1. SDDC Import/Export for VMware Cloud on AWS

1.1. Table of Contents

1.2. Overview

The SDDC Import/Export for VMware Cloud on AWS tool enable customers to save and restore their VMware Cloud on AWS (VMC) Software-Defined Data Center (SDDC) configuration.

There are many situations when customers want to migrate from an existing SDDC to a different one. While HCX addresses the data migration challenge, this tool offers customers the ability to copy the configuration from a source to a destination SDDC.

A few example migration scenarios are:

- SDDC to SDDC migration from bare-metal (i3) to a different bare-metal type (i3en)
- SDDC to SDDC migration from VMware-based org to an AWS-based org
- SDDC to SDDC migration from region (i.e. London) to a different region (i.e. Dublin).

Other use cases are:

- Backups - save the entire SDDC configuration
- Lab purposes - customers or partners might want to deploy SDDCs with a pre-populated configuration.
- DR purposes - deploy a pre-populated configuration in conjunction with VMware Site Recovery or VMware Cloud Disaster Recovery

1.3. Getting Started

1.3.1. Install Python

This tool is dependent on Python3, you can find installation instructions for your operating system in the Python documentation.

1.3.2. Download code

If you know git, clone the repo with

```
git clone https://github.com/vmware-samples/sddc-import-export-for-vmware-cloud-on-aws.git
```

If you don't know git, you can download the code from the Flings site

1.3.3. Install Python modules and packages

When you navigate to the sddc_import_export folder, you will find a requirements.txt file that list all your Python packages. They can all be installed by running the following command on Linux/Mac:

```
pip3 install -r requirements.txt
```

On Windows, use

```
python -m pip install -r requirements.txt
```

If you get pip installation errors complaining about being unable to install `git+https://github.com/vmware/vsphere-automation-sdk-python.git@v7.0.2.0`, and you don't need to use the vCenter folder export feature, you can do the following:

- Remove line 10 in requirements.txt `git+https://github.com/vmware/vsphere-automation-sdk-python.git@v7.0.2.0`
- Comment out line 54 in sddc_import_export.py `import vcenter`

You must uncomment these lines in order to use the vCenter folder export feature. Installing the required library requires you to have git installed on your local system.

1.3.4. Update vmc.ini

Version 1.1 introduces the gov_cloud_urls flag in vmc.ini. The default value is False - change this value to True if you are using GovCloud.

Access to the VMware Cloud on AWS API is dependent on a refresh token. To generate a token for your account, see the Generate API Tokens help article.

The Org ID and SDDC ID can be found on the Support tab of your SDDCs.

```
# Refresh tokens generated in the VMC console. Users have a separate token in each
org
source_refresh_token    = XXXXXXXXXXXXXXX
dest_refresh_token      = XXXXXXXXXXXXXXX

# Organization and SDDC IDs are easily found in the support tab of any SDDC
source_org_id           = XXXXXXXXXXXXXXX
source_sddc_id          = XXXXXXXXXXXXXXX
dest_org_id             = XXXXXXXXXXXXXXX
dest_sddc_id            = XXXXXXXXXXXXXXX
```

The vmc.ini configuration can also be passed via command line. Use sddc_import_export --help for syntax.

1.3.5. Update config.ini

Config.ini contains configuration sections for import and export.

There are True/False flags that can be set for each configuration option. The default configuration enables all options.

For example, in this section of the configuration, the compute gateway networks would be exported, but the public IP and NAT associations would not be exported.

```
# Export the networks configured on the compute gateway?
network_export  = True
network_export_filename = cgw-networks.json

# Export the list of public IP addresses?
public_export = False
public_export_filename = public.json

# Export the NAT rules, including Public IP addresses?
nat_export = False
nat_export_filename = natrules.json
```

1.3.5.1. Sync Mode

An experimental feature was introduced in version 1.2. The configuration flag is named sync_mode, and is found in the importConfig section of config.ini. *Note* - sync does not support public IP or NAT configurations. It also does not support syncing deletions. It only syncs new object creations or existing object updates.

```
[importConfig]

# Set this to True if you want to do continuous sync operations i.e. a periodic
sync of
# DFW rules from a source to a destination SDDC. The default method of import
operations
# is a PUT. Setting this flag to True changes the method to a PATCH
# Not all settings are supported for sync - public IP and NAT mapping are
unsupported
sync_mode = False
```

The default value is False. When set to true, existing objects in the destination SDDC will be overwritten with values from the exported data. This feature is handy if you have multiple SDDCs for identical purposes, such as desktop clusters, and want to push identical firweall rules to all SDDCs. Not all settings are supported for sync - public IP and NAT mapping are unsupported.

