**Python Flask REST API in AWS SAM**

**What is Serverless:**

Build and run applications without thinking about servers.

Serverless is the native architecture of the cloud that enables you to shift more of your operational responsibilities to AWS, increasing your agility and innovation. Serverless allows you to build and run applications and services without thinking about servers. It eliminates infrastructure management tasks such as server or cluster provisioning, patching, operating system maintenance, and capacity provisioning. You can build them for nearly any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.

**SAM (Serverless Application Model):**

The introduction of serverless technology has enabled developers to shed the burden of managing infrastructure and concentrate on their application code. AWS Lambda has taken on that management by providing isolated, event-driven compute environments for the execution of application code. To use a Lambda function, a developer just needs to package their code and any dependencies into a zip file and upload that file to AWS. However, as serverless applications get larger and more functions are required for those applications, there is a need for the ability to share code across multiple functions within the application.

To meet this need, AWS released Lambda layers, providing a mechanism to externally package dependencies that can be shared across multiple Lambda functions. Lambda layers reduces lines of code and size of application artefacts and simplifies dependency management. Along with the release of Lambda layers, AWS also released support for layers in the AWS Serverless Application Model (SAM) and the AWS SAM command line interface (CLI). SAM is a template specification that enables developers to define a serverless application in clean and simple syntax. The SAM CLI is a command line tool that operates on SAM templates and application code. SAM can now define Lambda layers with the AWS::Serverless::LayerVersion type. The SAM CLI can build and test your layers locally as well as package, deploy, and publish your layers for public consumption.

**AWS Lambda:**

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume.

With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app

**Amazon API Gateway:**

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services. Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.

