multi-cloud-explorer

Release 0.1.0

stephane.rault

May 07, 2020

INTRODUCTION

Warning: **Ne pas utiliser ce projet en production pour l'instant !!!**

Multi-Cloud-Explorer a pour objectif de **centraliser** l'inventaire et le suivi des changements, pour les ressources de **toutes vos souscriptions** Cloud.

La récupération des resources se fait par plusieurs méthodes:

- Inventaire périodique lancé à partir d'un worker MCE
- Réception des évènements par l'api REST, à partir d'une fonction serverless
- Appel manuel de l'api REST déclenché par vos soins

L'inventaire pourra être exploité par différents moyens:

- Navigation et recherche dans l'application Web
- Bus d'évènements qui recevra tous les changements détectés
- Utilisation de l'api REST

CHAPTER

ONE

ARCHITECTURE

La solution est modulaire et hautement évolutive. Vous pouvez utiliser une partie des composants (ex: mce-lib-azure) ou la totalité (ex: mce-django-server).

1.1 Composants

1.1.1 mce-lib-azure

Librairie pour parcourir toutes les ressources d'une souscription Azure.

1.1.2 mce-lib-aws

Librairie pour parcourir toutes les ressources d'une souscription AWS.

1.1.3 mce-lib-vsphere

Librairie pour parcourir toutes les ressources d'un Vcenter

1.1.4 mce-django-app

Application Django que vous pouvez ajouter à un projet Django existant ou utiliser à travers le projet mce-django-server.

1.1.5 mce-tasks-rq

Implémentation Python RQ du worker de gestion des tâches (alternative à Celery).

1.1.6 mce-django-server

Solution complète, intégrant les composants précédents dans un projet Django et prêt à l'emploi dans un docker-compose.

1.1.7 mce-event-push

Service d'envoi de message pour chaque changement dans l'inventaire.

1.2 Composants obsolètes

1.2.1 mce-tasks-djq

Implémentation Django-Q du worker de gestion des tâches (alternative à Celery).

Ne convenait pas aux besoins.

CHAPTER

TWO

DÉMARRAGE RAPIDE

2.1 Installation avec docker-compose

```
git clone https://github.com/multi-cloud-explorer/mce-django-server.git
cd mce-django-server

docker-compose up -d

# Vérifiez l'état des services
docker-compose ps

# Créez le compte administrateur
docker-compose exec app ./manage.py createsuperuser \
   --username admin --email admin@localhost.net

# Récupérez le login/password de l'administrateur
docker-compose logs app --tail 10
```

2.2 Intégration à votre projet Django existant

TODO..

2.3 Evaluez la solution en toute sécurité

CHAPTER

THREE

INTÉGRATION

3.1 Azure

TODO. . .

- Créez un compte de service Principal
- Test en ligne de commande

CHAPTER

FOUR

STATUS

4.1 mce-django-server

mce-django-server

4.2 mce-django-app

mce-django-app

4.3 mce-tasks-rq

mce-tasks-rq

4.4 mce-lib-azure

mce-lib-azure

4.5 mce-lib-aws

mce-lib-aws

4.6 mce-lib-vsphere

mce-lib-vsphere

4.7 mce-event-push

mce-event-push

CHAPTER

FIVE

MICROSOFT AZURE

CHAPTER

SIX

AMAZON AWS

CHAPTER

SEVEN

MCE_LIB_VSPHERE PACKAGE

7.1 mce_lib_vsphere.core module

CHAPTER

EIGHT

PYVMOMI PACKAGE

8.1 pyVim.connect module

Connect to a VMOMI ServiceInstance.

Detailed description (for [e]pydoc goes here).

pyVim.connect.**Connect**(*host='localhost'*, *port=443*, *user='root'*, *pwd=''*, *service='hostd'*, *adapter='SOAP'*, *namespace=None*, *path='/sdk'*, *connectionPoolTimeout=900*, *version=None*, *keyFile=None*, *certFile=None*, *thumbprint=None*, *sslContext=None*, *b64token=None*, *mechanism='userpass'*)

Connect to the specified server, login and return the service instance object.

Throws any exception back to caller. The service instance object is also saved in the library for easy access.

Clients should modify the service parameter only when connecting to a VMOMI server other than hostd/vpxd. For both of the latter, the default value is fine.

