Ec2
https://boto3.amazonaws.com/v1/documentation/api/latest/guide/examples.html
https://www.botmetric.com/blog/aws-cloud-automation-python-boto3-scripts/
https://aws.amazon.com/blogs/storage/using-boto3-to-replicate-amazon-s3-buckets-at-scale/

class **`EC2.Client`**

```
import boto3
client = boto3.client('ec2')
```

Create an instance

```
import boto3
client = boto3.client('ec2')
resp =
client.run_instances(ImageId='ami-0ed9277fb7eb570c9',InstanceType='t2.micro',MaxCount=1,MinCount=1)
for instance in resp['Instances']:
    print(instance['InstanceId'])
```

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ec2.html#EC2.Client.run_instances

Create instance and tag them

```
import boto3
client = boto3.client('ec2')
resp =
client.run_instances(ImageId='ami-0ed9277fb7eb570c9',InstanceType='t2.micro',MaxCount=3,MinCount=3)
for instance in resp['Instances']:
    response = client.create_tags(
        Resources=[instance['InstanceId']],
        Tags=[
        {
            'Key': 'backup',
            'Value': 'true'
        },
    ])
    print(instance['InstanceId'])

import boto3
client = boto3.client('ec2')
```

[redacted]

start/stop/terminate
Each of these methods take instanceids as arguments.

```
response = client.start_instances(
    InstanceIds=[
        'string',
    ],
    AdditionalInfo='string',
    DryRun=True|False

)
```

—---------------------------

```
response = client.stop_instances(
