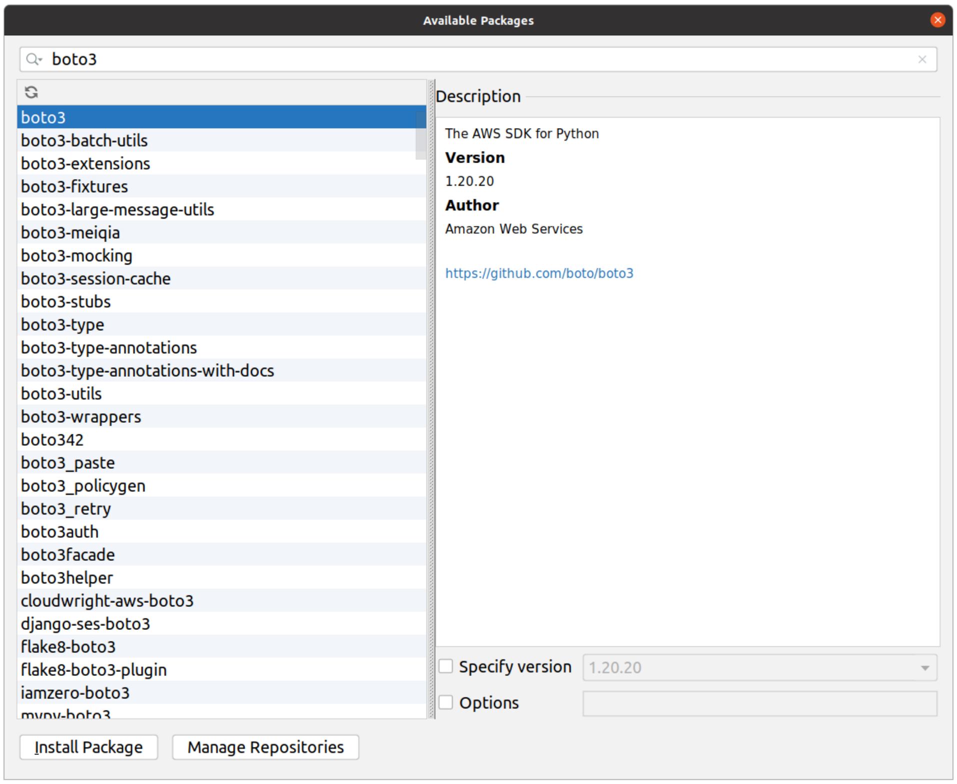
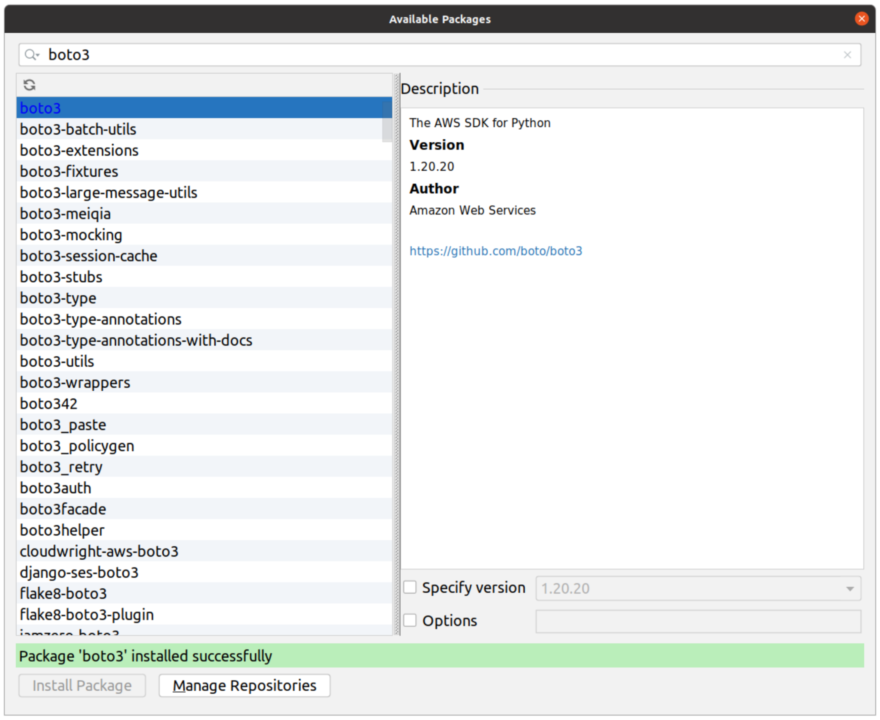
Search for **boto3** and hit **Install Package**.