@param host: Which host to connect to. @type host: string @param port: Port @type port: int @param user: User @type user: string @param pwd: Password @type pwd: string @param service: Service @type service: string @param adapter: Adapter @type adapter: string @param namespace: Namespace * **Deprecated: Use version instead** * @type namespace: string @param path: Path @type path: string @param connectionPoolTimeout: Timeout in secs for idle connections to close, specify negative numbers for never

closing the connections

@type connectionPoolTimeout: int @param version: Version @type version: string @param keyFile: ssl key file path @type keyFile: string @param certFile: ssl cert file path @type certFile: string @param thumbprint: host cert thumbprint @type thumbprint: string @param sslContext: SSL Context describing the various SSL options. It is only

supported in Python 2.7.9 or higher.

@type sslContext: SSL.Context @param b64token: base64 encoded token @type b64token: string @param mechanism: authentication mechanism: userpass or sspi @type mechanism: string

pyVim.connect.**ConnectNoSSL**(*host='localhost'*, *port=443*, *user='root'*, *pwd=''*, *service='hostd'*, *adapter='SOAP'*, *namespace=None*, *path='/sdk'*, *version=None*, *keyFile=None*, *certFile=None*, *thumbprint=None*, *b64token=None*, *mechanism='userpass'*)

Provides a standard method for connecting to a specified server without SSL verification. Useful when connecting to servers with self-signed certificates or when you wish to ignore SSL altogether. Will attempt to create an unverified SSL context and then connect via the Connect method.

class pyVim.connect.**Connection**(**args*, ***kwargs*)

Bases: object

pyVim.connect.**Disconnect**(*si*)
Disconnect (logout) service instance @param si: Service instance (returned from Connect)

pyVim.connect.**GetLocalTicket**(*si*, *user*)

pyVim.connect.**GetSi**()
Get the saved service instance.

pyVim.connect.**GetStub**()
Get the global saved stub.

pyVim.connect.**OpenPathWithStub**(*path*, *stub*, *verify=True*)
Open the specified path using HTTP, using the host/port/protocol associated with the specified stub. If the stub has a session cookie, it is included with the HTTP request. Returns the response as a file-like object.

pyVim.connect.**OpenUrlWithBasicAuth**(*url*, *user='root'*, *pwd=''*, *verify=True*)
Open the specified URL, using HTTP basic authentication to provide the specified credentials to the server as part of the request. Returns the response as a file-like object.

pyVim.connect.**SetSi**(*si*)
Set the saved service instance.

pyVim.connect.**SmartConnect**(*protocol='https'*, *host='localhost'*, *port=443*, *user='root'*, *pwd=''*, *service='hostd'*, *path='/sdk'*, *connectionPoolTimeout=900*, *preferredApiVersions=None*, *keyFile=None*, *certFile=None*, *thumbprint=None*, *sslContext=None*, *b64token=None*, *mechanism='userpass'*)
Determine the most preferred API version supported by the specified server, then connect to the specified server using that API version, login and return the service instance object.

Throws any exception back to caller. The service instance object is also saved in the library for easy access.

Clients should modify the service parameter only when connecting to a VMOMI server other than hostd/vpxd. For both of the latter, the default value is fine.

@param protocol: What protocol to use for the connection (e.g. https or http). @type protocol: string @param host: Which host to connect to. @type host: string @param port: Port @type port: int @param user: User @type user: string @param pwd: Password @type pwd: string @param service: Service @type service: string @param path: Path @type path: string @param connectionPoolTimeout: Timeout in secs for idle connections to close, specify negative numbers for never

closing the connections

@type connectionPoolTimeout: int @param preferredApiVersions: Acceptable API version(s) (e.g. vim.version.version3)

If a list of versions is specified the versions should be ordered from most to least preferred. If None is specified, the list of versions support by pyVmomi will be used.

@type preferredApiVersions: string or string list @param keyFile: ssl key file path @type keyFile: string @param certFile: ssl cert file path @type certFile: string @param thumbprint: host cert thumbprint @type thumbprint: string @param sslContext: SSL Context describing the various SSL options. It is only

supported in Python 2.7.9 or higher.