    InstanceIds=[
        'string',
    ],
    Hibernate=True|False,
    DryRun=True|False,
    Force=True|False

)
```

—--------------------------------

```
response = client.terminate_instances(
    InstanceIds=[
        'string',
    ],
    DryRun=True|False

)
```

- **describe_instances()**

- **describe_instances()**

```
response = client.describe_instances(
    Filters=[
        {
            'Name': 'string',
            'Values': [
                'string',
            ]
        },
    ],
    InstanceIds=[
        'string',
    ],
    DryRun=True|False,
    MaxResults=123,
    NextToken='string'

)
```

If you specify instance IDs, the output includes information for only the specified instances. If you specify filters, the output includes information for only those instances that meet the filter criteria. If you do not specify instance IDs or filters, the output includes information for all instances, which can affect performance. We recommend that you use pagination to ensure that the operation returns quickly and successfully.

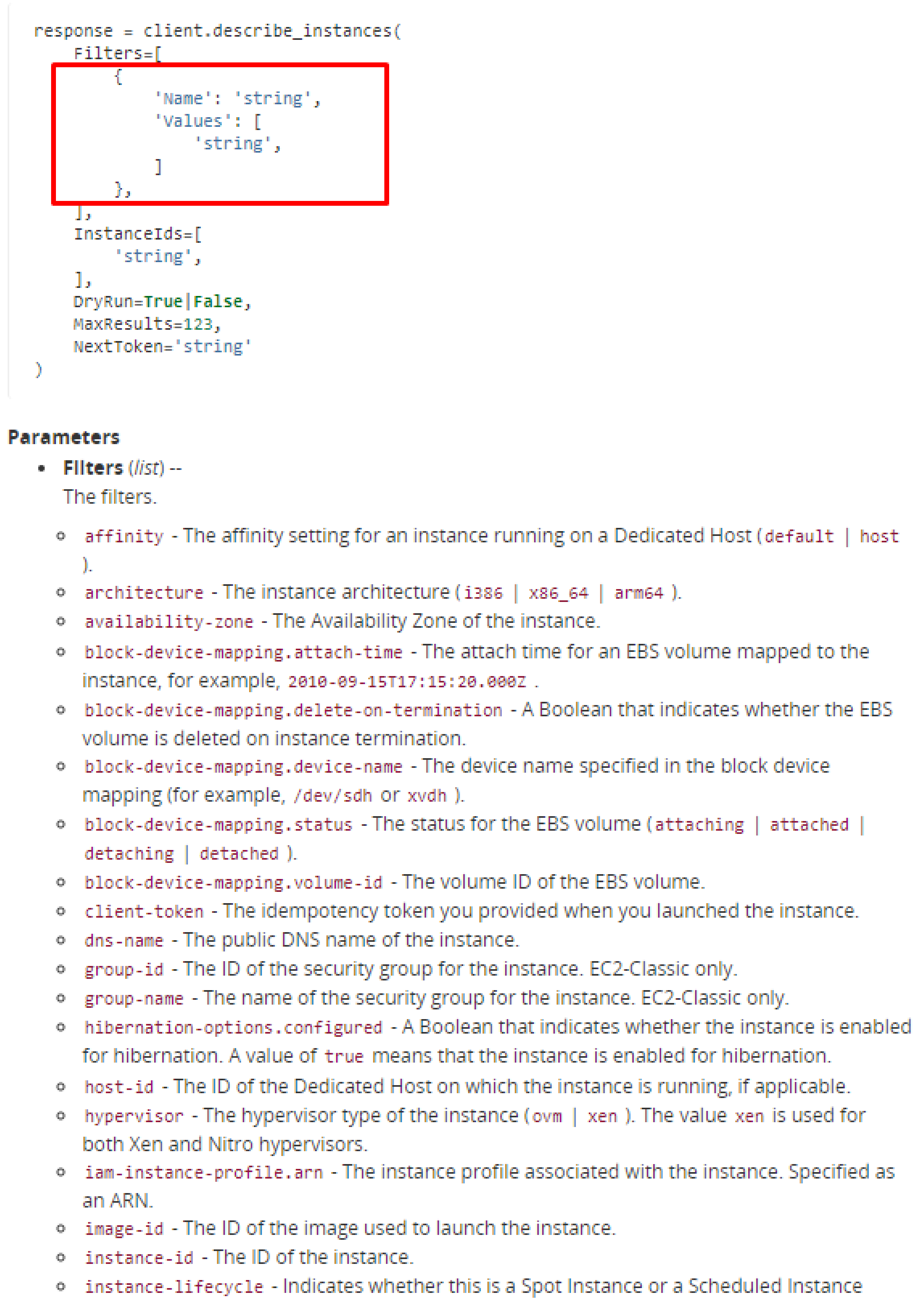
```
import boto3
client = boto3.client('ec2')
res = client.describe_instances()
#res = res['Reservations']
for instance in res['Reservations']:
    for instance in instance['Instances']:
        print(f"instance ids is {intance1['InstanceId']}")
        print(f"InstanceType is {intance1['InstanceType']}")
```

