



# Working with EC2 instances



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
- How to manage EC2 Instances using Boto3?
  - Creating EC2 instance
  - Listing EC2 Instances
  - Filtering EC2 instances
    - Filtering EC2 instances by state
    - Filtering EC2 instances by type
    - Filtering EC2 instances by Tag
    - Filtering EC2 instances by instance ID
  - Describing EC2 instance properties
    - Describing all EC2 instance properties
    - Listing EC2 instance EBS volumes
    - Getting EC2 instance state
  - Managing EC2 instance Tags
    - Adding Tags to EC2 instance
    - Listing EC2 instance Tags
    - Updating EC2 instance Tags
    - Deleting EC2 instance Tags
  - EC2 instance advanced monitoring
  - Managing EC2 instance state
    - Starting EC2 instance
    - Stopping EC2 instance
    - Rebooting EC2 instance
    - Terminating EC2 instance
  - Modifying EC2 instance attributes using Boto3
    - Changing EC2 instance type
- How to manage SSH keys using Boto3?
  - Creating SSH key



- Searching for SSH key by name
- Searching for SSH key by tag
- Deleting SSH key
- Managing Security Groups using Boto3
  - Creating a Security Group
  - Listing all Security Groups
  - Searching Security Groups
    - Searching Security Groups by ID
    - Searching Security Groups by Tag
  - Describing Security Groups
  - Deleting Security Groups
  - Attaching Security Groups to the EC2 Instance
  - Listing EC2 instance Security Groups
  - Detaching Security Group from the EC2 Instance
- How to manage Elastic IP addresses using Boto3?
  - Allocating Elastic IP address
  - Listing and describing Elastic IP addresses
  - Attaching an Elastic IP to an EC2 Instance
  - Detaching an Elastic IP address from an EC2 instance
  - Releasing the Elastic IP address
- Summary
- Related articles

## 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 journey 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'Public 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 use the `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 use the `filter()` method in the `instances` collection of the EC2 resource:



```
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 use the `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 the EC2 instance properties, you can use the `describe_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.



```
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:

The screenshot shows a code editor interface with a sidebar containing a file tree. The file tree includes categories like 'My-Python-Envir', 'aws', and 'python', with numerous sub-files under each. The main workspace shows a Python script named 'describe\_volumes.py' with the following code:

```
#!/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 device in device_mappings:
    print(f" - Volume {device['Ebs']['VolumeId']} attached as {device['DeviceName']}
```

Below the code editor is a terminal window titled 'bash - \*ip-172-31-29-135 x | Immediate (Javascript (bro x)'. It shows the command being run: `python3 describe_volumes.py`. The terminal output indicates that a volume is attached to the specified EC2 instance:

```
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 describe_volumes.py
Volumes attached to the EC2 instance "i-020b3f1914105320d":
 - Volume vol-06efd7304409af3e attached as /dev/xvda
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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](#).

The screenshot shows a Jupyter Notebook environment. On the left, there's a sidebar with a tree view of files and notebooks, including categories like 'aws' and 'My-Python-Enviro'. The main area has a tab titled 'describe\_state.py' containing the following Python code:

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

Below the code cell, there's a terminal window with the command `python3 describe_state.py` highlighted in red. The terminal output shows:

```
bash - *ip-172-31-29-135 ✘ Immediate Javascript (bro ✘ +)
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 describe_state.py
EC2 instance "i-020b3f1914105320d" state: running
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

A small note at the bottom right of the terminal window says 'AWS: (not connected)'.

## 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:



The screenshot shows a terminal window titled "python3 - \*p-172-31-29-1x" running on a Linux system. The command entered is "python3 add\_tags.py". The output of the command is: "Tags successfully added to the instance i-020b3f1914105320d". The terminal also shows the prompt "(.venv) admin:~/environment/python/working\_with\_ec2\_instances \$". The background shows a file explorer with a tree view of AWS-related Python scripts.

```
aws
├── python
│   ├── boto3_introduction
│   ├── conditionals
│   ├── example-python-pros
│   ├── functions
│   ├── hello_aws_world
│   ├── loops
│   ├── python_syntax
│   ├── s3_list_objects
│   ├── sets
│   ├── string_operations
│   ├── switch_case_examples
│   ├── tuples
│   ├── working_with_amis
│   └── working_with_ec2_instances
│       ├── add_tags.py
│       ├── create_instance.py
│       ├── describe_all_attributes.py
│       ├── describe_state.py
│       ├── describe_volumes.py
│       ├── filter_by_instance_id.py
│       ├── filter_by_state.py
│       ├── filter_by_tag.py
│       ├── filter_by_type.py
│       └── list_all_instances.py
└── working_with_s3
└── working_with_snapshots
└── working_with_ssh_keys
└── working_with_volumes
└── terraform
```

```
25:1 Python Spaces: 4
python3 - *p-172-31-29-1x | Immediate Javascript (bro × ④
(.venv) admin:~/environment/python/working_with_ec2_instances $
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 add_tags.py
Tags successfully added to the instance i-020b3f1914105320d
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:

```
Go to Anything (F5)  
File Edit Find View Go Run Tools Window Support Preview Run  
My-Python-Enviro  
aws  
python  
boto3_introduction  
conditionals  
example_python_pro  
functions  
hello_aws_world  
loops  
python_syntax  
s3_list_objects  
sets  
string_operations  
switch_case_exampl  
tuples  
working_with_amis  
working_with_ec2_in  
add_tags.py  
create_instance.py  
describe_all_attrib  
describe_state.py  
describe_volumes.  
filter_by_instance  
filter_by_state.py  
filter_by_tag.py  
filter_by_type.py  
list_all_instances.p  
list_instance_tags.  
working_with_s3  
working_with_snapsh  
working_with_ssh_ke  
working_with_volum  
terraform  
list_instance_tags.py  
#!/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,  
    ],  
)  
for instance in instances:  
    print(f'EC2 instance {instance.id} tags: ')  
    if len(instance.tags) > 0:  
        for tag in instance.tags:  
            print(f' - Tag: {tag["Key"]}:{tag["Value"]}')  
    else:  
        print(f' - No Tags')  
print('*'*60)  
  
python3 -i p-172-31-29-1x Immediate (Javascript bro...  
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 list_instance_tags.py  
EC2 instance i-020b3f1914105320d tags:  
- Tag: Name=my-ec2-instance  
- Tag: Environment=dev  
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:



The screenshot shows a code editor interface with a sidebar containing a file tree for an AWS project. The main area displays a Python script named `delete_tags.py`. The script imports `boto3`, sets the AWS region to "us-east-2", and defines an EC2 resource. It filters instances by ID and then iterates through them to delete specific tags. A terminal window at the bottom shows the script being run and outputting a success message.

```
aws
python
  - boto3_introduction
  - conditionals
  - example-python-programs
    - functions
      - hello_aws_world
    - loops
    - python_syntax
    - s3_list_objects
    - sets
    - string_operations
    - switch_case_examples
    - tuples
    - working_with_amis
    - working_with_ec2_instances
      - add_tags.py
      - create_instance.py
      - delete_tags.py
      - describe_all_attributes.py
      - describe_state.py
      - describe_volumes.py
      - filter_by_instance_id.py
      - filter_by_state.py
      - filter_by_tag.py
      - filter_by_type.py
      - list_all_instances.py
      - list_instance_tags.py
      - working_with_s3.py
      - working_with_snapshots.py
      - working_with_ssh_keys.py
      - working_with_volumes.py
    - terraform
```

```
python3 -i p-172-31-29-1x | Immediate Javascript (bro x ⌂
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 delete_tags.py
Tags successfully deleted from the instance i-020b3f1914105320d
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:

The screenshot shows a Jupyter Notebook environment. On the left, there's a sidebar with a tree view of files and a search bar. The main area has a code cell containing the provided Python script. Below it is an 'Immediate' output cell showing the terminal session where the script was run and its output.

