À l'aide de Python

Astra Automation 22.04

NetApp
December 04, 2023

Sommaire

À l'aide de Python

Kit de développement logiciel NetApp Astra Control Python

Le kit de développement logiciel NetApp Astra Control Python est un logiciel open source qui permet d'automatiser le déploiement d'Astra Control. Il constitue également une ressource précieuse pour en savoir plus sur l'API REST Astra Control, peut-être dans le cadre de la création de votre propre plateforme d'automatisation.

> Dans un souci de simplicité, le kit de développement NetApp Astra Control Python sera appelé **SDK** au cours du reste de cette page.

Deux outils logiciels connexes

Le SDK inclut deux outils différents bien que associés qui fonctionnent à différents niveaux d'abstraction lors de l'accès à l'API REST Astra Control.

Kit de développement Astra

Le kit de développement logiciel Astra fournit les principales fonctionnalités de la plateforme. Il inclut un ensemble de classes Python qui abstraite les appels de l'API REST sous-jacente. Ces classes prennent en charge les actions administratives sur diverses ressources Astra Control, notamment les applications, les sauvegardes, les instantanés et les clusters.

Le kit de développement Astra fait partie de l'offre et est fourni en une seule pièce `astraSDK.py` fichier. Vous pouvez importer ce fichier dans votre environnement et utiliser les classes directement.

> Le **SDK Python de contrôle Astra** (ou uniquement SDK) de NetApp est le nom de l'ensemble du package. Le **Astra SDK** fait référence aux classes Python de base dans le fichier unique `astraSDK.py`.

Script Toolkit

En plus du fichier Astra SDK, le `toolkit.py` script également disponible. Ce script fonctionne à un niveau d'abstraction plus élevé en donnant accès à des actions administratives discrètes définies en interne comme des fonctions Python. Le script importe le SDK Astra et fait des appels aux classes selon les besoins.

Comment y accéder

Vous pouvez accéder au SDK de la manière suivante.

Pack Python

Le SDK est disponible à l'adresse "Index des paquets Python" sous le nom **netapp-astra-toolkits**. Un numéro de version est attribué au package et continuera d'être mis à jour au besoin. Vous devez utiliser l'utilitaire de gestion de paquets **PIP** pour installer le package dans votre environnement.

Voir "PyPI : kit de développement Python de contrôle NetApp Astra" pour en savoir plus.

Code source GitHub

Le code source du SDK est également disponible sur GitHub. Le référentiel inclut les éléments suivants :

- `astraSDK.py` (SDK Astra avec classes Python)
- `toolkit.py` (script basé sur les fonctions de niveau supérieur)
- Conditions requises et instructions détaillées pour l'installation
- Scripts d'installation
- Documentation complémentaire

Vous pouvez cloner le "GitHub : kits NetApp/netapp-astra-toolkits" référentiel pour votre environnement local.

Conditions requises pour l'installation et de base

Il y a plusieurs options et conditions requises à considérer dans le cadre de l'installation de l'emballage et de la préparation à l'utilisation.

Résumé des options d'installation

Vous pouvez installer le SDK de l'une des manières suivantes :

- Utilisez PIP pour installer le package de PyPI dans votre environnement Python
- Clonez le référentiel Git Hub et :
  - Déploiement du package en tant que conteneur Docker (qui inclut tout ce dont vous avez besoin)
  - Copiez les deux fichiers Python de base afin qu'ils soient accessibles à votre code client Python

Reportez-vous aux pages PyPI et GitHub pour plus d'informations.

Exigences relatives à l'environnement Astra Control

Qu'il s'agisse d'utiliser directement les classes Python du SDK Astra ou des fonctions du `toolkit.py` À terme, vous accéderez à l'API REST lors d'un déploiement d'Astra Control. Vous aurez donc besoin d'un compte Astra et d'un jeton API. Voir "Avant de commencer" Et les autres pages de la section **Get Started** de cette documentation pour plus d'informations.

Exigences relatives au kit de développement logiciel NetApp Astra Control Python

Le SDK a plusieurs conditions préalables liées à l'environnement Python local. Par exemple, vous devez utiliser Python 3.5 ou ultérieur. En outre, plusieurs paquets Python sont requis. Pour plus d'informations, consultez la page de référentiel GitHub ou la page de package PyPI.

Résumé des ressources utiles

Voici quelques ressources utiles pour commencer.

- "PyPI : kit de développement Python de contrôle NetApp Astra"
- "GitHub : kits NetApp/netapp-astra-toolkits"

Python natif

Avant de commencer

Python est un langage de développement populaire en particulier pour l'automatisation des data centers. Avant d'utiliser les fonctions natives de Python avec plusieurs packages courants, vous devez préparer l'environnement et les fichiers d'entrée requis.

En plus d'accéder directement à l'API REST Astra Control avec Python, NetApp propose un kit d'outils qui résume l'API et élimine une partie de la complexité. Voir "Kit de développement logiciel NetApp Astra Control Python" pour en savoir plus.