The AWS SDK for Python
Version
1.20.20
Author
Amazon Web Services
https://github.com/boto/boto3



The AWS SDK for Python
Version
1.20.20
Author
Amazon Web Services
https://github.com/boto/boto3

Package 'boto3' installed successfully

Configuring the Credentials

There are two ways to pass the credentials:

- Shared credentials file ~ /.aws/credentials
- Passing credentials to boto3

In this book, as it is Python 3 oriented, I will pass the credentials to boto3.
I will not be using the /home/user/.aws/credentials file.

Some of the examples on Amazon documentation show the Access Key and the Secret Key enclosed by <>, for example:

```
<AKIATRML7HMNS6SQ5WOS>
```

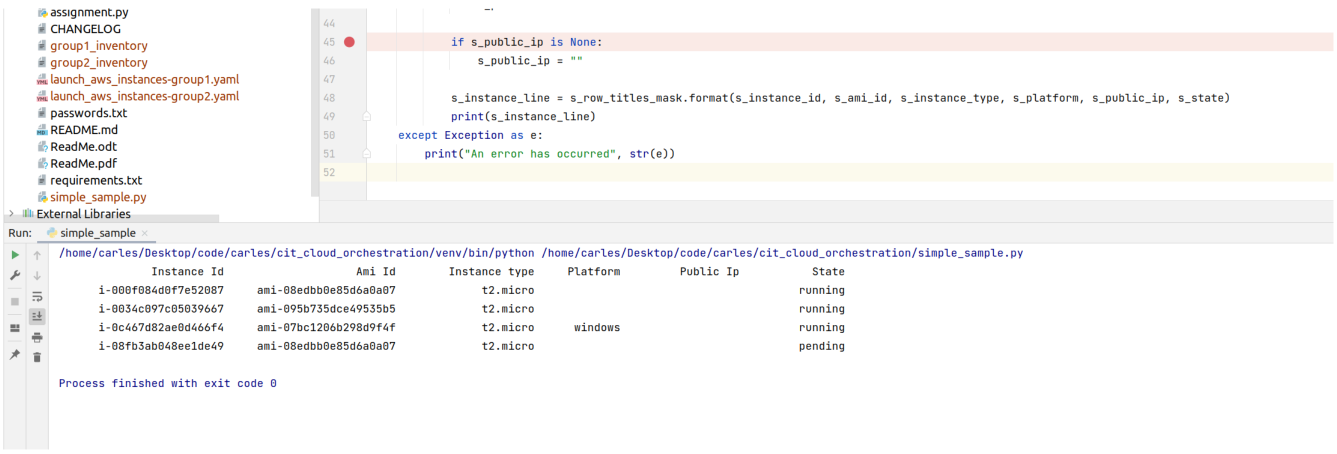
This does not mean that you have to add <> surrounding your Access Key.
It's just an unfortunate way to indicate that you have to just put what's between the <>.

Our first Python 3 sample

```
import boto3
from botocore.config import Config

if __name__ == "__main__":

    # s_region = "us-west-1"
    s_region = "eu-west-1"
    s_access_key = "AKIATRML7HMNS6SQ5WOS"
    s_secret_key = "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"

    try:
        o_config = Config(
            region_name=s_region,
            signature_version="v4",
            retries={
                'max_attempts': 10,
                'mode': 'standard'
            }
        )

        ec2_res = boto3.resource("ec2",
                                 config=o_config,
                                 aws_access_key_id=s_access_key,
                                 aws_secret_access_key=s_secret_key
                                 )

        s_row_titles_mask = "{:>25} {:>25} {:>20} {:>12} {:>17} {:>15}"
        print(s_row_titles_mask.format("Instance Id", "Ami Id", "Instance type", "Platform", "Public
Ip", "State"))
```

```
        for o_instance in ec2_res.instances.all():

            s_instance_id = o_instance.id
            s_ami_id = o_instance.image.id
            s_instance_type = o_instance.instance_type
            s_platform = o_instance.platform
            s_public_ip = o_instance.public_ip_address
            # o_instance.state is a dict like this {'Code': 16, 'Name': 'running'}
            s_state = o_instance.state['Name']

            if s_instance_type is None:
                s_instance_type = ""

            if s_platform is None:
                s_platform = ""

            if s_public_ip is None:
                s_public_ip = ""

            s_instance_line = s_row_titles_mask.format(s_instance_id, s_ami_id, s_instance_type,
s_platform, s_public_ip, s_state)
            print(s_instance_line)
    except Exception as e:
        print("An error has occurred", str(e))
```