Filter instances

```
import boto3
client = boto3.client('ec2')
res = client.describe_instances(Filters=[{
           'Name': 'instance-state-name',
           'Values': [
             'terminated'
            ]
    }])

for instance in res['Reservations']:
    for instance in instance['Instances']:
        print(f"instance id is {intance1['InstanceId']}")
```

	`print(f"InstanceType is {intance1['InstanceType']}")`



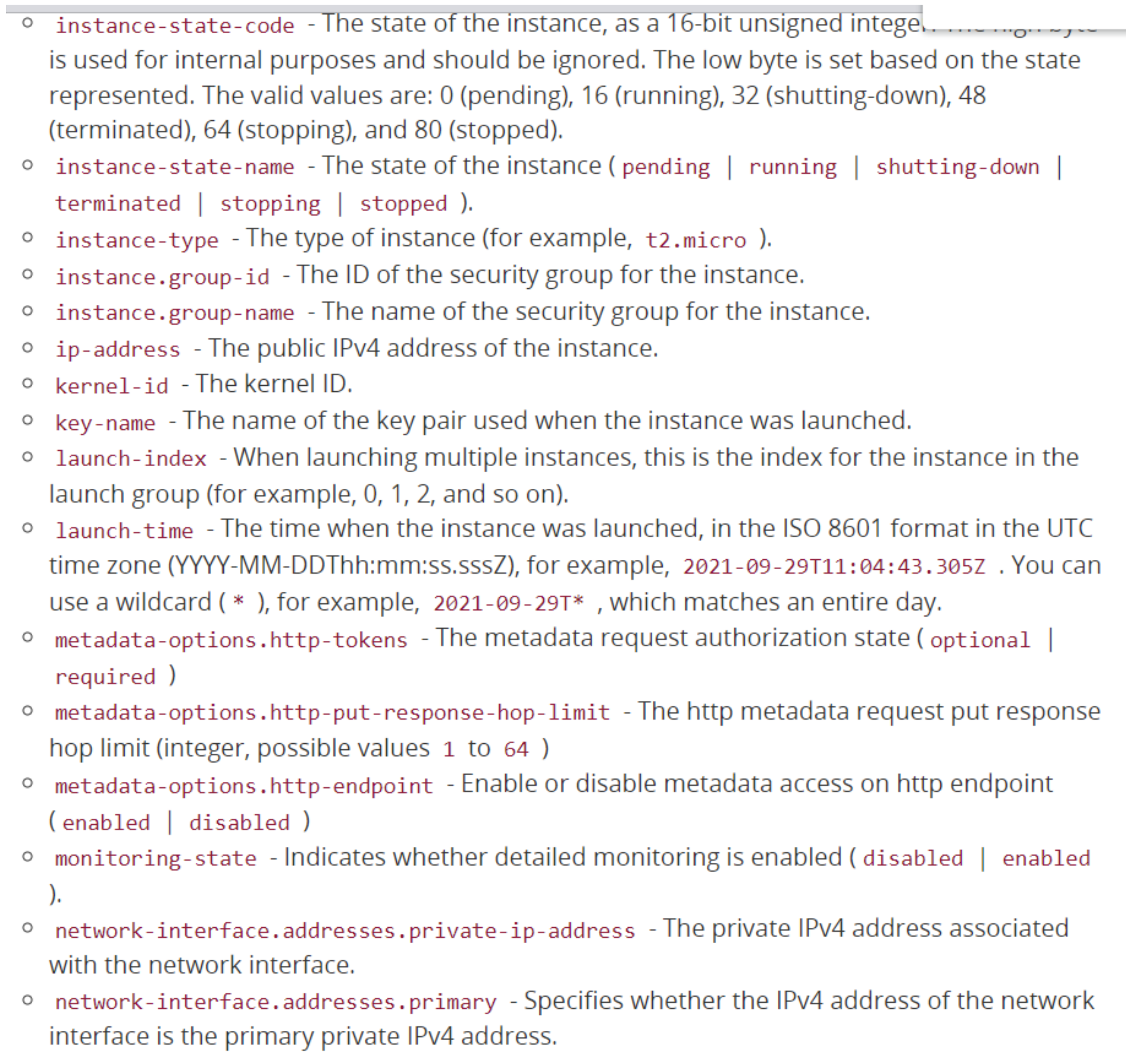
```
response = client.describe_instances(
    Filters=[
        {
            'Name': 'string',
            'Values': [
                'string',
            ]
        },
    ],
    InstanceIds=[
        'string',
    ],
    DryRun=True|False,
    MaxResults=123,
    NextToken='string'
)
```

Parameters

- **Filters** (*list*) --

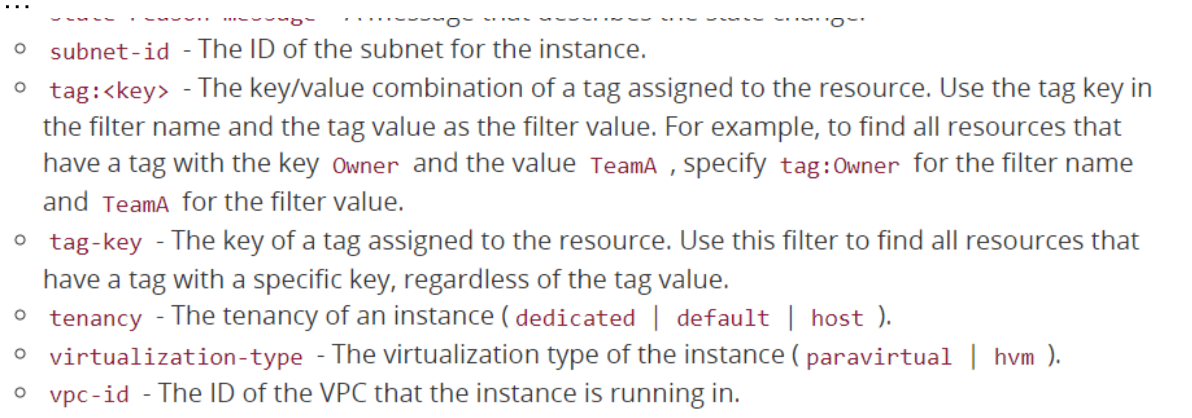
  The filters.