```

File Edit Find View Go Run Tools Window Support
File Edit Find View Go Run Tools Window Support
Q Go to Anything (F5) advanced_monitoring.ipynb
My-Python-Enviro
python
boto3_introduction
conditionals
example-python-programs
functions
hello_aws_world
loops
python_syntax
s3_list_objects
sets
string_operations
switch_case_examples
tuples
working_with_amis
working_with_ec2_instances
add_tags.py
advanced_monitor
create_instance.py
delete_tags.py
describe_all_attributes.py
describe_state.py
describe_volumes.py
filter_by_instance_id.py
filter_by_state.py
filter_by_tag.py
filter_by_type.py
list_all_instances.py
list_instance_tags.py
working_with_s3
working_with_snapshots
working_with_ssh_keys
working_with_volumes
terraform

```

```

1 #!/usr/bin/env python3
2
3 import boto3
4
5 AWS_REGION = "us-east-2"
6 EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
7 INSTANCE_ID = 'i-020b3f1914105320d'
8
9 instances = EC2_RESOURCE.instances.filter(
10     InstanceIds=[
11         INSTANCE_ID,
12     ],
13 )
14
15 for instance in instances:
16     monitoring_state = instance.monitoring['State']
17
18     if monitoring_state == 'enabled':
19         instance.unmonitor()
20     else:
21         instance.monitor()
22
23 print(f'Instance monitoring: {monitoring_state}')
24

```

```

python3 -i ip-172-31-29-1x | Immediate (Javascript (bro ...
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 advanced_monitoring.py
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 advanced_monitoring.py
Instance monitoring: disabled
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 advanced_monitoring.py
Instance monitoring: enabled
(.venv) admin:~/environment/python/working_with_ec2_instances $ 

```

## 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')
```

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



```
aws
  - python
    - boto3_introduction
    - conditionals
    - example-python-pro
    - functions
    - hello_aws_world
    - loops
    - python_syntax
    - s3_list_objects
    - sets
    - string_operations
    - switch_case_exampl
    - tuples
    - working_with_amis
    - working_with_ec2_in
      - add_tags.py
      - advanced_monitor
      - create_instance.py
      - delete_tags.py
      - describe_all_attrib
      - describe_state.py
      - describe_volumes
      - filter_by_instance
      - filter_by_state.py
      - filter_by_tag.py
      - filter_by_type.py
      - list_all_instances.p
      - list_instance_tags.
      - start_instance.py
      - stop_instance.py
    - working_with_s3
    - working_with_snapsh
    - working_with_ssh_ke
    - working_with_volum
  - terraform
```

```
bash - *p-172-31-29-135 ✘ Immediate Javascript (bro ✘ +)
(.venv) admin:~/environment/python/working_with_ec2_instances $ .venv/bin/python3 start_instance.py
Starting EC2 instance: i-020b3f1914105320d
EC2 instance "i-020b3f1914105320d" has been started
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

AWS: (not connected)

## 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()
```



The [wait\\_until\\_stopped\(\)](#) waiter method allows you to wait till the EC2 instance is completely stopped.

The screenshot shows a Jupyter Notebook interface with a sidebar containing a file tree. The main area has a code cell with the following Python script:

```
#!/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()
print(f'EC2 instance "{instance.id}" has been stopped')
```

Below the code cell is a terminal window showing the execution of the script and its output:

```
bash - *ip-172-31-29-135 x | Immediate (Javascript (bro x | +)
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 stop_instance.py
Stopping EC2 instance: i-020b3f1914105320d
EC2 instance "i-020b3f1914105320d" has been stopped
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:

The screenshot shows a Jupyter Notebook interface. On the left, there is a sidebar with a tree view of files and notebooks, including a folder named "My-Python-Envir" containing various Python scripts like "reboot\_instance.py", "add\_tags.py", etc. The main area has a tab titled "reboot\_instance.py". Below it, a code cell contains the following Python script:

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

Below the code cell is an "Immediate" output pane. It shows the command being run: ".venv) admin:~/environment/python/working\_with\_ec2\_instances \$ python3 reboot\_instance.py". The output of the command is: "EC2 instance "i-020b3f1914105320d" has been rebooted". The status bar at the bottom right indicates "14:1 Python Spaces: 4".

## 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')
```

