's properties, you can use the describe_security_groups() method that supports the same search attributes as the `filter()` method of the EC2 resource:

Describing a Security Group using Boto3

```
#!/usr/bin/env python3

import json
import boto3

AWS_REGION = "us-east-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
SECURITY_GROUP_ID = 'sg-0e0fe09d642656bf3'

response = EC2_CLIENT.describe_security_groups(
    GroupIds=[
        SECURITY_GROUP_ID,
    ],
)
```

```
for security_group in response['SecurityGroups']:
    print(json.dumps(
            security_group,
            indent=4
        )
    )
```

Here's an execution output:

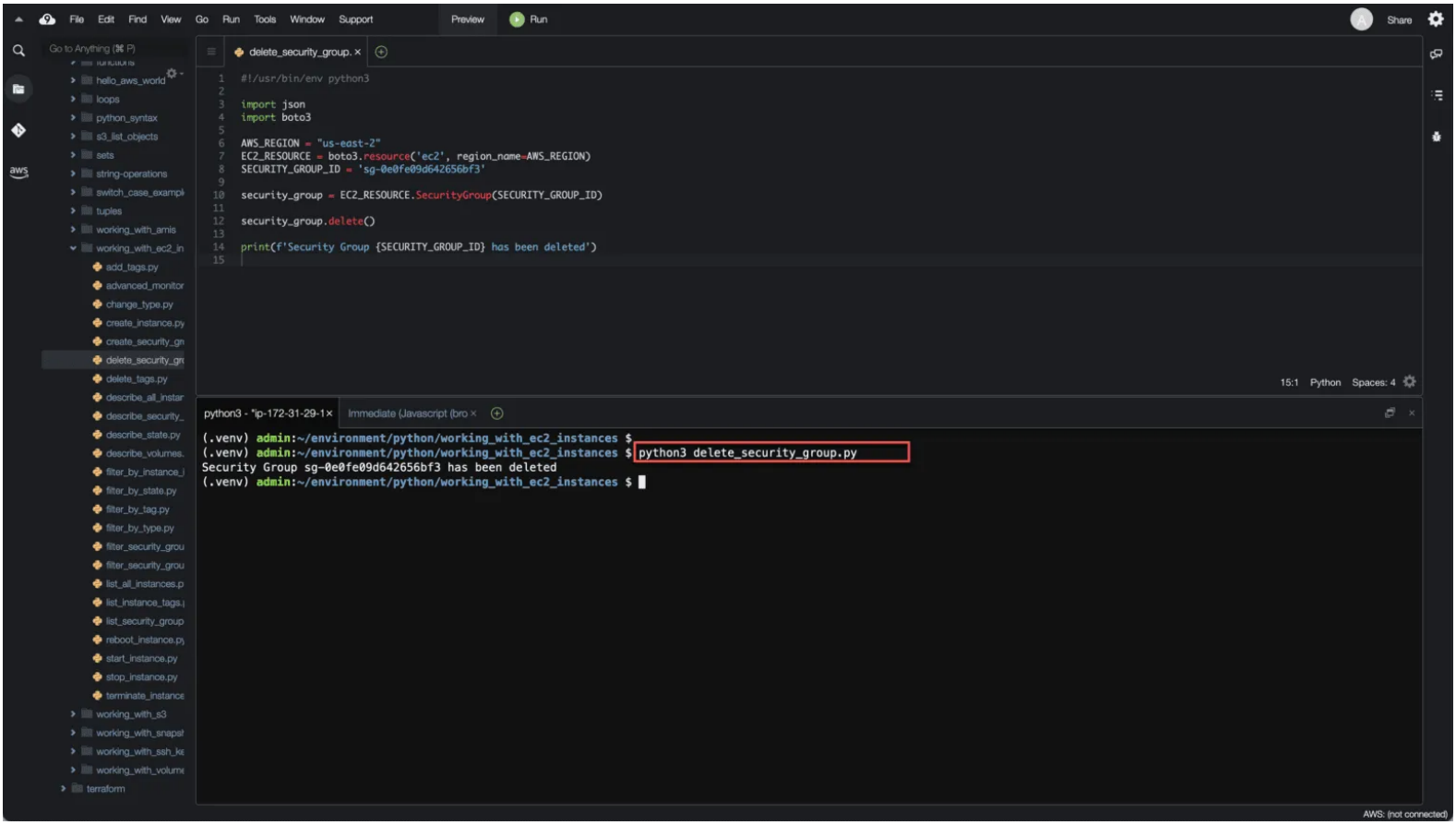


Deleting Security Groups

To delete a Security Group, you can use the delete() method of the SecurityGroup class of the EC2 resource:

```python
import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
SECURITY_GROUP_ID = 'sg-0e0fe09d642656bf3'

security_group = EC2_RESOURCE.SecurityGroup(SECURITY_GROUP_ID)

security_group.delete()

print(f'Security Group {SECURITY_GROUP_ID} has been deleted')
```

Here's an execution output:

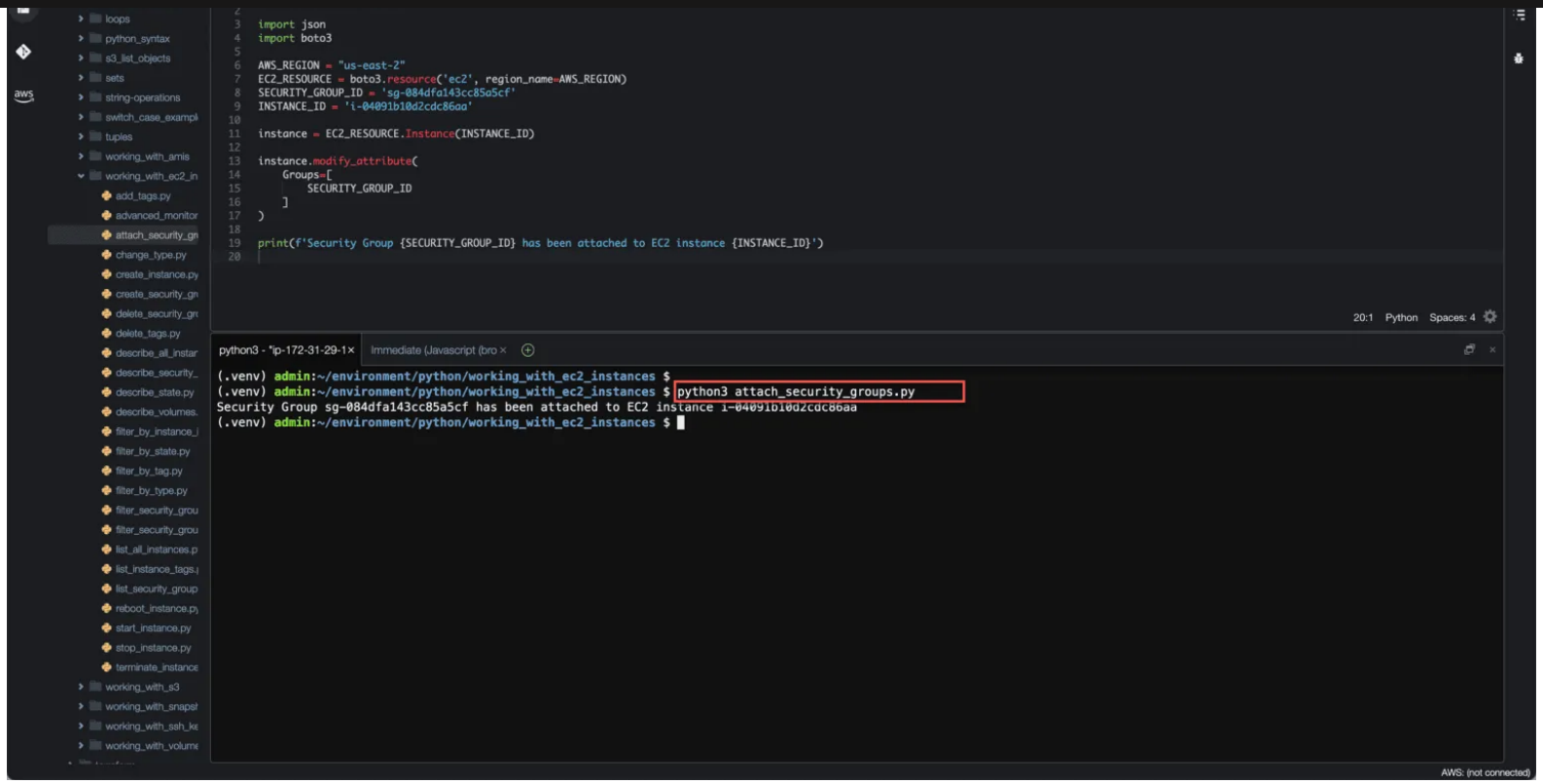


Attaching Security Groups to the EC2 Instance

Attaching Security Groups to the EC2 Instance

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
SECURITY_GROUP_ID = 'sg-084dfa143cc85a5cf'
INSTANCE_ID = 'i-04091b10d2cdc86aa'

instance = EC2_RESOURCE.Instance(INSTANCE_ID)

instance.modify_attribute(
    Groups=[
        SECURITY_GROUP_ID
    ]
)

