Utilizzo di Python

Astra Automation 22.08

NetApp
December 04, 2023

Sommario

Utilizzo di Python

SDK NetApp Astra Control Python

NetApp Astra Control Python SDK è un pacchetto open source che puoi utilizzare per automatizzare un'implementazione di Astra Control. Il pacchetto è anche una risorsa preziosa per imparare a conoscere l'API REST di Astra Control, magari come parte della creazione della tua piattaforma di automazione.

> Per semplicità, NetApp Astra Control Python SDK verrà indicato come **SDK** nella parte restante di questa pagina.

Due tool software correlati

L'SDK include due tool diversi, sebbene correlati, che operano a diversi livelli di astrazione quando si accede all'API REST di Astra Control.

SDK Astra

Astra SDK offre le funzionalità principali della piattaforma. Include un insieme di classi Python che astraggono le chiamate API REST sottostanti. Le classi supportano azioni amministrative su varie risorse di Astra Control, tra cui app, backup, snapshot e cluster.

Astra SDK è una parte del pacchetto e viene fornito nel singolo `astraSDK.py` file. È possibile importare questo file nel proprio ambiente e utilizzare direttamente le classi.

> L'SDK * NetApp Astra Control Python (o solo SDK) è il nome dell'intero pacchetto. L'SDK * Astra si riferisce alle classi Python principali nel singolo file `astraSDK.py`.

Script del toolkit

Oltre al file Astra SDK, il `toolkit.py` è disponibile anche uno script. Questo script opera a un livello di astrazione superiore fornendo l'accesso a azioni amministrative discrete definite internamente come funzioni Python. Lo script importa l'SDK Astra ed effettua chiamate alle classi in base alle necessità.

Come accedere

È possibile accedere all'SDK nei seguenti modi.

Pacchetto Python

L'SDK è disponibile all'indirizzo "Python Package Index" sotto il nome **actoolkit**. Al pacchetto viene assegnato un numero di versione e continuerà ad essere aggiornato in base alle necessità. Per installare il pacchetto nel proprio ambiente, è necessario utilizzare l'utility di gestione dei pacchetti **PIP**.

Una volta installate, le astraSDK.py classi possono essere utilizzate collocando `import astraSDK` negli script. Inoltre, `actoolkit` può essere richiamato direttamente dal prompt dei comandi ed è equivalente a. `toolkit.py` (`actoolkit list clusters` è uguale a. `./toolkit.py list clusters`).

Vedere "PyPI: SDK NetApp Astra Control Python" per ulteriori informazioni.

Codice sorgente di GitHub

Il codice sorgente dell'SDK è disponibile anche su GitHub. Il repository include quanto segue:

- `astraSDK.py` (SDK Astra con classi Python)
- `toolkit.py` (script basato sulle funzioni di livello superiore)
- Istruzioni e requisiti di installazione dettagliati
- Script di installazione
- Documentazione aggiuntiva

È possibile clonare "GitHub: NetApp/netapp-astra-toolkit" repository nel tuo ambiente locale.

Installazione e requisiti di base

Esistono diverse opzioni e requisiti da prendere in considerazione durante l'installazione del pacchetto e la preparazione per l'utilizzo.

Riepilogo delle opzioni di installazione

È possibile installare l'SDK in uno dei seguenti modi:

- Utilizzare il preparato "Docker: NetApp/astra-toolkit" immagine, che ha tutte le dipendenze necessarie installate, tra cui `actoolkit`
- Utilizzare PIP per installare `actoolkit` Pacchetto da PyPI nel tuo ambiente Python
- Clonare il repository di GitHub e copiare/modificare i due file Python principali in modo che siano accessibili al codice client Python

Per ulteriori informazioni, fare riferimento alle pagine PyPI e GitHub.

Requisiti per l'ambiente Astra Control

Sia che si utilizzi direttamente le classi Python nell'SDK Astra o le funzioni in `toolkit.py` Script, in ultima analisi, potrai accedere all'API REST in un'implementazione di Astra Control. Per questo motivo, avrai bisogno di un account Astra con un token API. Vedere "Prima di iniziare" E le altre pagine della sezione **Get Started** di questa documentazione per ulteriori informazioni.

Requisiti per NetApp Astra Control Python SDK

L'SDK ha diversi prerequisiti relativi all'ambiente Python locale. Ad esempio, è necessario utilizzare Python 3.8 o versione successiva. Inoltre, sono necessari diversi pacchetti Python. Per ulteriori informazioni, consulta la pagina del repository GitHub o la pagina del pacchetto PyPI.

Riepilogo delle risorse utili

Ecco alcune delle risorse necessarie per iniziare.

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

Python nativo

Prima di iniziare

Python è un popolare linguaggio di sviluppo per l'automazione dei data center. Prima di utilizzare le funzionalità native di Python insieme a diversi pacchetti comuni, è necessario preparare l'ambiente e i file di input richiesti.

> Oltre ad accedere direttamente all'API REST di Astra Control utilizzando Python, NetApp fornisce anche un pacchetto di toolkit che astratta l'API e rimuove alcune delle complessità. Vedere "SDK NetApp Astra Control Python" per ulteriori informazioni.

Preparare l'ambiente

I requisiti di configurazione di base per eseguire gli script Python sono descritti di seguito.

Python 3

Devi avere l'ultima versione di Python 3 installata.

Librerie aggiuntive

Le librerie **requests** e **urllib3** devono essere installate. È possibile utilizzare pip o un altro tool di gestione Python appropriato per il proprio ambiente.

Accesso alla rete

La workstation in cui vengono eseguiti gli script deve disporre dell'accesso di rete e poter raggiungere Astra Control. Quando si utilizza Astra Control Service, è necessario essere connessi a Internet ed essere in grado di connettersi al servizio all'indirizzo `https://astra.netapp.io`.

Informazioni sull'identità

È necessario un account Astra valido con l'identificativo dell'account e il token API. Vedere "Ottieni un token API" per ulteriori informazioni.

Creare i file di input JSON

Gli script Python si basano sulle informazioni di configurazione contenute nei file di input JSON. I file di esempio sono forniti di seguito.

> È necessario aggiornare gli esempi in base all'ambiente in uso.

Informazioni sull'identità

Il seguente file contiene il token API e l'account Astra. È necessario passare questo file agli script Python utilizzando `-i` (o. `--identity`) Parametro CLI.

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

Elencare le applicazioni

Puoi utilizzare il seguente script per elencare le applicazioni per il tuo account Astra.

> Vedere "Prima di iniziare" Per un esempio del file di input JSON richiesto.

```
#!/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)
```