The [wait\\_until\\_terminated\(\)](#) waiter method allows you to wait till the EC2 instance is completely terminated.

```
File Edit Find View Go Run Tools Window Support Share
```

```
Go Anything (⌘ P)
```

```
My-Python-Enviro
  + python
    + boto3_introduction
    + conditionals
    + example-python-project
    + functions
    + hello_aws_world
    + loops
    + python_syntax
    + s3_list_objects
    + sets
    + string_operations
    + switch_case_examples
    + tuples
    + working_with_amis
  + working_with_ec2_instances
    + add_tags.py
    + advanced_monitor
    + create_instance.py
    + delete_tags.py
    + describe_all_attributes
    + describe_state.py
    + describe_volumes
    + filter_by_instance_id
    + filter_by_state.py
    + filter_by_tag.py
    + filter_by_type.py
    + list_all_instances.py
    + list_instance_tags.py
    + reboot_instance.py
    + start_instance.py
    + stop_instance.py
    + terminate_instance
  + working_with_s3
  + working_with_snapshots
  + working_with_ssh_keys
  + working_with_volumes
  + terraform
```

```
terminate_instance.py
```

```
#!/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.terminate()
print(f'Terminating EC2 instance: {instance.id}')
instance.wait_until_terminated()
print(f'EC2 instance "{instance.id}" has been terminated')
```

```
python3 -i p-172-31-29-1x | Immediate Javascript (bro x)
```

```
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 terminate_instance.py
Terminating EC2 instance: i-020b3f1914105320d
EC2 instance "i-020b3f1914105320d" has been terminated
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:



The screenshot shows a terminal window with the following content:

```
python3 -i p-172-31-29-1x | Immediate Javascript (bro x +)
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 change_type.py
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 change_type.py
Instance type has been successfully changed.
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

The terminal window has a title bar "python3 - i-p-172-31-29-1x" and a status bar "24:1 Python Spaces: 4". The bottom right corner of the window frame says "AWS: (not connected)".

## 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:



The screenshot shows a terminal window with the following command and output:

```
bash - *ip-172-31-29-135 x Immediate (Javascript (bro x +)
(.venv) admin:~/environment/python/working_with_ssh_keys $
(.venv) admin:~/environment/python/working_with_ssh_keys $ python3 create_ssh_key.py
SSH key fingerprint: 33:f4:54:8c:e0:40:8b:fb:0b:51:5a:94:73:96:15:e1:75:3c:le:22
Private SSH key: -----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAmuacepc+9tB93VL8C0dCU0u10q5JUo1PL6yZtTx5CXEACn0
Z3G1uZgzW0EtifivLYbM1Q5zzazqgYHvuU1i/WAi5oqkJZS1kyjVHGRd7oGv0lqT9
ZdBx2SKuC5TdyBdggcBVk1ghPj0BHO2yVCiqh-qSzol2YUsn2XD8MWFgE
9ovCaw5SiIc70JV20e0G5kCDzhrh0/Q05Kx9ibDKxwmk1t0poWoD50zGXzEH4ugs
ntw1ExLYL6uiEcV4wg7jzK3bdXSex2Av71On21ejUBzlf1u++dH8v19+0tKG9u
2DhzGKyjwOurNBFWzYmMzf902Et2305ZT98wIDAQABaoIBAFIM1a0/AIZhhdQ
7EB4CjqxW0S1fAxeTrqJXyS2/HTA60wHlyRtw6ZJfmChAG8aopzglB/Bz4sQEUb
ooUmkrlijusf0chG5vrhuFCbn@NSw-A1/uxhzog8KgwouM1zKs1X/4GgZmcHq
Q2cAcazG9xhvPHG69XhcoYjbeW7k4cjB0ddLwJxq3Hkan6+LB506AxlbBV
vo1JAj0sNsChDVGE4qXeHdasR49r/bdTGTxctrNn1CBopxonTPs31VJMTK
6AEjy0h231m1ZjGJJp4CnInCn0YgCDNw1gNzhyeB1IU0tTALy6LITTLzYnH1DJ
alDrRaEcqYEAT7uMxL078eSggj0Hcn1fHzanPUHRjzqjkxwtnu1095hk45OBerH
St1F1ndzCvl13Sh1cW1m95eZbD150ypposy07rgc0wDRZtYf5ZL1cf4Byc2gz/B
LWhig700EoaIxCGNPKxp0mQRn1dfch4XVP5YPtvhTh0PGChtocvVbfKcgYApX3I
J41DciFCEW)7NE0s4KmfBFW7wEjJ/70+qYFTeoJvqcs0mCfLejjaa8sK+k0ste
oZCwKgr38hY78p3vBAcUKlM2dJf+NSE52UPP3pxpchiia3UkxGP1uVM0m6ttnmu
da1f3kszwM+y590033ufVBAfwcL8tY8AISpuWysCgYB0nYXhi6BxCgv1wuby4Z
-----END RSA PRIVATE KEY-----
```