This code will produce this result for my actual view of Instances:



```
            if s_public_ip is None:
                s_public_ip = ""

            s_instance_line = s_row_titles_mask.format(s_instance_id, s_ami_id, s_instance_type, s_platform, s_public_ip, s_state)
            print(s_instance_line)
    except Exception as e:
        print("An error has occurred", str(e))
```

```
/home/carles/Desktop/code/carles/cit_cloud_orchestration/venv/bin/python /home/carles/Desktop/code/carles/cit_cloud_orchestration/simple_sample.py
             Instance Id                    Ami Id       Instance type     Platform        Public Ip          State
     i-000f084d0f7e52087     ami-08edbb0e85d6a0a07             t2.micro                                     running
     i-0034c097c05039667     ami-095b735dce49535b5             t2.micro                                     running
     i-0c467d82ae0d466f4     ami-07bc1206b298d9f4f             t2.micro      windows                        running
     i-08fb3ab048ee1de49     ami-08edbb0e85d6a0a07             t2.micro                                     pending

Process finished with exit code 0
```

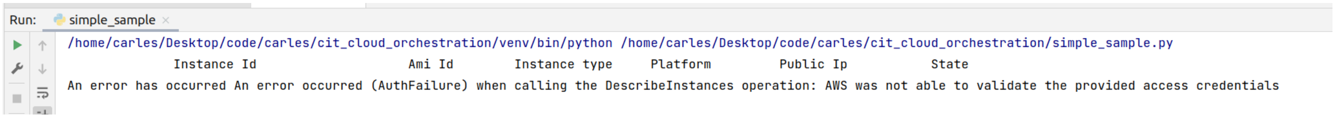
Please not how these Instances do not have a Public Ip.
This is because in my config, they belong to an VPC which doesn't provide a Public Ip automatically.

Error handling

Amazon SDK for Python 3, boto3, raises any error as an Exception.
For example, if your credentials are disabled or not valid, boto3 will raise this Exception:

```
botocore.exceptions.ClientError: An error occurred (AuthFailure) when calling the
DescribeInstances operation: AWS was not able to validate the provided access
credentials
```

As our sample code is catching the exceptions, an error would be catch and gracefully displayed as:



```
/home/carles/Desktop/code/carles/cit_cloud_orchestration/venv/bin/python /home/carles/Desktop/code/carles/cit_cloud_orchestration/simple_sample.py
              Instance Id                    Ami Id        Instance type     Platform         Public Ip           State
An error has occurred An error occurred (AuthFailure) when calling the DescribeInstances operation: AWS was not able to validate the provided access credentials
```

boto3 and None Objects

Amazon uses Java a lot. I guess that's why they like to return None objects when they don't have information for an Object. Other companies would just return an empty String.

For example for the Public Ip when there is none, or for the VPC Id.
This is particularly important when an Instance is Terminated, as some of the information is no longer available.

That's why in the sample code I check for Nones and I convert to an empty String.

```
if s_public_ip is None:
    s_public_ip = ""
```

If I won't do it, when trying to format the String for printing it nicely, it would return an error as None object doesn't know about formatting.

Other structures can have data or not, like Tags.
Do not assume that every instance will have Tags.

```
        self.o_awsdynamodb = o_awsdynamodb

        # Default values
        self.b_admin = False
        self.s_access_key = ""
        self.s_secret_key = ""

        # Menu as: title, is admin, Class, method
        self.a_menu_main = [("Create a user", True, self.o_usermanagement, "create_user"),
                            ("EC2 Instances", False, self.o_awsec2, "display_menu"),
                            ("EBS Storage", False, self.o_awsebs, "display_menu"),
                            ("S3 Storage", True, self.o_awss3, "display_menu"),
                            ("Cloud Watch", False, self.o_awscw, "display_menu"),
                            ("Amazon DynamoDB (information and samples)", False, self.o_awsdynamodb,
"display_menu")
                            ]

    def ask_for_username(self):
        s_username = input("Username:")

        return s_username

    def ask_for_password(self):
        s_password = input("Password:")

        return s_password

    def main(self):
        try:
            while True:
                s_role = ""
                b_admin = False

                s_username = self.ask_for_username()

                if s_username == "":
                    exit()

                s_password = self.ask_for_password()

                s_role, s_access_key, s_secret_key =
self.o_fileutils.get_credentials(self.s_credentials_file, s_username, s_password)
                if s_role == "ADMIN":
                    b_admin = True
                    self.b_admin = b_admin

                if s_access_key != "" and s_secret_key != "":
                    # Successful login
                    # Update credentials on the injected objects
                    self.o_credentials.b_admin = b_admin
                    self.o_credentials.s_access_key = s_access_key
                    self.o_credentials.s_secret_key = s_secret_key
                    self.o_awsec2.update_credentials()
                    self.o_awsebs.update_credentials()
                    self.o_awss3.update_credentials()
```