  - `affinity` - The affinity setting for an instance running on a Dedicated Host (`default` | `host`).
  - `architecture` - The instance architecture (`i386` | `x86_64` | `arm64`).
  - `availability-zone` - The Availability Zone of the instance.
  - `block-device-mapping.attach-time` - The attach time for an EBS volume mapped to the instance, for example, `2010-09-15T17:15:20.000Z` .
  - `block-device-mapping.delete-on-termination` - A Boolean that indicates whether the EBS volume is deleted on instance termination.
  - `block-device-mapping.device-name` - The device name specified in the block device mapping (for example, `/dev/sdh` or `xvdh`).
  - `block-device-mapping.status` - The status for the EBS volume (`attaching` | `attached` | `detaching` | `detached`).
  - `block-device-mapping.volume-id` - The volume ID of the EBS volume.
  - `client-token` - The idempotency token you provided when you launched the instance.
  - `dns-name` - The public DNS name of the instance.
  - `group-id` - The ID of the security group for the instance. EC2-Classic only.
  - `group-name` - The name of the security group for the instance. EC2-Classic only.
  - `hibernation-options.configured` - A Boolean that indicates whether the instance is enabled for hibernation. A value of `true` means that the instance is enabled for hibernation.
  - `host-id` - The ID of the Dedicated Host on which the instance is running, if applicable.
  - `hypervisor` - The hypervisor type of the instance (`ovm` | `xen`). The value `xen` is used for both Xen and Nitro hypervisors.
  - `iam-instance-profile.arn` - The instance profile associated with the instance. Specified as an ARN.
  - `image-id` - The ID of the image used to launch the instance.
  - `instance-id` - The ID of the instance.
  - `instance-lifecycle` - Indicates whether this is a Spot Instance or a Scheduled Instance



- `instance-state-code` - The state of the instance, as a 16-bit unsigned intege
is used for internal purposes and should be ignored. The low byte is set based on the state represented. The valid values are: 0 (pending), 16 (running), 32 (shutting-down), 48 (terminated), 64 (stopping), and 80 (stopped).
- `instance-state-name` - The state of the instance (`pending` | `running` | `shutting-down` | `terminated` | `stopping` | `stopped`).
- `instance-type` - The type of instance (for example, `t2.micro`).
- `instance.group-id` - The ID of the security group for the instance.
- `instance.group-name` - The name of the security group for the instance.
- `ip-address` - The public IPv4 address of the instance.
- `kernel-id` - The kernel ID.
- `key-name` - The name of the key pair used when the instance was launched.
- `launch-index` - When launching multiple instances, this is the index for the instance in the launch group (for example, 0, 1, 2, and so on).
- `launch-time` - The time when the instance was launched, in the ISO 8601 format in the UTC time zone (YYYY-MM-DDThh:mm:ss.sssZ), for example, `2021-09-29T11:04:43.305Z` . You can use a wildcard (`*`), for example, `2021-09-29T*` , which matches an entire day.
- `metadata-options.http-tokens` - The metadata request authorization state (`optional` | `required`)
- `metadata-options.http-put-response-hop-limit` - The http metadata request put response hop limit (integer, possible values `1` to `64`)
- `metadata-options.http-endpoint` - Enable or disable metadata access on http endpoint (`enabled` | `disabled`)
- `monitoring-state` - Indicates whether detailed monitoring is enabled (`disabled` | `enabled`).
- `network-interface.addresses.private-ip-address` - The private IPv4 address associated with the network interface.
- `network-interface.addresses.primary` - Specifies whether the IPv4 address of the network interface is the primary private IPv4 address.

…



- `subnet-id` - The ID of the subnet for the instance.
- `tag:<key>` - The key/value combination of a tag assigned to the resource. Use the tag key in the filter name and the tag value as the filter value. For example, to find all resources that have a tag with the key `Owner` and the value `TeamA` , specify `tag:Owner` for the filter name and `TeamA` for the filter value.
- `tag-key` - The key of a tag assigned to the resource. Use this filter to find all resources that have a tag with a specific key, regardless of the tag value.
- `tenancy` - The tenancy of an instance (`dedicated` | `default` | `host`).
- `virtualization-type` - The virtualization type of the instance (`paravirtual` | `hvm`).
- `vpc-id` - The ID of the VPC that the instance is running in.

Filter `instance` based on tags

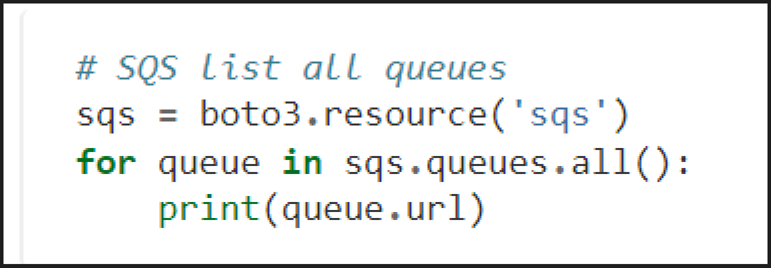
```python
import boto3
client = boto3.client('ec2')
res = client.describe_instances(Filters=[{
            'Name': 'tag:env',
            'Values': [
              'prod'
             ]
    }])

for instance in res['Reservations']:
    for instance in instance['Instances']:
        print(f"instance id is {intance1['InstanceId']}")
        print(f"InstanceType is {intance1['InstanceType']}")
```