AWS: (not connected)

## 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 `KeyValuePair` 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:

The screenshot shows a Jupyter Notebook environment. On the left, there's a sidebar with a tree view of files and notebooks, including "My-Python-Enviro", "python", "functions", "hello\_aws\_world", "loops", "python\_syntax", "s3\_list\_objects", "sets", "string-operations", "switch\_case\_exampl", "tuples", "working\_with\_amis", "working\_with\_s3", "working\_with\_snapsh", "working\_with\_ssh\_ke", "create\_ssh\_key.py", "list\_ssh\_keys.py", and "working\_with\_volume". The main area has two tabs: "list\_ssh\_keys.py" and "Immediate (JavaScript (bro) x)". The "list\_ssh\_keys.py" tab contains the following Python code:

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

The "Immediate (JavaScript (bro) x)" tab shows the terminal output of running the script:

```
(.venv) admin:~/environment/python/working_with_ssh_keys $ (.venv) admin:~/environment/python/working_with_ssh_keys $ python3 list_ssh_keys.py  
SSH key "my-ssh-key-pair" fingerprint: 33:f4:54:8c:e0:40:8b:fb:0b:51:5a:94:73:96:f5:e1:75:3c:1e:22  
.venv) admin:~/environment/python/working_with_ssh_keys $
```

## 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:



```
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:

```
#!/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(
    KeyNames=[
        'my-ssh-key-pair',
    ],
)
for key in key_pairs:
    print(f'SSH key "{key.key_name}" fingerprint: {key.key_fingerprint}')
```

```
python3 - *ip-172-31-29-1x | Immediate (Javascript (bro x)
```

```
(.venv) admin:~/environment/python/working_with_ssh_keys $ (.venv) admin:~/environment/python/working_with_ssh_keys $ python3 filter_by_name.py
SSH key "my-ssh-key-pair" fingerprint: 33:f4:54:8c:e0:40:8b:fb:0b:51:5a:94:73:96:f5:e1:75:3c:1e:22
(.venv) admin:~/environment/python/working_with_ssh_keys $
```

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:



The screenshot shows a terminal window with the following content:

```
aws
python
  - conditions
  - example-python-project
  - functions
  - hello_aws_world
  - loops
  - python_syntax
  - s3_list_objects
  - sets
  - string_operations
  - switch_case_examples
  - tuples
  - working_with_amis
  - working_with_s3
  - working_with_snapshots
  - working_with_ssh_keys
    - create_ssh_key.py
    - filter_by_name.py
    - filter_by_tag.py
    - list_ssh_keys.py
  - working_with_volumes
  - terraform

bash - *ip-172-31-29-135 ~ Immediate (Javascript (bro ~
(.venv) admin:~/environment/python/working_with_ssh_keys $ .venv) admin:~/environment/python/working_with_ssh_keys $ python3 filter_by_tag.py
SSH key "my-ssh-key-pair" fingerprint: 33:f4:54:8c:ed:40:8b:fb:0b:51:5a:94:73:96:f5:e1:75:3c:1e:22
(.venv) admin:~/environment/python/working_with_ssh_keys $ 
```

AWS: (not connected)

## 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')
```



The screenshot shows a Jupyter Notebook interface with a dark theme. On the left, there's a sidebar with a tree view of files and notebooks, including categories like 'My-Python-Enviro', 'aws', and 'working\_with\_aws'. The main area has two tabs: 'delete\_ssh\_key.py' and 'Immediate Javascript (bro)'. The 'delete\_ssh\_key.py' tab contains the following Python code:

```
#!/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')
```

The 'Immediate Javascript (bro)' tab shows a terminal window with the command 'python3 delete\_ssh\_key.py' run, resulting in the output: 'SSH key "my-ssh-key-pair" successfully deleted'. The status bar at the bottom right indicates '13:1 Python Spaces: 4'.