@type sslContext: SSL.Context

pyVim.connect.**SmartConnectNoSSL**(*protocol='https'*, *host='localhost'*, *port=443*, *user='root'*, *pwd=''*, *service='hostd'*, *path='/sdk'*, *connectionPoolTimeout=900*, *preferredApiVersions=None*, *keyFile=None*, *certFile=None*, *thumbprint=None*, *b64token=None*, *mechanism='userpass'*)
Provides a standard method for connecting to a specified server without SSL verification. Useful when connect-

ing to servers with self-signed certificates or when you wish to ignore SSL altogether. Will attempt to create an unverified SSL context and then connect via the SmartConnect method.

class pyVim.connect.**SmartConnection**(**args*, ***kwargs*)
Bases: object

pyVim.connect.**SmartStubAdapter**(*host='localhost'*, *port=443*, *path='/sdk'*, *url=None*, *sock=None*, *poolSize=5*, *certFile=None*, *certKeyFile=None*, *httpProxyHost=None*, *httpProxyPort=80*, *sslProxyPath=None*, *thumbprint=None*, *cacertsFile=None*, *preferredApiVersions=None*, *acceptCompressedResponses=True*, *connectionPoolTimeout=900*, *samlToken=None*, *sslContext=None*)
Determine the most preferred API version supported by the specified server, then create a soap stub adapter using that version

The parameters are the same as for pyVmomi.SoapStubAdapter except for version which is renamed to prefferedApiVersions

@param preferredApiVersions: Acceptable API version(s) (e.g. vim.version.version3) If a list of versions is specified the versions should be ordered from most to least preferred. If None is specified, the list of versions support by pyVmomi will be used.

@type preferredApiVersions: string or string list

class pyVim.connect.**VimSessionOrientedStub**(*soapStub*, *loginMethod*, *retryDelay=0.1*, *retryCount=4*)
Bases: pyVmomi.SoapAdapter.SessionOrientedStub

A vim-specific SessionOrientedStub. See the SessionOrientedStub class in pyVmomi/SoapAdapter.py for more information.

SESSION_EXCEPTIONS = (<class 'pyVmomi.VmomiSupport.vim.fault.NotAuthenticated'>,)

static makeCertHokTokenLoginMethod(*stsUrl*, *stsCert=None*)
Return a function that will call the vim.SessionManager.LoginByToken() after obtaining a HoK SAML token from the STS. The result of this function can be passed as the "loginMethod" to a SessionOrientedStub constructor.

@param stsUrl: URL of the SAML Token issuing service. (i.e. SSO server). @param stsCert: public key of the STS service.

static makeCredBearerTokenLoginMethod(*username*, *password*, *stsUrl*, *stsCert=None*)
Return a function that will call the vim.SessionManager.LoginByToken() after obtaining a Bearer token from the STS. The result of this function can be passed as the "loginMethod" to a SessionOrientedStub constructor.

@param username: username of the user/service registered with STS. @param password: password of the user/service registered with STS. @param stsUrl: URL of the SAML Token issueing service. (i.e. SSO server). @param stsCert: public key of the STS service.

static makeExtensionLoginMethod(*extensionKey*)
Return a function that will call the vim.SessionManager.Login() method with the given parameters. The result of this function can be passed as the "loginMethod" to a SessionOrientedStub constructor.

static makeUserLoginMethod(*username*, *password*, *locale=None*)
Return a function that will call the vim.SessionManager.Login() method with the given parameters. The result of this function can be passed as the "loginMethod" to a SessionOrientedStub constructor.

class pyVim.connect.**closing**(*obj*)
Bases: object

Helper class for using closable objects in a 'with' statement, similar to the one provided by contextlib.

pyVim.connect.**localSslFixup**(*host*, *sslContext*)
Connections to 'localhost' do not need SSL verification as a certificate will never match. The OS provides security by only allowing root to bind to low-numbered ports.

8.2 pyVmomi.VmomiSupport module

pyVmomi.VmomiSupport.**AddBreakingChangesInfo**(*branchName*, *vmodlNamespace*, *count*)

pyVmomi.VmomiSupport.**AddVersionParent**(*version*, *parent*)

class pyVmomi.VmomiSupport.**Array**
Bases: list

pyVmomi.VmomiSupport.**Capitalize**(*str*)

pyVmomi.VmomiSupport.**CheckField**(*info*, *val*)

pyVmomi.VmomiSupport.**CreateAndLoadDataType**(*vmodlName*, *wsdlName*, *parent*, *version*, *props*)

pyVmomi.VmomiSupport.**CreateAndLoadEnumType**(*vmodlName*, *wsdlName*, *version*, *values*)