print(f'Security Group {SECURITY_GROUP_ID} has been attached to EC2 instance {INSTANCE_ID}')
```

Here's an execution output:



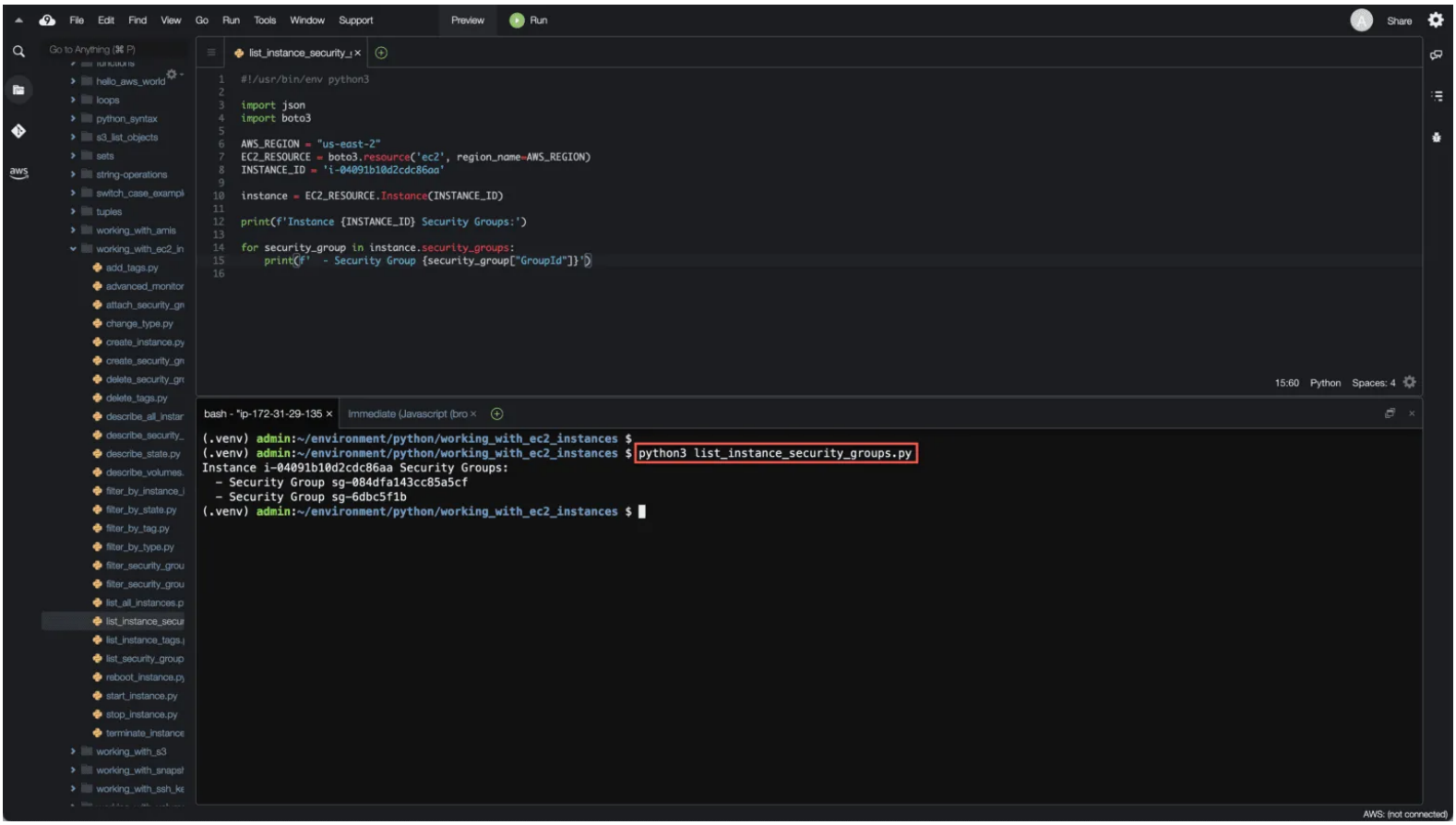
Listing EC2 instance Security Groups

To list EC2 instance Security Groups, you can use the security_groups attribute of the EC2.Instance class of the EC2 resource:

Listing EC2 instance Security Groups

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-04091b10d2cdc86aa'

instance = EC2_RESOURCE.Instance(INSTANCE_ID)

print(f'Instance {INSTANCE_ID} Security Groups:')

for security_group in instance.security_groups:
    print(f'  - Security Group {security_group["GroupId"]}')
```

Here's an execution output:



Detaching Security Group from the EC2 Instance

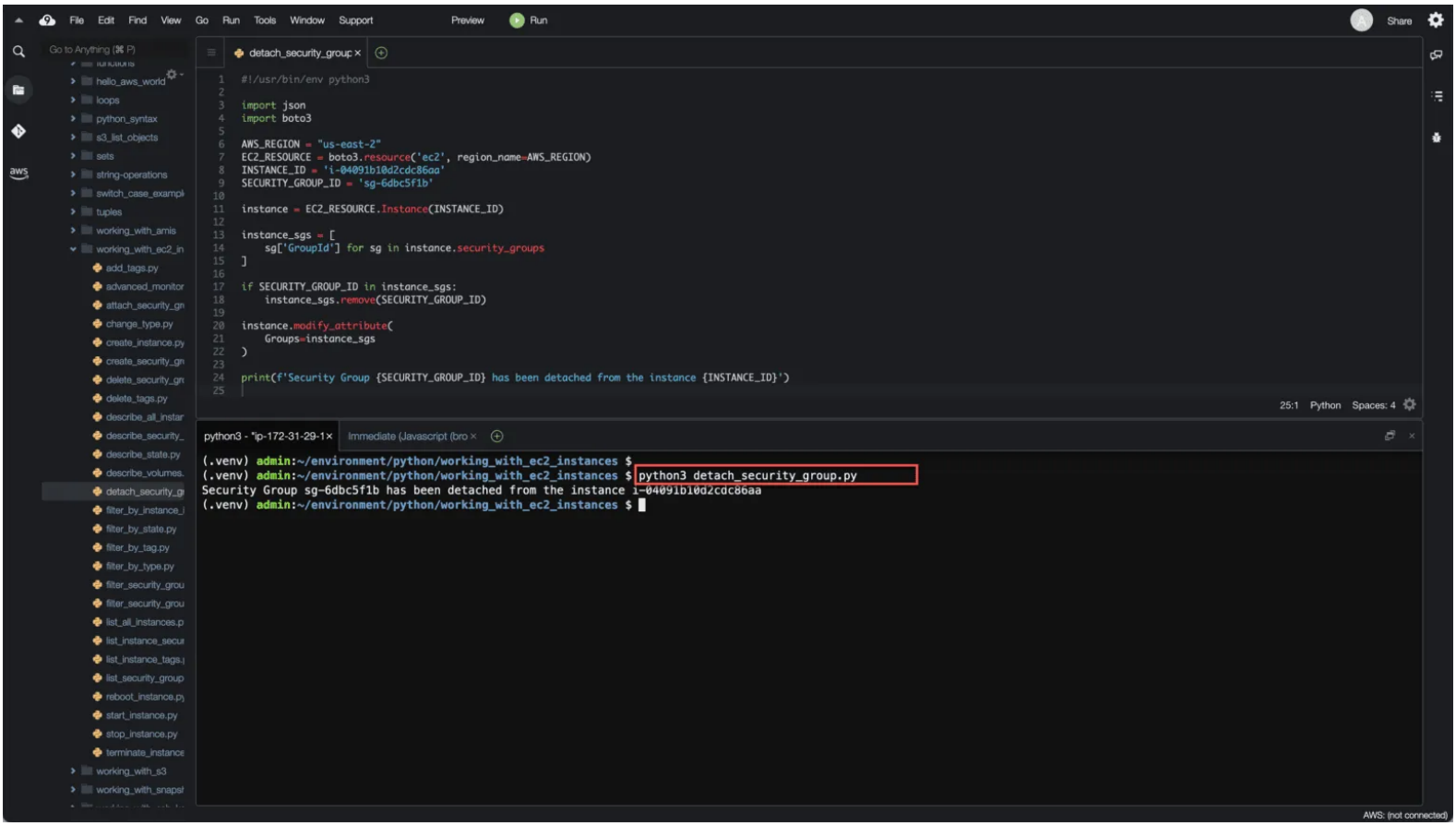
There's no specific API call to detach the Security Group from the EC2 instance. To detach aSecurity Group from the EC2 Instance, you need to get all instance Security Groups as a list, remove the required Security Group from the list, and overrise the `Groups` EC2 instance attribute.

Detaching Security Group from the EC2 Instance

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-04091b10d2cdc86aa'
```

```
instance_sgs = [
    sg['GroupId'] for sg in instance.security_groups
]

if SECURITY_GROUP_ID in instance_sgs:
    instance_sgs.remove(SECURITY_GROUP_ID)

instance.modify_attribute(
    Groups=instance_sgs
)

print(f'Security Group {SECURITY_GROUP_ID} has been detached from the instance {INSTANCE_ID}')
```

Here's an execution output:



How to manage Elastic IP addresses using Boto3?

any moment. By default, each AWS Account can allocate a maximum of five Elastic IP addresses.

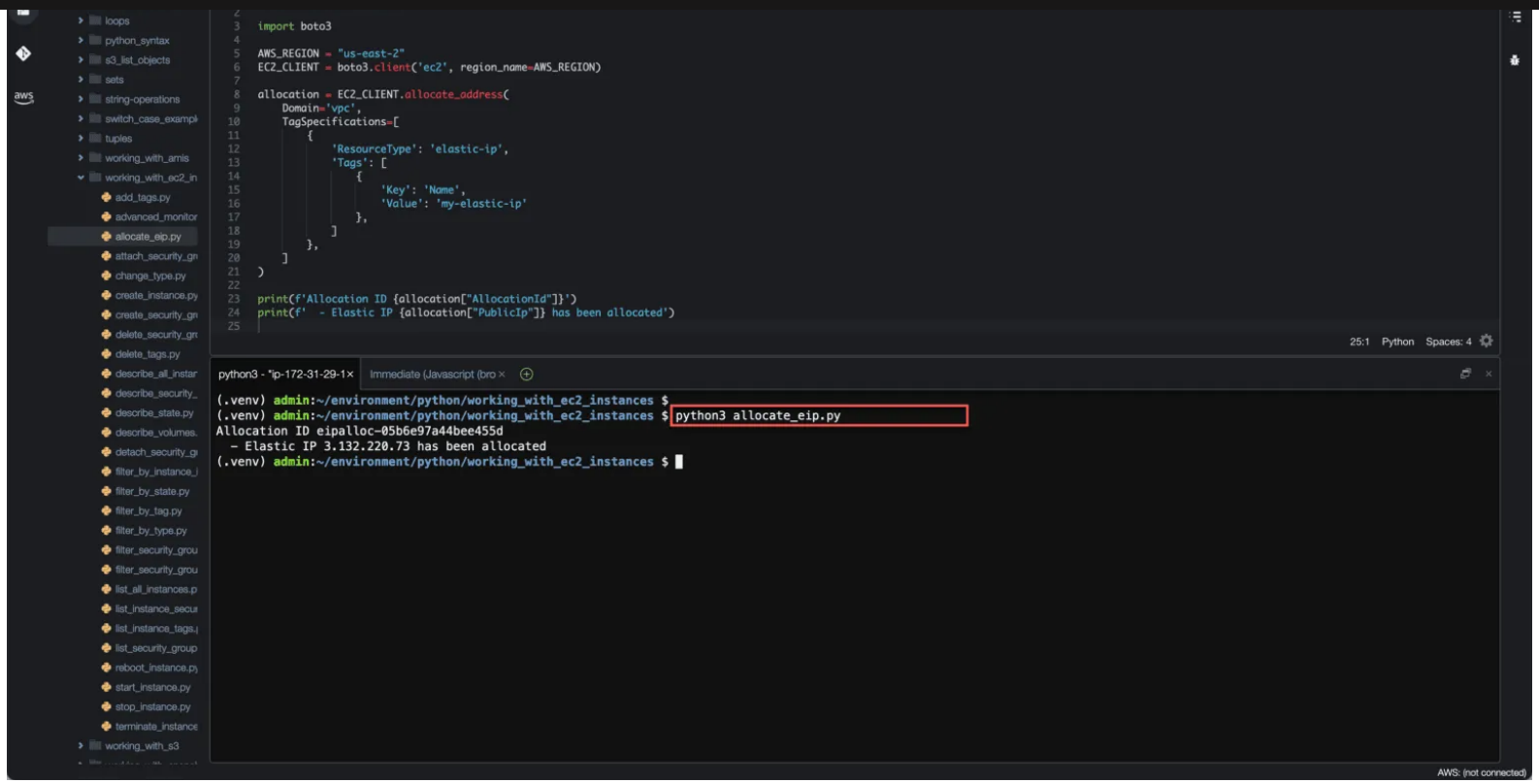
Allocating Elastic IP address

To allocate an Elastic IP address for your AWS account, you can use the allocate_address() method of the EC2 client:

Allocating Elastic IP address

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)

allocation = EC2_CLIENT.allocate_address(
    Domain='vpc',
    TagSpecifications=[
        {
            'ResourceType': 'elastic-ip',
            'Tags': [
                {
                    'Key': 'Name',
                    'Value': 'my-elastic-ip'
                },
            ]
        },
    ]
)

print(f'Allocation ID {allocation["AllocationId"]}')
print(f'  - Elastic IP {allocation["PublicIp"]} has been allocated')
```

