Verwendung Von Python

Astra Automation

NetApp
May 29, 2024

Inhalt

Verwendung Von Python

NetApp Astra Control Python SDK

NetApp Astra Control Python SDK ist ein Open-Source-Paket, mit dem Sie eine Astra Control Implementierung automatisieren können. Das Paket ist auch eine wertvolle Ressource, um sich über die Astra Control REST API zu informieren, vielleicht im Rahmen der Erstellung Ihrer eigenen Automatisierungsplattform.

Zur Einfachheit wird das NetApp Astra Control Python SDK durchgehend auf dieser Seite als **SDK** bezeichnet.

Zwei verwandte Softwaretools

Das SDK enthält zwei verschiedene, wenn auch verwandte Tools, die auf verschiedenen Ebenen der Abstraktion beim Zugriff auf die Astra Control REST API arbeiten.

Astra SDK

Das Astra SDK bietet die Kernfunktionen der Plattform. Es enthält eine Reihe von Python-Klassen, in denen die zugrunde liegenden REST-API-Aufrufe abstrahiert werden. Die Klassen unterstützen administrative Aktionen auf verschiedenen Astra Control Ressourcen, einschließlich Apps, Backups, Snapshots und Cluster.

Das Astra SDK ist ein Teil des Pakets und wird in der Single bereitgestellt `astraSDK.py` Datei: Sie können diese Datei in Ihre Umgebung importieren und die Klassen direkt verwenden.

Das **NetApp Astra Control Python SDK** (oder nur SDK) ist der Name des gesamten Pakets. Das **Astra SDK** bezieht sich auf die Core Python Klassen in der einzelnen Datei `astraSDK.py`.

Toolkit-Skript

Neben der Astra SDK-Datei, die `toolkit.py` Skript ist ebenfalls verfügbar. Dieses Skript wird auf einer höheren Abstraktionsebene ausgeführt und bietet Zugriff auf diskrete, intern als Python-Funktionen definierte administrative Aktionen. Das Skript importiert das Astra SDK und ruft nach Bedarf die Klassen an.

Zugang zu

Sie haben folgende Möglichkeiten, auf das SDK zuzugreifen:

Python-Paket

Das SDK ist unter verfügbar "Python-Paketindex" Unter dem Namen **acToolkit**. Dem Paket wird eine Versionsnummer zugewiesen und bei Bedarf auch weiterhin aktualisiert. Sie müssen das Paketverwaltungsprogramm * PIP* verwenden, um das Paket in Ihrer Umgebung zu installieren.

Nach der Installation können die astraSDK.py Klassen durch Platzieren genutzt werden `import astraSDK` In Ihren Skripten. Darüber Hinaus `actoolkit` Kann direkt an Ihrer Eingabeaufforderung aufgerufen werden und entspricht `toolkit.py` (`actoolkit list clusters` Ist das gleiche wie `./toolkit.py list clusters`).

Siehe "PyPI: NetApp Astra Control Python SDK" Finden Sie weitere Informationen.

GitHub-Quellcode

Der SDK-Quellcode ist auch bei GitHub erhältlich. Das Repository umfasst Folgendes:

- `astraSDK.py` (Astra SDK mit Python-Klassen)
- `toolkit.py` (Auf höherer Ebene funktionbasiertes Skript)
- Detaillierte Installationsanforderungen und Anweisungen
- Installationsskripte
- Zusätzliche Dokumentation

Sie können die klonen "GitHub: NetApp/netapp-astra-Toolkits" Repository in Ihre lokale Umgebung einbinden

Installation und grundlegende Anforderungen

Es gibt verschiedene Optionen und Anforderungen, die bei der Installation des Pakets und bei der Vorbereitung der Verwendung berücksichtigt werden müssen.

Zusammenfassung der Installationsoptionen

Sie können das SDK auf eine der folgenden Arten installieren:

- Verwenden Sie das vorbereitete "Docker: NetApp/astra-Toolkits" Image, das alle erforderlichen Abhängigkeiten installiert hat, einschließlich `actoolkit`
- Verwenden Sie Pip, um den zu installieren `actoolkit` Paket von PyPI in Ihre Python-Umgebung
- Klonen Sie das GitHub Repository und kopieren/ändern Sie die beiden Core Python-Dateien, damit sie für Ihren Python-Client-Code zugänglich sind

Weitere Informationen finden Sie auf den Seiten PyPI und GitHub.