pyVmomi.VmomiSupport.**CreateAndLoadManagedType**(*vmodlName*, *wsdlName*, *parent*, *version*, *props*, *methods*)

pyVmomi.VmomiSupport.**CreateAndLoadMethodFaultType**()

pyVmomi.VmomiSupport.**CreateArrayType**(*itemType*)

pyVmomi.VmomiSupport.**CreateDataType**(*vmodlName*, *wsdlName*, *parent*, *version*, *props*)

pyVmomi.VmomiSupport.**CreateEnumType**(*vmodlName*, *wsdlName*, *version*, *values*)

pyVmomi.VmomiSupport.**CreateManagedType**(*vmodlName*, *wsdlName*, *parent*, *version*, *props*, *methods*)

class pyVmomi.VmomiSupport.**Curry**(*f*, **args*)
Bases: object

class pyVmomi.VmomiSupport.**DataObject**(***kwargs*)
Bases: object

Array
alias of DataObject[]

class pyVmomi.VmomiSupport.**Enum**
Bases: str

pyVmomi.VmomiSupport.**FinalizeType**(*type*)

pyVmomi.VmomiSupport.**FormatObject**(*val*, *info=<pyVmomi.VmomiSupport.Object object>*, *indent=0*)

pyVmomi.VmomiSupport.**GetBreakingChanges**()

pyVmomi.VmomiSupport.**GetCompatibleType**(*type*, *version*)

pyVmomi.VmomiSupport.**GetHttpContext**()
Get the Http context for the current thread

pyVmomi.VmomiSupport.**GetPropertyInfo**(*type*, *name*)

pyVmomi.VmomiSupport.**GetPythonMethodName**(*wsdlTypeName*, *ns*, *wsdlMethodName*)

pyVmomi.VmomiSupport.**GetPythonTypeName**(*wsdlTypeName*, *ns*)

pyVmomi.VmomiSupport.**GetQualifiedWsdlName**(*type*)

pyVmomi.VmomiSupport.**GetRequestContext**()
Get the RequestContext for the current thread

pyVmomi.VmomiSupport.**GetServiceVersions**(*namespace*)
Get all the versions for the service with specified namespace (partially) ordered by compatibility (i.e. any version in the list that is compatible with some version v in the list will preceed v)

pyVmomi.VmomiSupport.**GetVersionFromVersionUri**(*version*)

pyVmomi.VmomiSupport.**GetVersionNamespace**(*version*)
Get version namespace from version

pyVmomi.VmomiSupport.**GetVersionParents**(*version*)

pyVmomi.VmomiSupport.**GetVersionProps**(*version*)
Get version properties

This function is a fixed version of GetVersion().

pyVmomi.VmomiSupport.**GetVmodlName**(*typ*)
Get vmodl type name from type

pyVmomi.VmomiSupport.**GetVmodlNs**(*version*)

pyVmomi.VmomiSupport.**GetVmodlType**(*name*)
Get type from vmodl name

pyVmomi.VmomiSupport.**GetWsdlMethod**(*ns*, *wsdlName*)
Get wsdl method from ns, wsdlName

pyVmomi.VmomiSupport.**GetWsdlMethodName**(*pythonTypeName*, *pythonMethodName*)

pyVmomi.VmomiSupport.**GetWsdlName**(*type*)

pyVmomi.VmomiSupport.**GetWsdlNamespace**(*version*)
Get wsdl namespace from version

pyVmomi.VmomiSupport.**GetWsdlType**(*ns*, *name*)

pyVmomi.VmomiSupport.**GetWsdlTypeName**(*pythonTypeName*)

pyVmomi.VmomiSupport.**GetWsdlTypes**()

pyVmomi.VmomiSupport.**GuessWsdlMethod**(*name*)

pyVmomi.VmomiSupport.**GuessWsdlType**(*name*)

pyVmomi.VmomiSupport.**InverseMap**(*map*)

class pyVmomi.VmomiSupport.**LazyModule**(*name*)
Bases: object

class pyVmomi.VmomiSupport.**LazyObject**(***kwargs*)
Bases: *pyVmomi.VmomiSupport.Object*

class pyVmomi.VmomiSupport.**LazyType**
Bases: type

class pyVmomi.VmomiSupport.**Link**
Bases: str

class pyVmomi.VmomiSupport.**LinkResolver**(*scope*)
Bases: object

ResolveLink(*key*)