Here's an execution output:



Listing and describing Elastic IP addresses

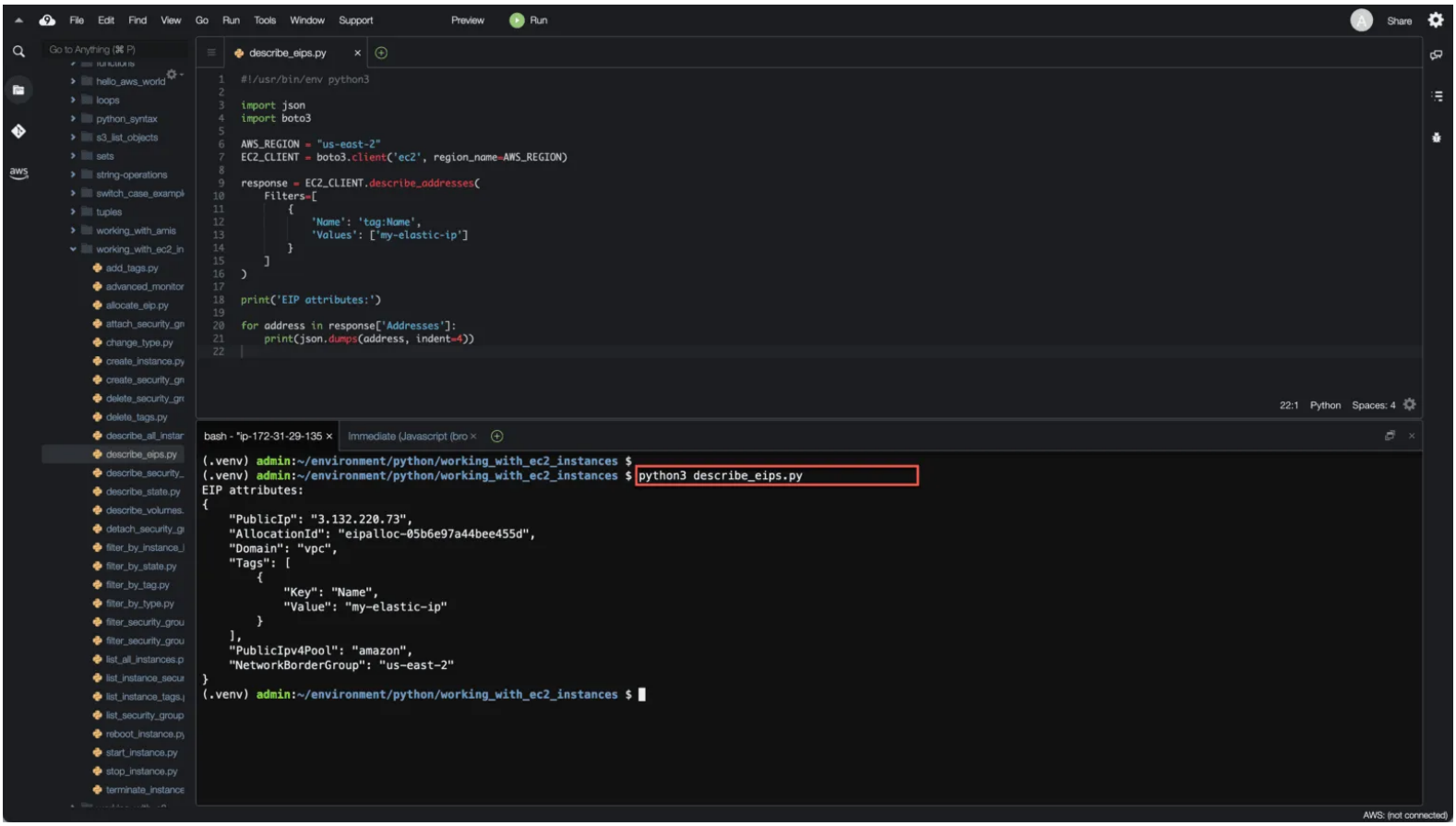
To list and describe Elastic IP addresses, you can use the describe_addresses() method of the EC2 client:

Listing and describing Elastic IP addresses

```
#!/usr/bin/env python3

import json
import boto3

AWS_REGION = "us-east-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)

response = EC2_CLIENT.describe_addresses(
    Filters=[
        {
            'Name': 'tag:Name',
            'Values': ['my-elastic-ip']
        }
    ]
```

```
for address in response['Addresses']:
    print(json.dumps(address, indent=4))
```

Here's an execution output:



Attaching an Elastic IP to an EC2 Instance

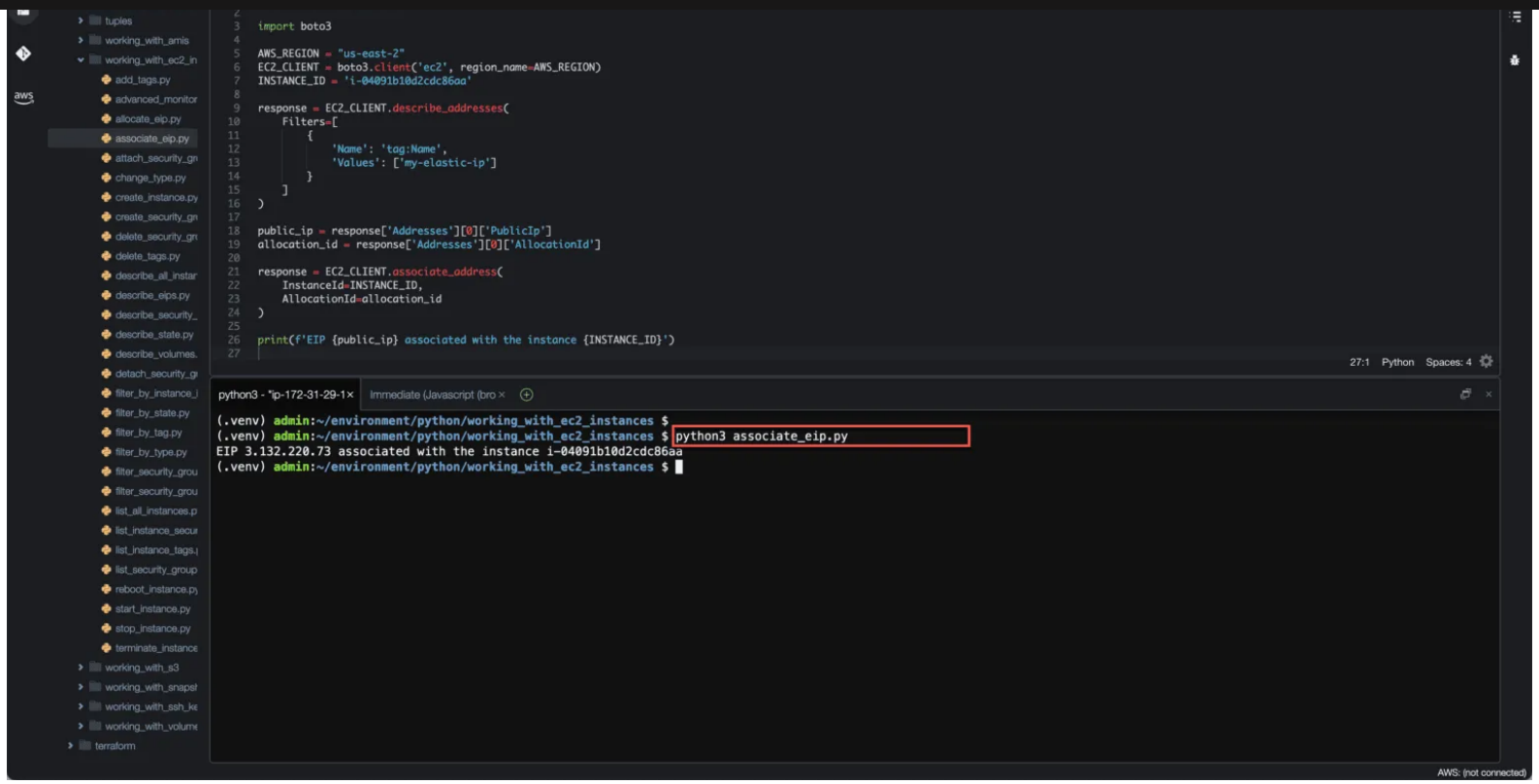
To associate an Elastic IP address with an EC2 Instance, you can use the associate_address() method of the EC2 client:

Attaching an Elastic IP to an EC2 Instance

```
#!/usr/bin/env python3
```

```
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-04091b10d2cdc86aa'

response = EC2_CLIENT.describe_addresses(
    Filters=[
        {
            'Name': 'tag:Name',
            'Values': ['my-elastic-ip']
        }
    ]
)

public_ip = response['Addresses'][0]['PublicIp']
allocation_id = response['Addresses'][0]['AllocationId']

response = EC2_CLIENT.associate_address(
    InstanceId=INSTANCE_ID,
    AllocationId=allocation_id
)

print(f'EIP {public_ip} associated with the instance {INSTANCE_ID}')
```

Here's an execution output:



Detaching an Elastic IP address from an EC2 instance

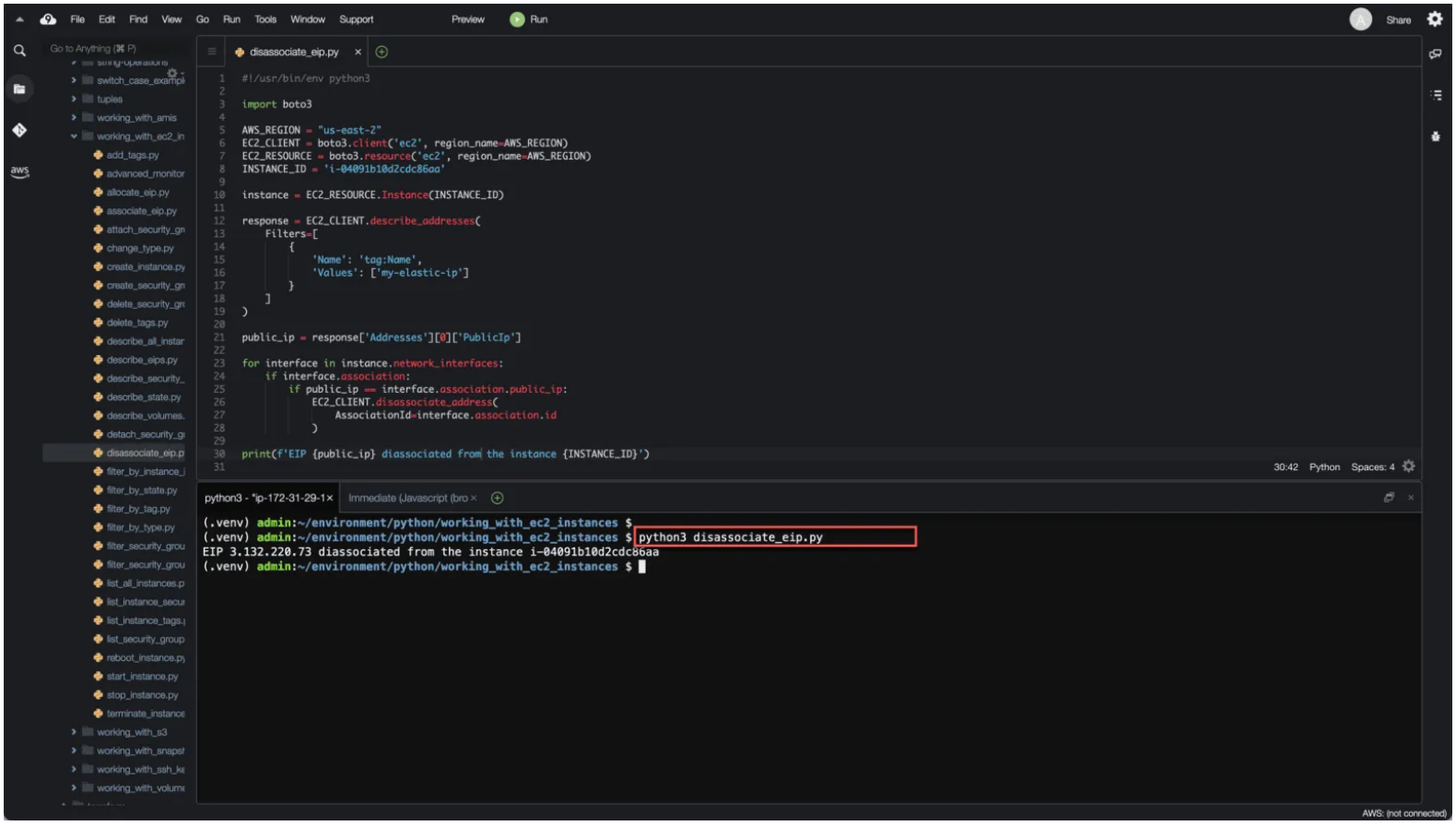
To disassociate (detach) an Elastic IP address from the EC2 instance, you need to use the disassociate_address() method of the EC2 client. The `disassociate_address()` method requires the `AssociationId` argument, which you can find by processing the list of the EC2 instance network interfaces:

Boto3 Dissociate Elastic IP Address

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
INSTANCE_ID = 'i-04091b10d2cdc86aa'

instance = EC2_RESOURCE.Instance(INSTANCE_ID)
```

```
            'Name': 'tag:Name',
            'Values': ['my-elastic-ip']
        }
    ]
)

public_ip = response['Addresses'][0]['PublicIp']

for interface in instance.network_interfaces:
    if interface.association:
        if public_ip == interface.association.public_ip:
            EC2_CLIENT.disassociate_address(
                AssociationId=interface.association.id
            )

print(f'EIP {public_ip} diassociated from the instance {INSTANCE_ID}')
```

Here's an execution output:

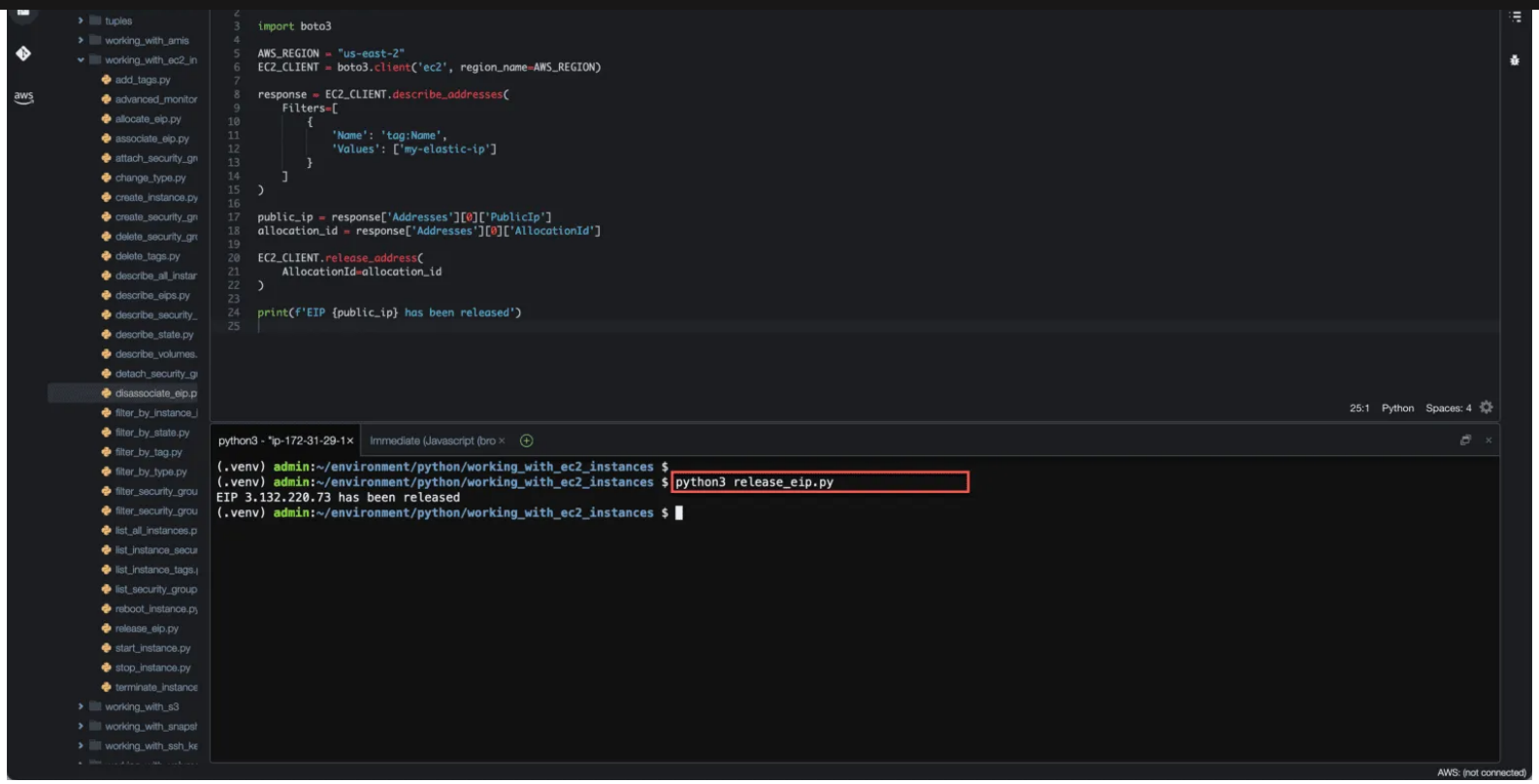


To release an Elastic IP address, you can use the release_address() method of the EC2 client.

Boto3 release elastic ip address

```
#!/usr/bin/env python3

import boto3

AWS_REGION = "us-east-2"
EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)

response = EC2_CLIENT.describe_addresses(
    Filters=[
        {
            'Name': 'tag:Name',
            'Values': ['my-elastic-ip']
        }
    ]
)

public_ip = response['Addresses'][0]['PublicIp']
allocation_id = response['Addresses'][0]['AllocationId']

EC2_CLIENT.release_address(
    AllocationId=allocation_id
)

print(f'EIP {public_ip} has been released')
```

Here's an execution output:



Summary

In this article, we've provided many Python code snippets for creating, starting, stopping, rebooting, filtering, deleting, tagging EC2 Instances using the Boto3. In addition to that, we've covered the management of SSH keys, Security Groups, and Elastic IP addresses.

Related articles

- Quick Intro to Python for AWS Automation Engineers
- Terraform – Managing AWS VPC – Creating Public Subnet
- Security Best Practices For Serverless Applications
- Top 10 SSH Features You MUST Know To Be More Productive
- Working with EBS volumes in Python using Boto3
- CloudFormation Tutorial – How To Automate EC2 Instance In 5 Mins (Example)