1.3.5.2. Exclude List Filtering

Version 1.3 introduced the ability to filter out objects during an import. The following objects can be filtered:

- CGW firewall rule
- CGW firewall group
- CGW network segment
- MGW firewall rule
- MGW firewall group

The exclude filter performs a Python regex match on the display name of the object. The default is to have no filter. The CGW exclude list section of config.ini is shown below

```
# Python regex match on CGW group display name, pipe-delimited. See README for
examples.
cgw_groups_import_exclude_list =
# Python regex match on CGW rule display name, pipe-delimited. See README for
examples.
cgw_import_exclude_list =
```

You must be careful not to filter out a group that a firewall rule is dependent on. The tool does not enforce group dependencies. If you filter out a group that a firewall rule uses, the firewall rule will fail to import.

1.3.5.2.1. Filtering Examples

Exclude all groups that begin with 'abcd' or 'efgh'

```
cgw_groups_import_exclude_list = abcd*|efgh*
```

Exclude all network segments that begin with L2E (HCX extended segements)

```
network_import_exclude_list = L2E*
```

A sample config file for VCDR is included in this repository.

1.3.6. Update aws.ini (optional)

If you want to use the optional feature to archive exported zipfiles to S3 storage, you must update aws.ini with a bucket name and credentials with write access to the bucket. You must also set the export_type value in config.ini to 's3'.

```
[awsConfig]

aws_s3_export_access_id = ""
aws_s3_export_access_secret = ""
aws_s3_export_bucket = ""
```

The aws.ini configuration can also be passed via command line. Use sddc_import_export --help for syntax.

1.3.7. Update vcenter.ini (optional)

If you want to use the optional feature to sync your on-prem vCenter folder structure to VMC, you must update vcenter.ini with the appropriate URLs, credentials, and Datacenter name. The tool can only export and import a single datacenter object. The required libraries are commented out by default in `requirements.txt`, and line 54 of `sddc_import_export.py` is also commented out `import vmc`. You must uncomment both lines and run the requirements.txt installation again for dependencies. This library is dependent on git being installed on your local system.

```
[vCenterConfig]

srcvCenterURL = on-prem-vcenter.domain.com
srcvCenterUsername = administrator@domain.com
srcvCenterPassword = x
srcvCenterDatacenter = Datacenter-Name
srcvCenterSSLVerify = False

destvCenterURL = vcenter.sddc-xx-xx-xx-xx.vmwarevmc.com
destvCenterUsername = cloudadmin@vmc.local
destvCenterPassword = x
destvCenterDatacenter = SDDC-Datacenter
destvCenterSSLVerify = True
```

1.4. Running the script

1.4.1. Export

Export will export your existing SDDC configuration from your source SDDC to a set of files that can be subsequently used for import.

Run the following command to export:

```
python3 sddc_import_export.py -o export
```

If all of the export options are enabled, this will export a set of files:

- Services.json
- cgw_groups.json
- cgw-networks.json
- cgw.json
- dfw_details.json
- dfw.json
- mgw_groups.json
- mgw.json
- natrules.json
- public.json
- s3-service_access.json
- sddc_info.json
- service_access.json
- services.json
- vpn-bgp.json
- vpn-dpd.json
- vpn-ike.json
- vpn-l2.json
- vpn-l3.json
- vpn-local-bgp.json

- vpn-tunnel.json

A config.ini flag named 'export_history' allows for the JSON files to be zipped for archival purposes. A related configuration option named 'max_export_history_files' lets you control how many zipped archive files are retained.

Export is read-only and will not make any changes to your source SDDC.

1.4.1.1. Exclusions

Manual VM memberships in groups are not supported, they will be filtered out of the export.

1.4.2. Import

Import will import your saved SDDC configuration from a set of exported JSON files to a destination SDDC.

