kustomize

Diogo Baeder

Nov 20, 2020

CONTENTS

CHAPTER

ONE

0.5.0

- Separating generated modules by directories to avoid name clashes.

CHAPTER

TWO

0.4.1

- Fixing directory creation for dumping built YAML files.

CHAPTER

THREE

0.4.0

- Now supporting models from python-kubernetes>=11.

CHAPTER

FOUR

0.3.6

- Adding logs.

CHAPTER

FIVE

0.3.5

- Now supporting Kubernetes objects correctly.

CHAPTER

SIX

0.3.4

- Fixing cleanup of data before dump.

CHAPTER

SEVEN

0.3.3

- Cleaning up the data before dumping.

CHAPTER

EIGHT

0.3.2

- Preferring to_dict() to dataclasses.asdict(). This is because objects might need to be more specific about how they should convert themselves to a dict.

CHAPTER

NINE

0.3.1

- Fixing the serialization of objects inside tuples.

CHAPTER

TEN

0.3.0

- Supporting tuples for multi-section YAML generation.

CHAPTER

ELEVEN

0.2.0

Supporting other types for generating dictionaries:

- Classes with a `to_dict` method
- Classes from the `attr` library
- "Flat" classes that can be serialized through `__dict__`

CHAPTER

TWELVE

0.1.2

- Supporting patchesJson6902

CHAPTER

THIRTEEN

0.1.1

- Supporting patchesStrategicMerge

CHAPTER

FOURTEEN

0.1.0

- First release!
- Supporting dictionaries and dataclasses.

CHAPTER

FIFTEEN

PYTHON-KUSTOMIZE

Build your Kubernetes manifests for Kustomize in Python!

- PyPI: https://pypi.org/project/kustomize/
- Repository: https://github.com/yougov/python-kustomize
- Documentation: https://python-kustomize.readthedocs.io/en/latest/

15.1 Overview

The reason for this project to exist is to make it easier to create dynamic manifests to be exported for usage in Kubernetes' "Kustomize" tool. And, by using Python and supporting the "dataclasses" language feature, it also helps reducing boilerplate by encouraging code reuse.

Kustomize, by itself, is already a very powerful tool, and it's possible to deal with different apps and environments by using the "overlays" approach; but it's not dynamic enough if you need to define manifests parameters through environment variables, for example. So this project aims to cover that gap.

15.2 A complement for Kustomize

This project is by no means a replacement for Kustomize, but rather a complement. The idea is to generate kustomization files from Python files, and then use `kubectl apply -k` or `kustomize build` to transform them into final manifests for Kubernetes (even applying them to the cluster).

In other words, the idea is to "compile" Python files into Kustomize files, then just use Kustomize for the rest of the deployment.

15.3 Installing

The only mandatory dependency to this project is `PyYAML`. Besides this, you can have `attr` installed if you want to use their classes, and, if you're running on Python 3.6, you can install `dataclasses` to use them - although this project is tested on Python 3.7 and 3.8 only, it probably runs fine on 3.6.

This package will be available as `kustomize`; you may install it with pip, for example:

```
$ pip install kustomize
```

This will also install `PyYAML` if it's not already installed.

Alternatively, you can use any other package manager capable of installing packages from PyPI.

15.4 Usage

The summary is:

1. You write a source directory with Python files representing the Kustomize files (see directories at `python-kustomize/tests/fixtures/`);
2. You run:

   ```
   $ pykustomize <source-dir> <dest-dir>
   ```

   where `<dest-dir>` will be the directory where Kustomize YAML files will be put at;
3. Then you can apply the generated Kustomize files into your cluster:

   ```
   $ kubectl apply -f <dest-dir>
   ```

   and done!

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

INDEX