:doctype: book :pp: {plus}{plus}

include::attributes.txt[]

// Attributes [.topic] [#work-with-cdk-python] = Working with the {aws} CDK in Python :info\_titleabbrev: In Python

// Content start

Python is a fully-supported client language for the {aws} Cloud Development Kit ({aws} CDK) and is considered stable. Working with the {aws} CDK in Python uses familiar tools, including the standard Python implementation (CPython), virtual environments with virtualenv, and the Python package installer pip. The modules comprising the {aws} Construct Library are distributed via https://pypi.org/search/?q=aws-cdk[pypi.org]. The Python version of the {aws} CDK even uses Python-style identifiers (for example, snake\_case method names).

You can use any editor or IDE. Many {aws} CDK developers use https://code.visualstudio.com/[Visual Studio Code] (or its open-source equivalent https://vscodium.com/[VSCodium]), which has good support for Python via an https://marketplace.visualstudio.com/items?itemName=ms-python.python[official extension]. The IDLE editor included with Python will suffice to get started. The Python modules for the {aws} CDK do have type hints, which are useful for a linting tool or an IDE that supports type validation.

[#python-prerequisites] == Get started with Python

To work with the {aws} CDK, you must have an {aws} account and credentials and have installed Node.js and the {aws} CDK Toolkit. See xref:getting-started[Getting started with the {aws} CDK].

Python {aws} CDK applications require Python 3.6 or later. If you don’t already have it installed, https://www.python.org/downloads/[download a compatible version] for your operating system at https://www.python.org/[python.org]. If you run Linux, your system may have come with a compatible version, or you may install it using your distro’s package manager (yum, apt, etc.). Mac users may be interested in https://brew.sh/[Homebrew], a Linux-style package manager for macOS.

= [NOTE]

Third-party language deprecation: language version is only supported until its EOL (End Of Life) shared by the vendor or community and is subject to change with prior notice.

====

The Python package installer, pip, and virtual environment manager, virtualenv, are also required. Windows installations of compatible Python versions include these tools. On Linux, pip and virtualenv may be provided as separate packages in your package manager. Alternatively, you may install them with the following commands:

== [source,none,subs=“verbatim,attributes”]

python -m ensurepip –upgrade python -m pip install –upgrade pip python -m pip install –upgrade virtualenv —

If you encounter a permission error, run the above commands with the --user flag so that the modules are installed in your user directory, or use sudo to obtain the permissions to install the modules system-wide.

= [NOTE]

It is common for Linux distros to use the executable name python3 for Python 3.x, and have python refer to a Python 2.x installation. Some distros have an optional package you can install that makes the python command refer to Python 3. Failing that, you can adjust the command used to run your application by editing cdk.json in the project’s main directory.

====

= [NOTE]

On Windows, you may want to invoke Python (and pip) using the py executable, the link:https://docs.python.org/3/using/windows.html#launcher[Python launcher for Windows]. Among other things, the launcher allows you to easily specify which installed version of Python you want to use.

If typing python at the command line results in a message about installing Python from the Windows Store, even after installing a Windows version of Python, open Windows’ Manage App Execution Aliases settings panel and turn off the two App Installer entries for Python.

====

[#python-newproject] == Creating a project

You create a new {aws} CDK project by invoking cdk init in an empty directory. Use the --language option and specify python:

== [source,bash,subs=“verbatim,attributes”]

$ mkdir my-project $ cd my-project $ cdk init app –language python —

cdk init uses the name of the project folder to name various elements of the project, including classes, subfolders, and files. Hyphens in the folder name are converted to underscores. However, the name should otherwise follow the form of a Python identifier; for example, it should not start with a number or contain spaces.

To work with the new project, activate its virtual environment. This allows the project’s dependencies to be installed locally in the project folder, instead of globally.