**Use case:**

Create a REST API with Python Flask in AWS SAM

**IDE Used:** PyCharm

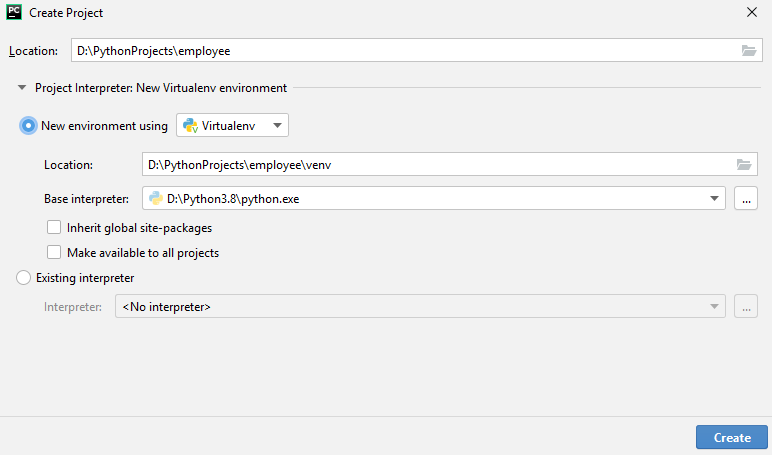
**Python Version :**3.8

**Step 1:** Install Python and set the python path in system

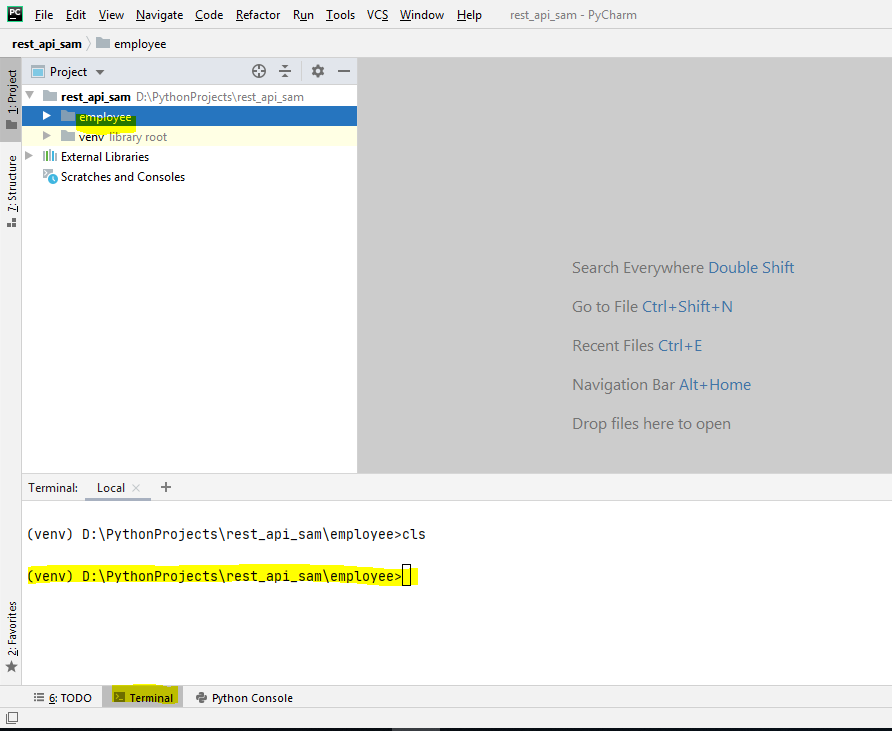
**Step 2:** Install PyCharm

**Step 3:** Create a new project using PyCharm with Virtual Environment. So that we can isolate our installation within virtual environment.

**Note:** *A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated python virtual environments for them.*



**Step 4:** Go to the terminal window of PyCharm.



**Step 5:** Run below two command to install AWS SAM CLI and AWS CLI

* pip install aws-sam-cli
* pip install awscli

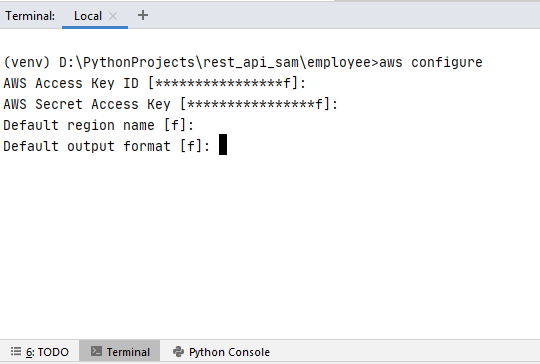
**Note:** *The AWS Serverless Application Model (AWS SAM) is an open-source framework that enables you to build serverless applications on AWS. It provides you with a template specification to define your serverless application, and a command line interface (CLI) tool.*

*The SAM Command Line Interface (CLI) lets you locally build, test, and debug serverless applications defined by AWS SAM templates. SAM CLI provides a Lambda-like execution environment locally and helps you catch issues upfront.*

*The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services*

**Step 6:** Get and IAM user account in AWS and run below for configure AWS CLI and provide the details

* *aws configure*



**Note:** *IAM Users. An AWS Identity and Access Management (IAM) user is an entity that you create in AWS to represent the person or application that uses it to interact with AWS.*

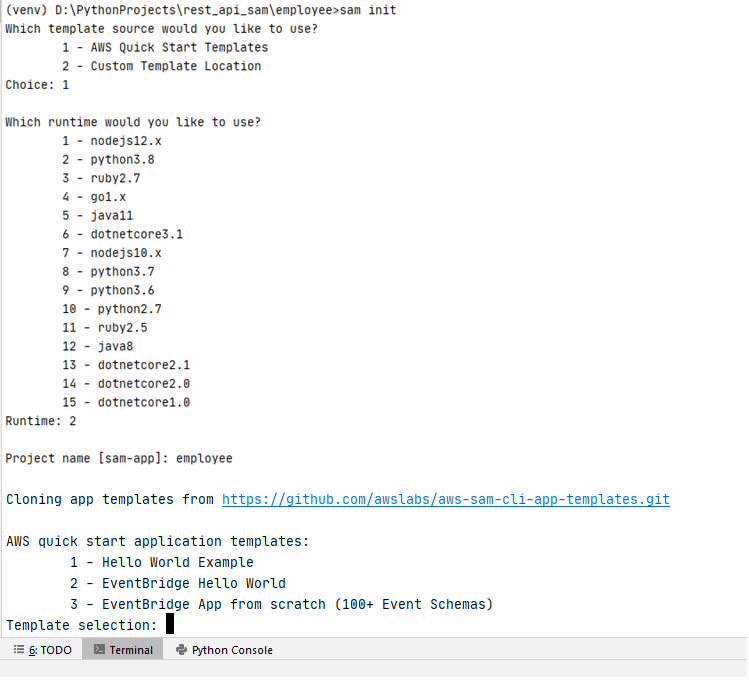
*Access keys are long-term credentials for an IAM user or the AWS account root user. You can use access keys to sign programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK)*

*Access keys consist of two parts: an* ***access key ID*** *and a* ***secret access key*** *Like a username and password, you must use both the access key ID and secret access key together to authenticate your requests.*