## 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 more 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:



The screenshot shows a Jupyter Notebook environment. On the left, there's a sidebar titled 'aws' containing a tree view of Python scripts under the 'python' directory. A file named 'create\_security\_group.py' is selected. The main area displays the code for creating a security group:

```
1 import boto3
2
3 AWS_REGION = "us-east-2"
4 EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
5 VPC_ID = 'vpc-4b43de20'
6
7 security_group = EC2_RESOURCE.create_security_group(
8     Description='Allow inbound SSH traffic',
9     GroupName='allow-inbound-ssh',
10    VpcId=VPC_ID,
11    TagSpecifications=[
12        {
13            'ResourceType': 'security-group',
14            'Tags': [
15                {
16                    'Key': 'Name',
17                    'Value': 'allow-inbound-ssh'
18                },
19            ],
20        },
21    ],
22),
23
24
25 security_group.authorize_ingress(
26     CidrIp='0.0.0.0/0',
27     FromPort=22,
28     ToPort=22,
29     IpProtocol='tcp',
30 )
31
32
33 print(f'Security Group {security_group.id} has been created')
```

To the right of the code, a terminal window shows the command being run and its output:

```
bash - *p-172-31-29-135 ✘ Immediate Javascript (bro ✘ +)
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 create_security_group.py
Security Group sg-0e0fe09d642656bf3 has been created
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

A small note at the bottom right says 'AWS: (not connected)'.

## 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}')
```



The screenshot shows the hands-on.cloud IDE interface. On the left is a sidebar with a tree view of 'My-Python-Enviro' containing various Python files under categories like 'aws', 'functions', 'loops', etc. The main workspace has two tabs open: 'create\_security\_group.py' and 'list\_security\_groups.py'. The 'list\_security\_groups.py' tab contains the following code:

```
#!/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}')
```

Below the code editor is a terminal window titled 'bash - \*ip-172-31-29-135 x'. It shows the command `python3 list_security_groups.py` being run, and the output displays a list of security group IDs:

```
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 list_security_groups.py
Security Groups:
- Security Group sg-05ffad71ec1b899b5
- Security Group sg-0a1882cd57bf2ac1
- Security Group sg-0bde9150a6d4c8c02
- Security Group sg-0cef679a705a78be8
- Security Group sg-0df659bcb313c0b9a3
- Security Group sg-0e0fe09d642656bf3
- Security Group sg-0e9cf0acba7bc78c9
- Security Group sg-6dbc5f1b
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

A status bar at the bottom right indicates '13:1 Python Spaces: 4'.

## 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:

The screenshot shows the hands-on.cloud IDE interface. On the left, there's a sidebar with a search bar and a tree view of available Python scripts under 'My-Python-Enviro'. The main workspace contains a code editor with the following Python script:

```
#!/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(  
    GroupIds=[  
        SECURITY_GROUP_ID  
    ]  
)  
  
for security_group in security_groups:  
    print(f'Security Group {security_group.id} description: {security_group.description}')
```

To the right of the code editor is a terminal window titled 'bash - \*p-172-31-29-135 x'. It shows the command being run and its output:

```
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 filter_security_group_by_id.py  
Security Group sg-0e0fe09d642656bf3 description: Allow inbound SSH traffic  
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:

The screenshot shows a Jupyter Notebook environment. On the left is a sidebar with a tree view of files and notebooks. In the center is a code cell containing Python code to describe a security group. Below it is a terminal window showing the command being run and the resulting JSON output.

```
#!/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],
)
print(f'Security Group {SECURITY_GROUP_ID} attributes:')
for security_group in response['SecurityGroups']:
    print(json.dumps(security_group, indent=4))
```

```
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 describe_security_group.py
Security Group sg-0e0fe09d642656bf3 attributes:
{
    "Description": "Allow inbound SSH traffic",
    "GroupName": "allow-inbound-ssh",
    "IpPermissions": [
        {
            "FromPort": 22,
            "IpProtocol": "tcp",
            "IpRanges": [
                {
                    "CidrIp": "0.0.0.0/0"
                }
            ],
            "Ipv6Ranges": [],
            "PrefixListIds": [],
            "ToPort": 22,
            "UserIdGroupPairs": []
        }
    ],
    "OwnerId": "585584209241",
    "GroupId": "sg-0e0fe09d642656bf3",
    "IpPermissionsEgress": [
        {
    
```

## Deleting Security Groups

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



```
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:

The screenshot shows a Jupyter Notebook environment. On the left is a sidebar with various AWS-related notebooks listed. The main area shows a code cell containing the provided Python script. Below the code cell is an 'Immediate Javascript' cell showing the command run in a terminal and its output: 'Security Group sg-0e0fe09d642656bf3 has been deleted'. The status bar at the bottom right indicates '15:1 Python Spaces: 4'.