== [source,bash,subs=“verbatim,attributes”]

## $ source .venv/bin/activate

= [NOTE]

You may recognize this as the Mac/Linux command to activate a virtual environment. The Python templates include a batch file, source.bat, that allows the same command to be used on Windows. The traditional Windows command, .\venv\Scripts\activate, works, too.

If you initialized your {aws} CDK project using CDK Toolkit v1.70.0 or earlier, your virtual environment is in the .env directory instead of .venv.

====

= [IMPORTANT]

Activate the project’s virtual environment whenever you start working on it. Otherwise, you won’t have access to the modules installed there, and modules you install will go in the Python global module directory (or will result in a permission error).

====

After activating your virtual environment for the first time, install the app’s standard dependencies:

== [source,bash,subs=“verbatim,attributes”]

## $ python -m pip install -r requirements.txt

[#python-managemodules] == Managing {aws} Construct Library modules

Use the Python package installer, pip, to install and update {aws} Construct Library modules for use by your apps, as well as other packages you need. pip also installs the dependencies for those modules automatically. If your system does not recognize pip as a standalone command, invoke pip as a Python module, like this:

== [source,bash,subs=“verbatim,attributes”]

$ python -m pip ++++++—-++++++

Most {aws} CDK constructs are in aws-cdk-lib. Experimental modules are in separate modules named like aws-cdk.<SERVICE-NAME>.alpha. The service name includes an *aws* prefix. If you’re unsure of a module’s name, https://pypi.org/search/?q=aws-cdk[search for it at PyPI]. For example, the command below installs the {aws} CodeStar library.

== [source,bash,subs=“verbatim,attributes”]

## $ python -m pip install aws-cdk.aws-codestar-alpha

Some services’ constructs are in more than one namespace. For example, besides aws-cdk.aws-route53, there are three additional Amazon Route 53 namespaces, named aws-route53-targets, aws-route53-patterns, and aws-route53resolver.

= [NOTE]

The https://docs.aws.amazon.com/cdk/api/v2/python/index.html[Python edition of the CDK API Reference] also shows the package names.

====

The names used for importing {aws} Construct Library modules into your Python code look like the following.

== [source,python,subs=“verbatim,attributes”]

import aws\_cdk.aws\_s3 as s3 import aws\_cdk.aws\_lambda as lambda\_ —

We recommend the following practices when importing {aws} CDK classes and {aws} Construct Library modules in your applications. Following these guidelines will help make your code consistent with other {aws} CDK applications as well as easier to understand.

* Generally, import individual classes from top-level aws\_cdk.

## [source,python,subs=“verbatim,attributes”]

## from aws\_cdk import App, Construct

* If you need many classes from the aws\_cdk, you may use a namespace alias of cdk instead of importing individual classes. Avoid doing both.

## [source,python,subs=“verbatim,attributes”]

## import aws\_cdk as cdk

* Generally, import {aws} Construct Libraries using short namespace aliases.

## [source,python,subs=“verbatim,attributes”]

## import aws\_cdk.aws\_s3 as s3

After installing a module, update your project’s requirements.txt file, which lists your project’s dependencies. It is best to do this manually rather than using pip freeze. pip freeze captures the current versions of all modules installed in your Python virtual environment, which can be useful when bundling up a project to be run elsewhere.

Usually, though, your requirements.txt should list only top-level dependencies (modules that your app depends on directly) and not the dependencies of those libraries. This strategy makes updating your dependencies simpler.

You can edit requirements.txt to allow upgrades; simply replace the == preceding a version number with ~= to allow upgrades to a higher compatible version, or remove the version requirement entirely to specify the latest available version of the module.

With requirements.txt edited appropriately to allow upgrades, issue this command to upgrade your project’s installed modules at any time:

== [source,bash,subs=“verbatim,attributes”]

## $ pip install –upgrade -r requirements.txt

[#work-with-cdk-python-dependencies] == Managing dependencies in Python

In Python, you specify dependencies by putting them in requirements.txt for applications or setup.py for construct libraries. Dependencies are then managed with the PIP tool. PIP is invoked in one of the following ways:

== [source,none,subs=“verbatim,attributes”]

pip ++++++++++++ python -m pip ++++++++++++ —

The python -m pip invocation works on most systems; pip requires that PIP’s executable be on the system path. If pip doesn’t work, try replacing it with python -m pip.

