Building a SAM Application

In this lesson, you will learn how to build a SAM application.

WE'LL COVER THE FOLLOWING

Step 1: Build

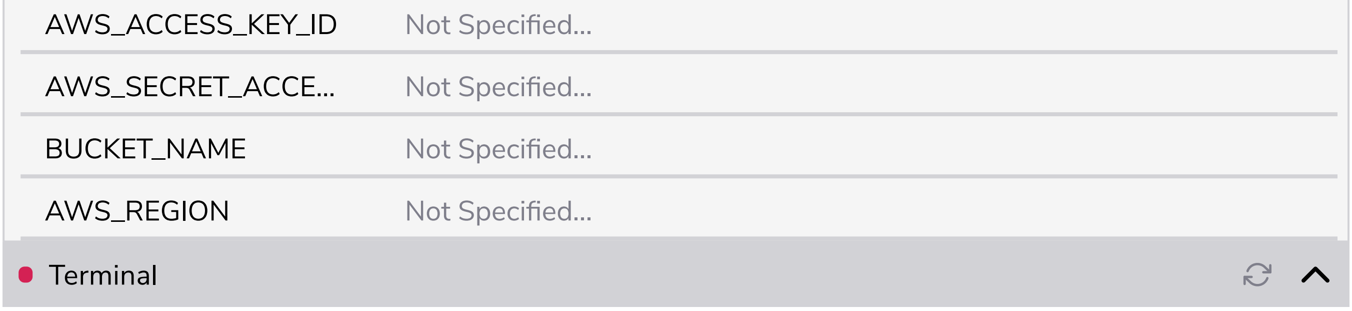
Most modern software programming languages make it easy to reference third-party libraries and packages. For example, Node.js comes with NPM, a package manager that installs the libraries listed in the `package.json` project manifest. Python works with `pip`. Java has `maven`. In order for Lambda functions to start as quickly as possible, you need to install and bundle dependencies before uploading it to AWS. CloudFormation doesn't know how to do that. However, the SAM command line tools know how to install and bundle dependencies for many common package managers. For compiled languages, such as Java and Go, SAM knows how to turn source files into executable code.

To prepare the function source for uploading to AWS, execute the following command in the main project directory (`app`). This will be the one that contains the CloudFormation template (`template.yaml`):

```
sam build
```

Environment Variables

Key:	Value:
LANG	C.UTF-8
LC_ALL	C.UTF-8



AWS_ACCESS_KEY_ID	Not Specified...
AWS_SECRET_ACCE...	Not Specified...
BUCKET_NAME	Not Specified...
AWS_REGION	Not Specified...

You should see a report that SAM built the `HelloWorldFunction` using `npm`:

```
$ sam build
Building resource 'HelloWorldFunction'
Running NodejsNpmBuilder:NpmPack
Running NodejsNpmBuilder:CopyNpmrc
Running NodejsNpmBuilder:CopySource
Running NodejsNpmBuilder:NpmInstall
Running NodejsNpmBuilder:CleanUpNpmrc

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml
```

The `sam build` command copies project source files into a temporary subdirectory and runs the required packager to install all production dependencies for functions. It knows how to ignore test code, resources, and avoids bundling development dependencies. This means that you can safely install development tools in your source directory; SAM will ignore them when building the functions.

If your build process needs to compile binary executables, pass the `--use-container` option to `sam build`. This will execute the build process inside a Docker container that matches the Lambda runtime. For JavaScript, this is normally not needed. On the other hand, many Python libraries try to compile native dependencies and building inside a Lambda-like container is very useful for those cases.

SAM creates a temporary directory for build artefacts, defaulting to a subdirectory inside your project called `.aws-sam`. You can make it write the build results to a different location by using the `--build-dir` option. Check out the SAM Build documentation for more information on this and other options.

Building packages for other languages

Building packages for other languages

> At the time when it was written, in January 2020, SAM command-line tools supported building JavaScript, Python, Ruby, Java, and Go projects. This feature was under active development then, so it is likely that support for other languages will have been implemented by the time you start reading this course.
>
> If your chosen packaging system is not yet supported, you will have to somehow bundle dependencies and remove development tools before uploading the code to CloudFormation. In this case, you can skip the `sam build` step.

In the next lesson, you’ll look at the remaining steps required to deploy SAM applications.