Préparation de l'environnement

Les exigences de configuration de base pour exécuter les scripts Python sont décrites ci-dessous.

Python 3

La dernière version de Python 3 doit être installée.

Bibliothèques supplémentaires

Les bibliothèques **requêtes** et **urllib3** doivent être installées. Vous pouvez utiliser pip ou un autre outil de gestion Python, selon les besoins de votre environnement.

Accès réseau

Le poste de travail sur lequel les scripts s'exécutent doit disposer d'un accès réseau et pouvoir accéder à Astra Control. Lorsque vous utilisez le service Astra Control, vous devez être connecté à Internet et être en mesure de vous connecter au service à `https://astra.netapp.io`.

Informations d'identité

Vous avez besoin d'un compte Astra valide avec l'identifiant de compte et le jeton API. Voir "Obtenir un jeton API" pour en savoir plus.

Créez les fichiers d'entrée JSON

Les scripts Python s'appuient sur les informations de configuration contenues dans les fichiers d'entrée JSON. Des exemples de fichiers sont fournis ci-dessous.

Vous devez mettre à jour les échantillons en fonction de votre environnement.

Informations d'identité

Le fichier suivant contient le jeton API et le compte Astra. Vous devez transmettre ce fichier aux scripts Python à l'aide de `-i` (ou `--identity`) Paramètre CLI.

```
{
  "api_token": "kH4CA_uVIa8q9UuPzhJaAHaGlaR7-no901DkkrVjIXk=",
  "account_id": "5131dfdf-03a4-5218-ad4b-fe84442b9786"
}
```

Répertorier les applications gérées

Vous pouvez utiliser le script suivant pour répertorier les applications gérées pour votre compte Astra.

Voir "Avant de commencer" Par exemple le fichier d'entrée JSON requis.

```
#!/usr/bin/env python3
##-----------------------------------------------------------------------
------
#
# Usage: python3 list_man_apps.py -i identity_file.json
#
# (C) Copyright 2021 NetApp, Inc.
#
# This sample code is provided AS IS, with no support or warranties of
# any kind, including but not limited for warranties of merchantability
# or fitness of any kind, expressed or implied. Permission to use,
# reproduce, modify and create derivatives of the sample code is granted
# solely for the purpose of researching, designing, developing and
# testing a software application product for use with NetApp products,
# provided that the above copyright notice appears in all copies and
# that the software application product is distributed pursuant to terms
# no less restrictive than those set forth herein.
#
##-----------------------------------------------------------------------
------

import argparse
import json
import requests
import urllib3
import sys

# Global variables
api_token = ""
account_id = ""

def get_managed_apps():
    ''' Get and print the list of managed apps '''

    # Global variables
    global api_token
    global account_id

    # Create an HTTP session
    sess1 = requests.Session()

    # Suppress SSL unsigned certificate warning
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

    # Create URL
    url1 = "https://astra.netapp.io/accounts/" + account_id +
```

```
"/k8s/v1/managedApps"

    # Headers and response output
    req_headers  = {}
    resp_headers = {}
    resp_data    = {}

    # Prepare the request headers
    req_headers.clear
    req_headers['Authorization'] = "Bearer " + api_token
    req_headers['Content-Type'] = "application/astra-managedApp+json"
    req_headers['Accept'] = "application/astra-managedApp+json"

    # Make the REST call
    try:
        resp1 = sess1.request('get', url1, headers=req_headers,
allow_redirects=True, verify=False)

    except requests.exceptions.ConnectionError:
        print("Connection failed")
        sys.exit(1)

    # Retrieve the output
    http_code = resp1.status_code
    resp_headers = resp1.headers

    # Print the list of managed apps
    if resp1.ok:
        resp_data = json.loads(resp1.text)
        items = resp_data['items']
        for i in items:
            print(" ")
            print("Name: " + i['name'])
            print("ID: " + i['id'])
            print("State: " + i['state'])
    else:
        print("Failed with HTTP status code: " + str(http_code))

    print(" ")

    # Close the session
    sess1.close()

    return

def read_id_file(idf):
    ''' Read the identity file and save values '''
```

```
    # Global variables
    global api_token
    global account_id

    with open(idf) as f:
        data = json.load(f)

    api_token = data['api_token']
    account_id = data['account_id']

    return

def main(args):
    ''' Main top level function '''

    # Global variables
    global api_token
    global account_id

    # Retrieve name of JSON input file
    identity_file = args.id_file

    # Get token and account
    read_id_file(identity_file)

    # Issue REST call
    get_managed_apps()

    return

def parseArgs():
        ''' Parse the CLI input parameters '''

        parser = argparse.ArgumentParser(description='Astra REST API -
List the managed apps',
                                add_help = True)
        parser.add_argument("-i", "--identity", action="store", dest
="id_file", default=None,
                                help='(Req) Name of the identity input file',
required=True)

        return parser.parse_args()

if __name__ == '__main__':
    ''' Begin here '''
```

```
    # Parse input parameters
    args = parseArgs()

    # Call main function
    main(args)
```