Run the following command to import:

```
python3 sddc_import_export.py -o import
```

Before making changes, the script will prompt you for confirmation

```
Script is running in live mode - changes will be made to your destination SDDC.
Continue? (y/n): y
```

There are two sections of the config.ini file that control whether the script will make changes to your destination SDDC. The first is import_mode. If import_mode is set to live, changes will be made to the destination SDDC. There is one final flag - import_live_mode_warning. If this flag is set to true, you will be warned that the script is in live mode and given the option to switch back to test mode. If you want to run the script repeatedly and are absolutely sure of your configuration, you can set the import_mode_live_warning flag to False, this will enable the script to run in live mode without user intervention.

```
[importConfig]
# Import mode
# import_mode = test
#  - No changes will be made to the destination SDDC
#
# import_mode = live
#  - Changes will be made to the destination SDDC
import_mode = live

# Script will warn, ask for a confirmation before continuing in live mode
# Set this to false if you are absolutely sure you have your script configured
correctly and want to run it automatically
import_mode_live_warning = True
```

Example of a successful import:

```
nvibert-a01:sddc_import_export nicolasvibert$ python3 sddc_import_export.py -o
import
Import mode: live
Importing data into org VMC-SET-TEST (b7793958-b6b6-4916-a008-40c5c47ec24c), SDDC
AVI-LONDON-SDDC (1eadc044-a195-4f43-8bbd-fa1089544e6d)
Script is running in live mode - changes will be made to your destination SDDC.
Continue? (y/n): y
Live import will proceed
Beginning CGW network import:
Added sddc-cgw-network-1
Import results:
+--------------------+---------+-------------+--------------------+
|    Display Name    |  Result | Result Note |     Segment ID     |
+--------------------+---------+-------------+--------------------+
| sddc-cgw-network-1 | SUCCESS |             | sddc-cgw-network-1 |
+--------------------+---------+-------------+--------------------+
Beginning Services import
Service nico has been imported.
Service nico has been imported.
Service nico3 has been imported.
Beginning CGW import...
CGW Group Blue_VMs has been imported.
CGW Group Red_VMs has been imported.
CGW Group tf-group12 has been imported.
CGW Group tf-group13 has been imported.
CGW Group tf-group14 has been imported.
Firewall Rule VMC to AWS has been imported.
Firewall Rule AWS to VMC has been imported.
Firewall Rule Internet out has been imported.
Firewall Rule Default VTI Rule has been imported.
Beginning MGW import...
Firewall Rule ESXi Provisioning has been imported.
Firewall Rule vCenter Inbound has been imported.
Firewall Rule ESXi Outbound has been imported.
Firewall Rule vCenter Outbound has been imported.
Beginning Public IP export...
Previous IP 44.242.3.1 has been imported and remapped to 3.10.11.226.
Previous IP 34.215.71.251 has been imported and remapped to 3.11.245.46.
Previous IP 54.184.3.133 has been imported and remapped to 18.134.166.17.
Beginning NAT import...
NAT Rule test-nat-rule has been imported.
NAT Rule test-nat-second-rule has been imported.
NAT Rule test-nat-third-rule has been imported.
Import has been concluded. Thank you for using SDDC Import/Export for VMware Cloud
on AWS.
```

1.4.3. Export-Import

Some customers want to run export from source, then immediately import into the destination. This option is useful if you want to set the script on a schedule to keep 2 SDDCs in sync.

Run the following command to run an export-import:

```
python3 sddc_import_export.py -o export-import
```

1.4.4. Export NSX-T on-prem

To export your DFW configuration, first edit the nsxConfig section of vcenter.ini with your NSX-T manager URL and credentials.

```
[nsxConfig]

srcNSXmgrURL =
srcNSXmgrUsername =
srcNSXmgrPassword =
srcNSXmgrSSLVerify = False
```

Then, run the export-nsx command

```
python3 sddc_import_export.py -o export-nsx
```

1.4.5. Import NSX-T on-prem

To export your on-prem DFW configuration into VMC on AWS, first run the export-nsx function.

After the export is complete, ensure that you have vmc.ini configured with a dest_refresh_token, dest_org_id, and dest_sddc_id. The import-nsx process uses the code as a standard import from an SDDC, including all of the options to enable and disable sections in config.ini

Finally, run import-nsx command.

```
python3 sddc_import_export.py -o import-nsx
```

1.4.6. Export vCenter

To export your vCenter server folder structure, set the export_vcenter_folders flag in config.ini to True. Then run the export command:

```
python3 sddc_import_export.py -o export-vcenter
```

1.4.7. Import vCenter

To Import your vCenter server folder structure, set the import_vcenter_folders flag in config.ini to True. Then run the import command:

```
python3 sddc_import_export.py -o import-vcenter
```