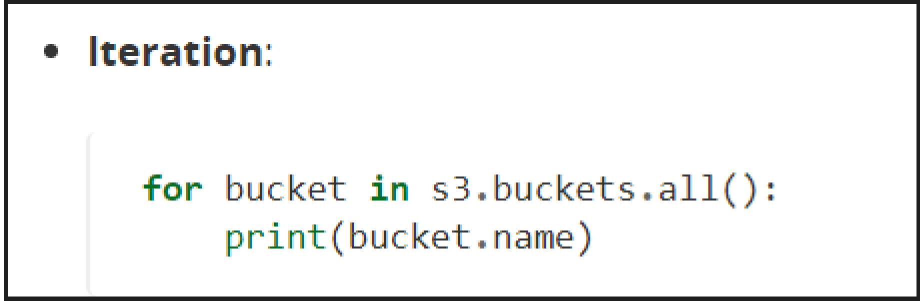
Collections

A collection provides an iterable interface to a group of resources.

A collection seamlessly handles pagination for you, making it possible to easily iterate over all items from all pages of data. Example of a collection:



```python
# SQS list all queues
sqs = boto3.resource('sqs')
for queue in sqs.queues.all():
    print(queue.url)
```



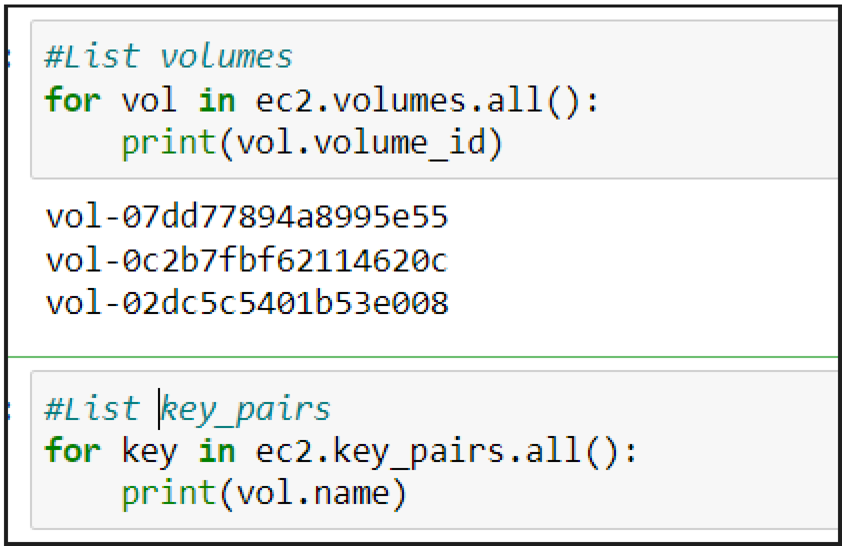
- **Iteration**:

```python
for bucket in s3.buckets.all():
    print(bucket.name)
```

EC2	## Service Resource
	
	A resource representing Amazon Elastic Compute Cloud (EC2):
	
	`import boto3`
	
	`ec2 = boto3.resource('ec2')`
	
	These are the resource's available actions:
	
	• `create_instances()`
	• `create_internet_gateway()`
	• `create_key_pair()`
	• `create_route_table()`
	• `create_security_group()`
	• `create_snapshot()`
	• `create_subnet()`
	• `create_tags()`
	• `create_volume()`
	• `create_vpc()`
	
	These are the resource's available collections:
	
	• `classic_addresses`
	• `dhcp_options_sets`
	• `images`
	• `instances`
	• `internet_gateways`
	• `key_pairs`
	• `network_acls`
	• `network_interfaces`
	• `placement_groups`
	• `route_tables`

	• security_groups • snapshots • subnets • volumes • vpc_addresses • vpc_peering_connections • vpcs

Ec2 collections	`import boto3`
	`ec2 = boto3.resource('ec2')`
	
	`for instance in ec2.instances.all():`
	`    print(instance.instance_id)`

List all key pairs, volumes using ec2 collections	
	`#List volumes`
	`for vol in ec2.volumes.all():`
	`    print(vol.volume_id)`
	
	`vol-07dd77894a8995e55`
	`vol-0c2b7fbf62114620c`
	`vol-02dc5c5401b53e008`
	
	`#List key_pairs`
	`for key in ec2.key_pairs.all():`
	`    print(vol.name)`
	
	`Similarly you can list network lists, security groups and basically all items available under ec2 available collections.`