```
delete_security_group.x
1 #!/usr/bin/env python3
2
3 import json
4 import boto3
5
6 AWS_REGION = "us-east-2"
7 EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
8 SECURITY_GROUP_ID = 'sg-0e0fe09d642656bf3'
9
10 security_group = EC2_RESOURCE.SecurityGroup(SECURITY_GROUP_ID)
11
12 security_group.delete()
13
14 print(f'Security Group {SECURITY_GROUP_ID} has been deleted')
15
```

```
python3 -i ip-172-31-29-1x | Immediate Javascript (bro x) x
(.venv) admin:~/environment/python/working_with_ec2_instances $ .venv) admin:~/environment/python/working_with_ec2_instances $ python3 delete_security_group.py
Security Group sg-0e0fe09d642656bf3 has been deleted
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:



The screenshot shows a Jupyter Notebook environment. On the left, there's a sidebar with a tree view of files under the 'aws' directory, including 'loops', 'python\_syntax', 's3\_list\_objects', 'sets', 'string\_operations', 'switch\_case\_exmpl', 'tuples', 'working\_with\_amis', 'working\_with\_ec2\_instances', 'add\_tags.py', 'advanced\_monitor', 'attach\_security\_group.py', 'change\_type.py', 'create\_instance.py', 'create\_security\_group.py', 'delete\_security\_group.py', 'delete\_tags.py', 'describe\_all\_instances.py', 'describe\_security\_group.py', 'describe\_state.py', 'describe\_volumes.py', 'filter\_by\_instance\_id.py', 'filter\_by\_state.py', 'filter\_by\_tag.py', 'filter\_by\_type.py', 'filter\_security\_group.py', 'list\_all\_instances.py', 'list\_instance\_tags.py', 'list\_security\_group.py', 'reboot\_instance.py', 'start\_instance.py', 'stop\_instance.py', 'terminate\_instance.py', 'working\_with\_s3', 'working\_with\_snapshots', 'working\_with\_ssh\_keys', and 'working\_with\_volumes'. The main area contains a code cell with the following Python script:

```
3 import json
4 import boto3
5
6 AWS_REGION = "us-east-2"
7 EC2_RESOURCE = boto3.resource('ec2', region_name=AWS_REGION)
8 SECURITY_GROUP_ID = 'sg-084dfa143cc85a5cf'
9 INSTANCE_ID = 'i-04091b10d2cdc86aa'
10
11 instance = EC2_RESOURCE.Instance(INSTANCE_ID)
12
13 instance.modify_attribute(
14     Groups=[
15         SECURITY_GROUP_ID
16     ]
17 )
18
19 print(f'Security Group {SECURITY_GROUP_ID} has been attached to EC2 instance {INSTANCE_ID}')
20
```

Below the code cell is a terminal window titled 'python3 - ip-172-31-29-1x' showing the command being run and its output:

```
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 attach_security_groups.py
Security Group sg-084dfa143cc85a5cf has been attached to EC2 instance i-04091b10d2cdc86aa
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

AWS: (not connected)

## 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:

The screenshot shows a Jupyter Notebook interface. On the left, there's a sidebar with various Python scripts listed under the 'aws' category. The main area has two tabs: 'list\_instance\_security\_groups.py' and 'Immediate (Javascript (bro...))'. The code in 'list\_instance\_security\_groups.py' is as follows:

```
#!/usr/bin/env python3
import json
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"]}'')
```

The 'Immediate (Javascript (bro...))' tab shows the execution of the script:

```
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 list_instance_security_groups.py
Instance i-04091b10d2cdc86aa Security Groups:
- Security Group sg-084dfa143cc85a5cf
- Security Group sg-6dbc5f1b
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## Detaching Security Group from the EC2 Instance

There's no specific API call to detach the Security Group from the EC2 instance. To detach a Security 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 override 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:

```
File Edit Find View Go Run Tools Window Support Preview Run
Go Anything (⌘ P) detach_security_group x
aws
python3 - *ip-172-31-29-1x Immediate (Javascript (bro x
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 detach_security_group.py
Security Group sg-bdbc5f1b has been detached from the instance i-04091b10d2cd86aa
(.venv) admin:~/environment/python/working_with_ec2_instances $
```



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:



The terminal window shows the following Python code:

```
3 import boto3
4
5 AWS_REGION = "us-east-2"
6 EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
7
8 allocation = EC2_CLIENT.allocate_address(
9     Domain='vpc',
10    TagSpecifications=[{
11        'ResourceType': 'elastic-ip',
12        'Tags': [
13            {
14                'Key': 'Name',
15                'Value': 'my-elastic-ip'
16            },
17        ],
18    }],
19 )
20
21 ]
22
23 print(f'Allocation ID {allocation["AllocationId"]}')
24 print(f' - Elastic IP {allocation["PublicIp"]} has been allocated')
25
```

Below the code, a command-line session is shown:

```
python3 -i ip-172-31-29-1x | Immediate (Javascript (bro ×) ⊞)
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 allocate_eip.py
Allocation ID eipalloc-05b6e9/a44bee455d
- Elastic IP 3.132.220.73 has been allocated
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

AWS: (not connected)

## 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:

```
File Edit Find View Go Run Tools Window Support Preview Run Share
Go Anything (⌘P)
describe_eips.py
1 #!/usr/bin/env python3
2
3 import json
4 import boto3
5
6 AWS_REGION = "us-east-2"
7 EC2_CLIENT = boto3.client('ec2', region_name=AWS_REGION)
8
9 response = EC2_CLIENT.describe_addresses(
10     Filters=[
11         {
12             'Name': 'tag:Name',
13             'Values': ['my-elastic-ip']
14         }
15     ]
16 )
17
18 print('EIP attributes:')
19
20 for address in response['Addresses']:
21     print(json.dumps(address, indent=4))
22 |
```

```
bash - *ip-172-31-29-135 ✘ Immediate (Javascript (bro) ✘
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 describe_eips.py
EIP attributes:
{
    "PublicIp": "3.132.220.73",
    "AllocationId": "eipalloc-05b6e97a44bee455d",
    "Domain": "vpc",
    "Tags": [
        {
            "Key": "Name",
            "Value": "my-elastic-ip"
        }
    ],
    "PublicIpv4Pool": "amazon",
    "NetworkBorderGroup": "us-east-2"
}
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

## 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:



The screenshot shows a terminal window with the following content:

```
python3 -i p-172-31-29-1x | Immediate Javascript (bro x ⌂
(.venv) admin:~/environment/python/working_with_ec2_instances $ (.venv) admin:~/environment/python/working_with_ec2_instances $ python3 associate_eip.py
EIP 3.132.220.73 associated with the instance i-04091b10d2cdc86aa
(.venv) admin:~/environment/python/working_with_ec2_instances $
```

The terminal is running a Python script named `associate_eip.py`. The script uses the `boto3` library to interact with the AWS EC2 service. It filters for instances with a specific tag (`my-elastic-ip`) and associates a public IP address with the first instance found.

AWS: (not connected)

## 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:

The screenshot shows a Jupyter Notebook interface with two panes. The left pane displays a file tree for a directory named 'working\_with\_ec2\_instances'. The right pane contains a code editor and a terminal window.

**Code Editor Content:**

```
#!/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-04091b10d2cdcb6aa'
instance = EC2_RESOURCE.Instance(INSTANCE_ID)
response = EC2_CLIENT.describe_addresses(
    Filters=[
        {
            '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}')
```

**Terminal Output:**

```
python3 -i ip-172-31-29-1x | Immediate (Javascript (bro x) +)
(.venv) admin:~/environment/python/working_with_ec2_instances $
(.venv) admin:~/environment/python/working_with_ec2_instances $ python3 disassociate_eip.py
EIP 3.132.220.73 diassociated from the instance i-04091b10d2cdcb6aa
(.venv) admin:~/environment/python/working_with_ec2_instances $
```



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:



The screenshot shows a Jupyter Notebook environment. On the left, there's a sidebar with a tree view of files under the 'aws' directory, including various Python scripts for managing EC2 instances, security groups, and EIP addresses. The main area contains a code cell with the following Python script:

```
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')
```

Below the code cell is a terminal window titled 'python3 - \*ip-172-31-29-1x'. It shows the command being run: 'python3 release\_eip.py'. The output of the command is: 'EIP 3.132.220.73 has been released'. The terminal window has a red border around the command line.

AWS: (not connected)

## 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 \[Example\]](#)

How useful was this post?

Click on a star to rate it!