The cdk init --language python command creates a virtual environment for your new project. This lets each project have its own versions of dependencies, and also a basic requirements.txt file. You must activate this virtual environment by running source .venv/bin/activate each time you begin working with the project. On Windows, run .\venv\Scripts\activate instead

[#work-with-cdk-python-dependencies-apps] === CDK applications

The following is an example requirements.txt file. Because PIP does not have a dependency-locking feature, we recommend that you use the == operator to specify exact versions for all dependencies, as shown here.

== [source,none,subs=“verbatim,attributes”]

aws-cdk-lib==2.14.0 aws-cdk.aws-appsync-alpha==2.10.0a0 —

Installing a module with pip install does not automatically add it to requirements.txt. You must do that yourself. If you want to upgrade to a later version of a dependency, edit its version number in requirements.txt.

To install or update your project’s dependencies after creating or editing requirements.txt, run the following:

== [source,none,subs=“verbatim,attributes”]

## python -m pip install -r requirements.txt

= [TIP]

The pip freeze command outputs the versions of all installed dependencies in a format that can be written to a text file. This can be used as a requirements file with pip install -r. This file is convenient for pinning all dependencies (including transitive ones) to the exact versions that you tested with. To avoid problems when upgrading packages later, use a separate file for this, such as freeze.txt (not requirements.txt). Then, regenerate it when you upgrade your project’s dependencies.

====

[#work-with-cdk-python-dependencies-libraries] === Third-party construct libraries

In libraries, dependencies are specified in setup.py, so that transitive dependencies are automatically downloaded when the package is consumed by an application. Otherwise, every application that wants to use your package needs to copy your dependencies into their requirements.txt. An example setup.py is shown here.

== [source,python,subs=“verbatim,attributes”]

from setuptools import setup

setup( name=‘my-package’, version=‘0.0.1’, install\_requires=[ ‘aws-cdk-lib==2.14.0’, ], … ) —

To work on the package for development, create or activate a virtual environment, then run the following command.

== [source,none,subs=“verbatim,attributes”]

## python -m pip install -e .

Although PIP automatically installs transitive dependencies, there can only be one installed copy of any one package. The version that is specified highest in the dependency tree is selected; applications always have the last word in what version of packages get installed.

[#python-cdk-idioms] == {aws} CDK idioms in Python

[#python-keywords] === Language conflicts

In Python, lambda is a language keyword, so you cannot use it as a name for the {aws} Lambda construct library module or Lambda functions. The Python convention for such conflicts is to use a trailing underscore, as in lambda\_, in the variable name.

By convention, the second argument to {aws} CDK constructs is named id. When writing your own stacks and constructs, calling a parameter id “shadows” the Python built-in function id(), which returns an object’s unique identifier. This function isn’t used very often, but if you should happen to need it in your construct, rename the argument, for example construct\_id.

[#python-props] === Arguments and properties

All {aws} Construct Library classes are instantiated using three arguments: the *scope* in which the construct is being defined (its parent in the construct tree), an *id*, and *props*, a bundle of key/value pairs that the construct uses to configure the resources it creates. Other classes and methods also use the “bundle of attributes” pattern for arguments.

*scope* and *id* should always be passed as positional arguments, not keyword arguments, because their names change if the construct accepts a property named *scope* or *id*.

In Python, props are expressed as keyword arguments. If an argument contains nested data structures, these are expressed using a class which takes its own keyword arguments at instantiation. The same pattern is applied to other method calls that take a structured argument.

For example, in a Amazon S3 bucket’s add\_lifecycle\_rule method, the transitions property is a list of Transition instances.