Ec2 collections with filters

```python
import boto3
ec2 = boto3.resource('ec2')

for instance in ec2.instances.filter(Filters=[{
    'Name': 'availability-zone',
    'Values': ['us-east-1b']
}]):

    print(instance.instance_id)
```

Stop all instances matching filter

```python
import boto3
ec2 = boto3.resource('ec2')
ec2.instances.filter(Filters=[{'Name': 'availability-zone','Values': ['us-east-1b']}]).stop()
```

Create snapshot for ec2 instances with matching tags

```python
import boto3
ec2 = boto3.resource('ec2')
backup_filter=[{'Name': 'tag:backup','Values': ['true']}]
sns_client = boto3.client('sns')

snapshot_ids =[]
for instance in ec2.instances.filter(Filters=backup_filter):
       print(instance)
#instance object has volumes as collections
       for vol in instance.volumes.all():
           print(vol)
           snapshot_id = vol.create_snapshot(Description='created by boto3')
           snapshot_ids.append(snapshot_id)
print(snapshot_ids)
response = sns_client.publish(
    TopicArn='arn:aws:sns:us-east-1:534173283575:vprofile-pipe',
    Message='Boto3'+ snapshot_ids.,
    Subject='Boto3 snapshots taken',
)
```


Delete snapshot matching tags

```
import boto3
ec2 = boto3.resource('ec2')
backup_filter=[{'Name': 'tag:backup','Values': ['true']}]
for instance in ec2.instances.filter(Filters=[{'Name':
'tag:backup','Values': ['true']}]):
      print(intance.instance_id)
      for vol in instance.volumes.all():
          print(vol.volume_id)
#Volumes has snapshots as collections
          for snap in vol.snapshots.all():
              print(snap.snapshot_id)
              snap.delete()
```

Delete unused and untagged EBS volumes

```
```

Diff between client and resource object

Client and **Resource** are two different abstractions within the boto3 SDK for making AWS service requests. If you want to make API calls to an AWS service with boto3, then you do so via a Client or a Resource.

Client:

- this is the original boto3 API abstraction
- it provides low-level AWS service access
- all AWS service operations are supported by clients

Resource:

- this is the newer boto3 API abstraction
- it provides high-level, object-oriented API
- it does not provide 100% API coverage of AWS services