**Step 7:** Install GIT if not already installed and restart the PyCharm IDE.

**Step 8:** Run below command to download a sample application

* sam init



**Step 9:** Run the below code script

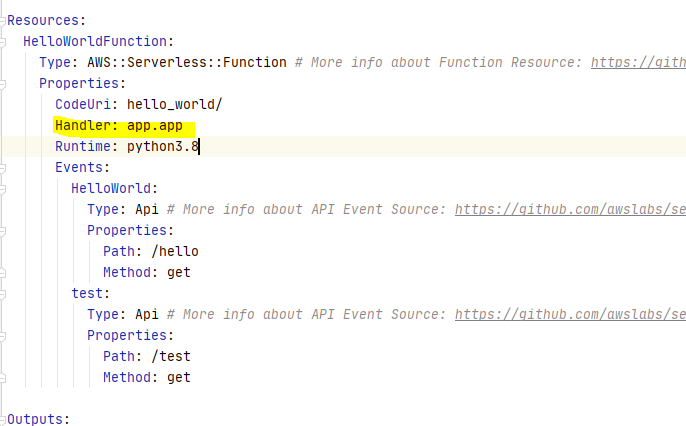
* pip install flask-lambda-python36

**Note:** Lambda is response to only events. So, flask-lambda-python36 module to make Flask compatible with AWS Lambda

**Step 10:** Replace below code into ..\employee\hello\_world\app.py



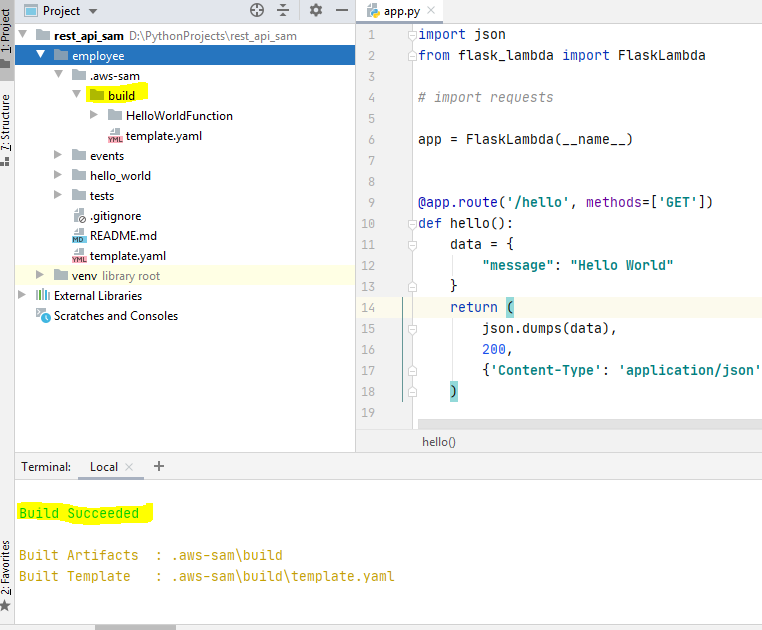
**Step 10:** Open the employee\template.xml file and change value of **Handler** to **app.app**



**Step 11:** Run the below command for building the application

* sam build --use-container

Once build successful you will get build folder and success message as shown below



**Note:** Read the file (employee\README.md ) and follow the step for deploy the application

To test the service in locally we need to install and configure Docker.

* SET DOCKER\_TLS\_VERIFY=1
* SET DOCKER\_HOST=tcp://localhost:2375
* SET DOCKER\_CERT\_PATH=C:\Users\name.docker\machine\machines\box
* SET DOCKER\_MACHINE\_NAME=box
* SET COMPOSE\_CONVERT\_WINDOWS\_PATHS=true

**Step 12:** Run the below command to run the API in locally.

* sam local start-api

**Step 13:** Once testing completed deployed the application using below command

* sam deploy –guided

**Cons of Lambda Function:**

AWS Lambda functions are timeboxed, with a default timeout of three seconds (it is configurable up to five minutes). This means you need to spend more time orchestrating and organizing your functions, so that they can work in a distributed fashion on your data.

Many Lambda functions operate in a hot-cold fashion, similar to containers. The first call to a function after a long idle period results in a brief delay as AWS spins up a machine instance for your function – this is called a “cold function”. While this instance is preserved for about 10 minutes after your code’s execution (making the function a “hot function,” meaning it is readily available without spin-up), after that 10th minute your function will be subject to the warmup period again. While there are some techniques to keep your functions “warm”, the spin-up costs cannot be avoided entirely

**Ref:** <https://docs.aws.amazon.com/>