== [source,python,subs=“verbatim,attributes”]

bucket.add\_lifecycle\_rule( transitions=[ Transition( storage\_class=StorageClass.GLACIER, transition\_after=Duration.days(10) ) ] ) —

When extending a class or overriding a method, you may want to accept additional arguments for your own purposes that are not understood by the parent class. In this case you should accept the arguments you don’t care about using the {pp}+**kwargs{pp}+ idiom, and use keyword-only arguments to accept the arguments you’re interested in. When calling the parent’s constructor or the overridden method, pass only the arguments it is expecting (often just {pp}+**kwargs{pp}+). Passing arguments that the parent class or method doesn’t expect results in an error.

== [source,python,subs=“verbatim,attributes”]

class MyConstruct(Construct): def *init*(self, id, \*, MyProperty=42, **kwargs): super().*init*(self, id,** kwargs) # … —

A future release of the {aws} CDK could coincidentally add a new property with a name you used for your own property. This won’t cause any technical issues for users of your construct or method (since your property isn’t passed “up the chain,” the parent class or overridden method will simply use a default value) but it may cause confusion. You can avoid this potential problem by naming your properties so they clearly belong to your construct. If there are many new properties, bundle them into an appropriately-named class and pass it as a single keyword argument.

[#python-missing-values,] === Missing values

The {aws} CDK uses None to represent missing or undefined values. When working with {pp}+\*\*kwargs{pp}+, use the dictionary’s get() method to provide a default value if a property is not provided. Avoid using +kwargs[...]+, as this raises KeyError for missing values.

== [source,bash,subs=“verbatim,attributes”]

encrypted = kwargs.get(“encrypted”) # None if no property “encrypted” exists encrypted = kwargs.get(“encrypted”, False) # specify default of False if property is missing —

Some {aws} CDK methods (such as tryGetContext() to get a runtime context value) may return None, which you will need to check explicitly.

[#python-interfaces] === Using interfaces

Python doesn’t have an interface feature as some other languages do, though it does have https://docs.python.org/3/library/abc.html[abstract base classes], which are similar. (If you’re not familiar with interfaces, Wikipedia has link:https://en.wikipedia.org/wiki/Interface\_(computing)#In\_object-oriented\_languages[a good introduction].) TypeScript, the language in which the {aws} CDK is implemented, does provide interfaces, and constructs and other {aws} CDK objects often require an object that adheres to a particular interface, rather than inheriting from a particular class. So the {aws} CDK provides its own interface feature as part of the https://github.com/aws/jsii[JSII] layer.

To indicate that a class implements a particular interface, you can use the @jsii.implements decorator:

== [source,python,subs=“verbatim,attributes”]

from aws\_cdk import IAspect, IConstruct import jsii

@jsii.implements(IAspect) class MyAspect(): def visit(self, node: IConstruct) -> None: print(“Visited”, node.node.path) —

[#python-type-pitfalls] === Type pitfalls

Python uses dynamic typing, where all variables may refer to a value of any type. Parameters and return values may be annotated with types, but these are “hints” and are not enforced. This means that in Python, it is easy to pass the incorrect type of value to a {aws} CDK construct. Instead of getting a type error during build, as you would from a statically-typed language, you may instead get a runtime error when the JSII layer (which translates between Python and the {aws} CDK’s TypeScript core) is unable to deal with the unexpected type.

In our experience, the type errors Python programmers make tend to fall into these categories.

* Passing a single value where a construct expects a container (Python list or dictionary) or vice versa.
* Passing a value of a type associated with a layer 1 (CfnXxxxxx) construct to a L2 or L3 construct, or vice versa.

The {aws} CDK Python modules do include type annotations, so you can use tools that support them to help with types. If you are not using an IDE that supports these, such as https://www.jetbrains.com/pycharm/[PyCharm], you might want to call the http://mypy-lang.org/[MyPy] type validator as a step in your build process. There are also runtime type checkers that can improve error messages for type-related errors.