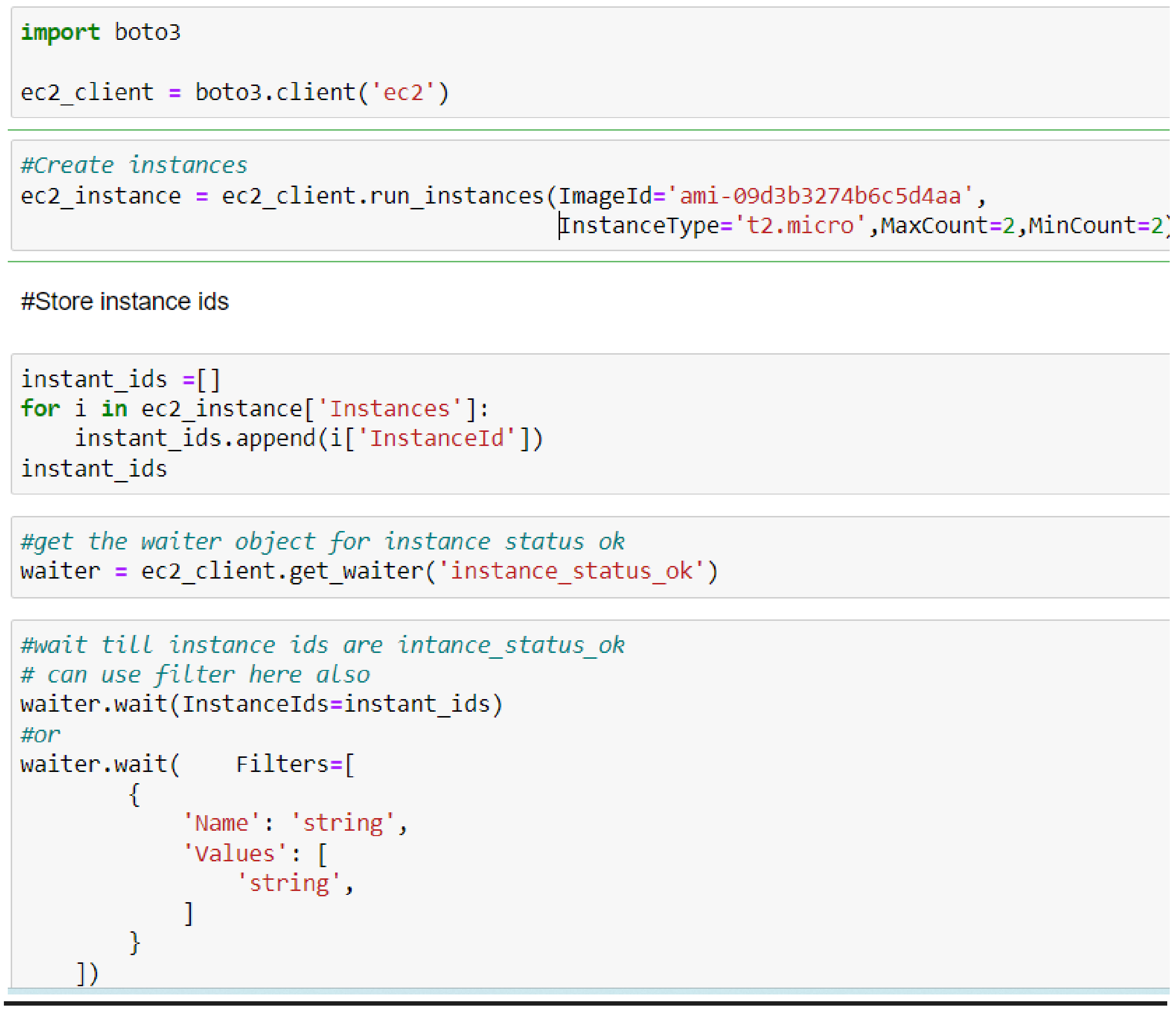
```
```

waiters

A number of requests in AWS using boto3 are not instant. Common examples of boto3 requests are deploying a new server or RDS instance. For some long running requests, we are ok to initiate the request and then check for completion at some later time. But in many cases, we want to wait for the request to complete before we move on to the subsequent parts of the script that may rely on a long running process to have been completed. One example would be a script that might copy an

	AMI to another account by sharing all the snapshots. After sharing the snapshots to the other account, you would need to wait for the local snapshot copies to complete before registering the AMI in the receiving account. [redacted] The available waiters are: • `EC2.Waiter.ImageAvailable` • `EC2.Waiter.ImageExists` • `EC2.Waiter.InstanceExists` • `EC2.Waiter.InstanceRunning` • `EC2.Waiter.InstanceStatusOk` • `EC2.Waiter.InstanceStopped` • `EC2.Waiter.InstanceTerminated` [redacted]

Create an instance and wait till Instance Status Ok



```python
import boto3

ec2_client = boto3.client('ec2')
```

```python
#Create instances
ec2_instance = ec2_client.run_instances(ImageId='ami-09d3b3274b6c5d4aa',
                                        InstanceType='t2.micro',MaxCount=2,MinCount=2)
```