Anforderungen für die Astra Control-Umgebung

Ob direkt die Python-Klassen im Astra SDK oder die Funktionen im `toolkit.py` Skript, schließlich sind Sie bei einer Implementierung von Astra Control auf DIE REST-API zugreifen. Aus diesem Grund benötigen Sie ein Astra-Konto zusammen mit einem API-Token. Siehe "Bevor Sie beginnen" Und die anderen Seiten im Abschnitt **Get Started** dieser Dokumentation für weitere Informationen.

Anforderungen für das NetApp Astra Control Python SDK

Das SDK verfügt über mehrere Voraussetzungen für die lokale Python-Umgebung. Beispiel: Sie müssen Python 3.8 oder höher verwenden. Darüber hinaus sind mehrere Python-Pakete erforderlich. Weitere Informationen finden Sie auf der GitHub Repository-Seite oder auf der Seite des PyPI-Pakets.

Zusammenfassung hilfreicher Ressourcen

Im Folgenden finden Sie einige Ressourcen, die Sie für den Einstieg benötigen.

- "PyPI: NetApp Astra Control Python SDK"
- "GitHub: NetApp/netapp-astra-Toolkits"
- "Docker: NetApp/astra-Toolkits"

Native Python

Bevor Sie beginnen

Python ist eine beliebte Entwicklungssprache bei der Datacenter-Automatisierung. Bevor Sie die nativen Funktionen von Python zusammen mit mehreren gängigen Paketen nutzen, müssen Sie die Umgebung und die erforderlichen Eingabedateien vorbereiten.

NetApp hat nicht nur direkt mit Python auf die Astra Control REST API zugegriffen, sondern bietet auch ein Toolkit-Paket, das die API abstrahiert und einige der Komplexität beseitigt. Siehe "NetApp Astra Control Python SDK" Finden Sie weitere Informationen.

Bereiten Sie die Umgebung vor

Im Folgenden werden die grundlegenden Konfigurationsanforderungen für die Ausführung der Python-Skripte beschrieben.

Python 3

Sie müssen die neueste Version von Python 3 installiert haben.

Weitere Bibliotheken

Die Bibliotheken **Requests** und **urllib3** müssen installiert sein. Sie können je nach Ihrer Umgebung Pip oder ein anderes Python Management Tool verwenden.

Netzwerkzugriff

Die Arbeitsstation, auf der die Skripte ausgeführt werden, muss Netzwerkzugriff haben und Astra Control erreichen können. Bei der Verwendung des Astra Control Service müssen Sie mit dem Internet verbunden sein und eine Verbindung zum Dienst unter herstellen können `https://astra.netapp.io`.

Identitätsinformationen

Sie benötigen ein gültiges Astra-Konto mit der Account-ID und dem API-Token. Siehe "Holen Sie sich ein API-Token" Finden Sie weitere Informationen.

Erstellen Sie die JSON-Eingabedateien

Die Python-Skripte basieren auf Konfigurationsinformationen in JSON-Eingabedateien. Im Folgenden finden Sie Beispieldateien.

Sie müssen die Proben entsprechend Ihrer Umgebung aktualisieren.

Identitätsinformationen

Die folgende Datei enthält das API-Token und das Astra-Konto. Sie müssen diese Datei mit der an Python-Skripte übergeben `-i` (Oder `--identity`) CLI-Parameter.

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

Listen Sie die Apps auf

Mit dem folgenden Skript können Sie die Anwendungen für Ihr Astra-Konto auflisten.

Siehe "Bevor Sie beginnen" Beispiel für die erforderliche JSON-Eingabedatei.

```
#!/usr/bin/env python3
##-----------------------------------------------------------------------
------
#
# Usage: python3 list_man_apps.py -i identity_file.json
#
# (C) Copyright 2022 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 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/v2/apps"

    # 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-app+json"
    req_headers['Accept'] = "application/astra-app+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 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 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)
```