1.4.8. Import from zip archive

If you enable the 'export_history' flag, a zipfile containing all of the exported JSON will be exported into the /json folder. You can pass the filename to the script as shown to use it as the import source.

```
python3 sddc_import_export.py -o import -i json/path-to_json-export.zip
```

1.4.9. Running S3 export as a Lambda function

Install all required packages to a folder

```
mkdir python_req
cd python_req
pip3 install --target . -r ../requirements.txt
```

Zip python_req and upload it to a Lambda layer

Change export_folder in config.ini to /tmp, because /tmp is the only writable folder in Lambda

Ensure you have configured aws.ini with your S3 bucket settings

Ensure that you have granted the execution role write permissions to your S3 bucket

Add the following files individually to the function code, or zip them up and upload all at once:

- config_ini/*
- lambda_handler.py
- sddc_import_export.py
- VMCImportExport.py

Change the Handler runtime settings to invoke_lambda.lambda_handler

Execute your Lambda function. Although it is possible to configure values in the config_ini files that you upload to the function code, it might be preferable to pass the required values via command line argument. See invoke_lambda.py for an example.

1.4.10. Cloud Services Platform Role Sync

You can sync user roles with the rolesync option.

> **Note:** the API tokens you use must have Org Owner permissions. You can sync across orgs by using a different source and destination OrgID. Alternatively, if your sync template and destinations are in the same org, you can configure the source and destination OrgIDs to be identical.

> **Known issue:** If the source user has roles assigned that do not exist in the destination, the sync will fail. The initial release of this feature does not have any logic in it to search for invalid roles. For example: a source user is assigned a role for both VMC on AWS and vRealize Automation Cloud. The destination org only has VMC on AWS activated - vRealize Automation is not activated in the destination org. The sync will fail. But the sync would succeed if the source user only had VMC on AWS roles assigned.

> **Feature Limits** In the initial release of this feature, the sync is an addition - any roles assigned to the source will be added to the destination. It does not delete any roles in the destination.

First, configure a template account in the Cloud Services Platform, granting it all of the roles you want to synchronize.

Next, configure the `role_sync_source_user_email` poperty in `config.ini` with the email address of your template account - for this example, we set it to template@vmware.com.

Then, configure the `role_sync_dest_user_emails` property in `config.ini` with the email address(es) of your destination accounts. These accounts must already exist for the sync to work.

You can then run the sync:

```
python3 sddc_import_export.py -o rolesync
```

Example output:

```
Looking up template user template@vmware.com
Looking up destination user destination@vmware.com
Role sync success: template@vmware.com->destination@vmware.com
```

You can also run the sync directly from the command line without configuring `config.ini`:

```
python3 sddc_import_export.py -o rolesync -rss template@vmware.com -rsd
user1@vmware.com,user2@vmware.com
```

Example output:

```
loaded role sync source user email from command line
Loaded role sync dest user emails from command line
Looking up template user template@vmware.com
Looking up destination user user1@vmware.com
Role sync success: template@vmware.com->user1@vmware.com
```

```
Looking up destination user user2@vmware.com
Role sync success: template@vmware.com->user2@vmware.com
```

1.4.11. Testbed commands

The testbed command lets you create a number of test objects. This is useful when experimenting with API limits in VMC on AWS.

The testbed command always operates on the destination SDDC. It obeys the `import_mode` setting in `config.ini`.

This will create 1,500 test Compute Gateway groups, starting with `cgw-test-group-0000`.

```
python sddc_import_export.py -o testbed --test-name create-cgw-groups --num-
objects 1500
```

This will delete 1,500 test Compute Gateway groups, starting with `cgw-test-group-0000`

```
python sddc_import_export.py -o testbed --test-name delete-cgw-groups --num-
objects 1500
```

You can also pass the --start-num argument. The default is 0. This will create 1,500 test Compute Gateway groups, starting with `cgw-test-group-0050`

```
python sddc_import_export.py -o testbed --test-name create-cgw-groups --num-
objects 1500 --start-num 50
```

This will delete 1,500 test Compute Gateway groups, starting with `cgw-test-group-0050`

```
python sddc_import_export.py -o testbed --test-name delete-cgw-groups --num-
objects 1500 --start-num 50
```

This will delete ALL CGW GROUPS. Use with extreme caution.

```
python sddc_import_export.py -o testbed --test-name delete-all-cgw-groups
```