#Store instance ids

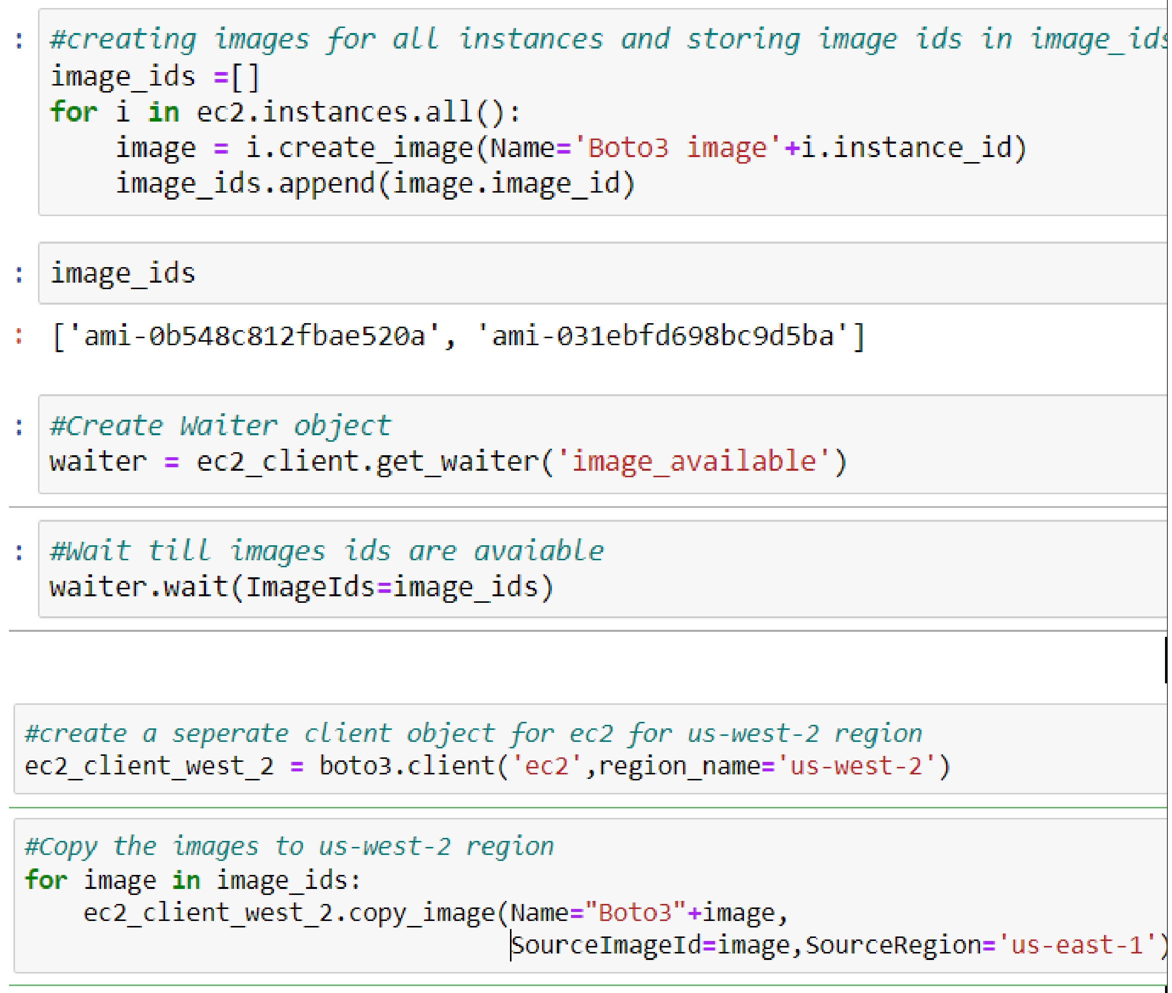
```python
instant_ids =[]
for i in ec2_instance['Instances']:
    instant_ids.append(i['InstanceId'])
instant_ids
```

```python
#get the waiter object for instance status ok
waiter = ec2_client.get_waiter('instance_status_ok')
```

```python
#wait till instance ids are intance_status_ok
# can use filter here also
waiter.wait(InstanceIds=instant_ids)
#or
waiter.wait(    Filters=[
        {
            'Name': 'string',
            'Values': [
                'string',
            ]
        }
    ])
```

Create an Image of instance and wait till it is available.
Also copy images to another region.

Note the creation of ec2 client objects by passing the region name.



```
: #creating images for all instances and storing image ids in image_ids
  image_ids =[]
  for i in ec2.instances.all():
      image = i.create_image(Name='Boto3 image'+i.instance_id)
      image_ids.append(image.image_id)

: image_ids

: ['ami-0b548c812fbae520a', 'ami-031ebfd698bc9d5ba']

: #Create Waiter object
  waiter = ec2_client.get_waiter('image_available')

: #Wait till images ids are avaiable
  waiter.wait(ImageIds=image_ids)


#create a seperate client object for ec2 for us-west-2 region
ec2_client_west_2 = boto3.client('ec2',region_name='us-west-2')

#Copy the images to us-west-2 region
for image in image_ids:
    ec2_client_west_2.copy_image(Name="Boto3"+image,
                                 SourceImageId=image,SourceRegion='us-east-1')
```

paginator

- **IAM.Paginator.ListUsers**

```
paginator = client.get_paginator('list_users')

page_iterator = paginator.paginate()

count=1

for page in page_iterator:

    for user in page['Users']:
```

	`        print(count , " ", user['UserName'])`
	
	`        count+=1`

List all snapshots owned by account
 	
Search “ec2.snapshots.filter” in ec2 boto3
	`import boto3`
	`ec2 = boto3.resource('ec2')`
	
	`owned_by_me = [{'Name': 'owner-id','Values': ['534173283575']}]`
	`for snap in ec2.snapshots.filter(Filters=owned_by_me):`
	`    print(snap)`
	`#     print(snap.owner_id)`
	`  #  if snap.owner_id=='534173283575':`
	`  #      print(snap)`