ResolveLinks(*keys*)

pyVmomi.VmomiSupport.**LoadDataType**(*vmodlName*, *wsdlName*, *parent*, *version*, *props*)

pyVmomi.VmomiSupport.**LoadEnumType**(*vmodlName*, *wsdlName*, *version*, *values*)

pyVmomi.VmomiSupport.**LoadManagedType**(*vmodlName*, *wsdlName*, *parent*, *version*, *props*, *methods*)

class pyVmomi.VmomiSupport.**ManagedMethod**(*info*)
Bases: *pyVmomi.VmomiSupport.Curry*

class pyVmomi.VmomiSupport.**ManagedObject**(*moId*, *stub=None*, *serverGuid=None*)
Bases: object

Array
alias of ManagedObject[]

class pyVmomi.VmomiSupport.**Object**(***kwargs*)
Bases: object

class pyVmomi.VmomiSupport.**PropertyPath**
Bases: str

pyVmomi.VmomiSupport.**ResolveLink**(*key*, *obj*)

pyVmomi.VmomiSupport.**ResolveLinks**(*keys*, *obj*)

pyVmomi.VmomiSupport.**SetAttr**()
Implement setattr(self, name, value).

class pyVmomi.VmomiSupport.**StringDict**(**args*, ***kwargs*)
Bases: dict

String only dictionary: same as dict, except it only accept string as value

dict in python is kind of strange. U cannot just override __setitem__, as __init__, update, and setdefault all bypass __setitem__. When override, we have to override all three together

setdefault(*key*, *val=None*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*[E]*, ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

pyVmomi.VmomiSupport.**Type**(*obj*)

pyVmomi.VmomiSupport.**TypeDefExists**(*name*)

class pyVmomi.VmomiSupport.**URI**
Bases: str

class pyVmomi.VmomiSupport.**UncallableManagedMethod**(*name*)
Bases: *pyVmomi.VmomiSupport.ManagedMethod*

pyVmomi.VmomiSupport.**Uncapitalize**(*str*)

pyVmomi.VmomiSupport.**UncapitalizeVmodlName**(*str*)

exception pyVmomi.VmomiSupport.**UnknownWsdlTypeError**
Bases: KeyError

class pyVmomi.VmomiSupport.**VmomiJSONEncoder**(**args*, ***kwargs*)
Bases: json.encoder.JSONEncoder

Custom JSON encoder to encode vSphere objects.

When a ManagedObject is encoded, it gains three properties: _vimid is the _moId (ex: 'vm-42') _vimref is the moRef (ex: 'vim.VirtualMachine:vm-42') _vimtype is the class name (ex: 'vim.VirtualMachine')

When a DataObject is encoded, it gains one property: _vimtype is the class name (ex: 'vim.VirtualMachineQuestionInfo')

If the dynamicProperty and dynamicType are empty, they are optionally omitted from the results of DataObjects and ManagedObjects

@example "Explode only the object passed in" data = json.dumps(vm, cls=VmomiJSONEncoder)

@example "Explode specific objects"

data = json.dumps(vm, cls=VmomiJSONEncoder, explode=[vm, vm.network[0]])

@example "Explode all virtual machines in a list and their snapshots"

data = json.dumps([vm1, vm2], cls=VmomiJSONEncoder,

explode=[templateOf('VirtualMachine'), templateOf('VirtualMachineSnapshot')])

default(*obj*)
Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

explode(*obj*)
Determine if the object should be exploded.

pyVmomi.VmomiSupport.**arrayType**
alias of pyVmomi.VmomiSupport.PropertyPath[]

class pyVmomi.VmomiSupport.**binary**
Bases: bytes

class pyVmomi.VmomiSupport.**byte**
Bases: int

class pyVmomi.VmomiSupport.**double**
Bases: float

class pyVmomi.VmomiSupport.**long**
Bases: int

class pyVmomi.VmomiSupport.**short**
Bases: int

pyVmomi.VmomiSupport.**templateOf**(*typestr*)
Returns a class template.

8.3 vim.VirtualMachine class

CHAPTER

NINE

FONCTIONNALITÉS

- [x] Inventaire automatique des resources
- [x] Historisation des changements au format Json Patch
- [ ] Push des évènements vers une queue de donnée ou un WebHook

CHAPTER

TEN

PROVIDERS

- [x] Azure
- [ ] AWS
- [ ] GCP
- [ ] VMware

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

I

L

M

O

P

R

S

T

U

V