Configuration Import/Export

Version Requirement: To use configuration import/export, you must be running the threat defense version 6.5(0) or higher, and the threat defense REST API v4 or higher.

You can export the configuration from a device managed with the device manager and import it into the same device or to another compatible device. For example, you can use configuration import/export to replicate a baseline configuration across multiple similar devices, then use the device manager on each device to configure the characteristics unique to each device.

About Configuration Import/Export

When you manage the threat defense device locally, with the device manager or through the Security Cloud Control, you can export the configuration of the device using the threat defense API. This method does not work with a device managed by the Secure Firewall Management Center.

When you export the configuration, the system creates a zip file. You can then download the zip file to your workstation. The configuration itself is represented as objects defined using attribute-value pairs in a JSON-formatted text file. You can edit the file prior to importing it back into the same device or a different device.

Thus, you can use an export file to create a template that you can deploy to other devices in your network.

When importing objects, you also have the option of defining the objects directly in the import command rather than in a configuration file. However, you should directly define objects only in cases where you are importing a small number of changes.

The following topics explain more about configuration import/export.

What is Included in the Export File

When you do an export, you specify which configurations to include in the export file. A full export includes everything in the export zip file. Based on what you choose to export, the export zip file might include the following:

- Attribute-value pairs that define each configured object. All configurable items are modeled as objects, not just those that are called "objects" in the device manager.

- If you configured remote access VPN, the AnyConnect packages and any other referenced files, such as client profile XML files, the DAP XML file, and Hostscan packages.
- If you configured custom file policies, any referenced clean list or custom detection list.

Comparing Import/Export and Backup/Restore

Configuration import/export is not the same as backup/restore.

- Backup/restore is for disaster recovery. You can restore a backup to a device only if the device is the same model, and running the same software version, as the device from which the backup was taken. Primarily, this is for recovering the “last good” configuration to the same device, or to restore the configuration to a replacement device.
- Import/export is for preserving all or part of a configuration. You can use an export file to restore the configuration to a device after you reimage it. Or, you can use the export file as a template, editing the contents before importing it into another device. With import/export, you can quickly get a new device up to a certain baseline configuration, so you can deploy it more rapidly into your network. Within limits, you can even import a file to different device models, for example, from a Firepower 2120 to a 2130. If the import file only includes objects that are supported on all device models, there should be very few restrictions on import. The one restriction is that the device needs to use the same API version used for the export file.

Strategies for Import/Export

Following are some ways you can use import/export.

- **Create a template for new devices.** Configure your model device to the baseline you need, then export the full configuration. Subsequently, you can import that configuration into new devices, then use the device manager or the threat defense API to make whatever modifications are needed. You can also edit the template prior to import to make these modifications, for example, to the IP addresses for each interface. Note that the full export includes the ManagementIP object (type=managementip); assuming that you have already configured the management address and gateway on the target device, you should remove this object from the export file when you create the template for the new device, or you will overwrite the management addressing information.
- **Deploy configuration changes from one device to other similar devices.** For example, when editing the configuration of device A, you create a few new network objects and access control rules. You can then export the pending changes, and import those changes into device B. After you deploy the configuration on both devices, they are running the same new rules.
- **Reapply the configuration after a system reimage.** Reimaging a device erases the configuration. If you first export the full configuration, you can them import it after you complete the reimage.
- **Apply targeted configurations.** Because you can edit or even manually create an export file, you can remove all objects except those you want to import into another device. For example, you could create a configuration file that contains a set of network objects, and use it to import the same group of network objects into all of your threat defense devices.

Guidelines for Configuration Import/Export

- During an export job, the system holds a write lock on the configuration database. You cannot use the API, or the device manager, to make configuration changes until the job completes. However, you can view the configuration in the device manager, or use GET calls in the API, during the export job.
- During an import job, the system holds both read and write locks on the configuration database. You cannot use the API or the device manager to view the configuration or make changes to it until the job completes.
- The imported configuration is added to the existing configuration. You cannot wipe away the device's configuration and replace it with the imported configuration. If you need to reset the device configuration prior to import, you can go to the device CLI and issue the **configure manager delete** command, followed by the **configure manager local** command. Only the management interface configuration will be preserved.
- You can import a file into a device only if the device is running the same API version as defined in the apiVersion attribute in the metadata object contained in the file.
- The SRU version must be the same on the export and import devices, or the import will fail.

Importing and Exporting Configurations

The import/export process starts with exporting the configuration from a locally-managed device. You can then download the export file, and optionally edit it, before uploading it into the same device or a compatible device. The following topics explain each step.

Export the Configuration

Use the POST /action/configexport method to create and start a configuration export job.

Procedure

Step 1 Create the JSON object body for the export job.

Following is an example of the JSON object to use with this call.

```
{
  "diskFileName": "string",
  "encryptionKey": "*********",
  "doNotEncrypt": false,
  "configExportType": "FULL_EXPORT",
  "deployedObjectsOnly": true,
  "entityIds": [
    "string"
  ],
  "jobName": "string",
  "type": "scheduleconfigexport"
}
```

The attributes are:

- **diskFileName**—(Optional.) The name of the export zip file. If you do not specify a name, the system generates one for you. Even if you specify a name, the system might append characters to the name to ensure uniqueness. The name has a maximum length of 60 characters.
- **encryptionKey**—(Optional.) An encryption key for the zip file. If you do not want to encrypt the file, omit this field and specify "doNotEncrypt": true instead. If you specify a key, you will need to use the key to open the zip file after you download it to your workstation. Note that the exported configuration file exposes secret keys, passwords, and other sensitive data in clear text (because otherwise they cannot be imported), so you might want to apply an encryption key to protect sensitive data. The system uses AES 256 encryption.
- **doNotEncrypt**—(Optional.) Whether the export file should be encrypted (false), or not encrypted (true). The default is false, which means you must specify a non-empty encryptionKey attribute. If you specify true, then the encryptionKey attribute is ignored.
- **configExportType**—One of the following enum values:
    - **FULL_EXPORT**—Include the entire configuration in the export file. This is the default.
    - **PARTIAL_EXPORT**—Include only those objects, and their descendant objects, that are identified in the entityIds list. Unexportable objects are not included even if you specify their identities. All user-defined objects are exportable.
    - **PENDING_CHANGE_EXPORT**—Include only those objects that have not yet been deployed, that is, the pending changes.
- **deployedObjectsOnly**—(Optional.) Whether to include objects in the export file only if they have been deployed. That is, do not include pending changes. This attribute is ignored for PENDING_CHANGE_EXPORT jobs, because those jobs include undeployed objects only. The default is false, which means all pending changes are included in the export. Specify true to exclude pending changes.
- **entityIds**—A comma-separated list of the identities of a set of starting-point objects, enclosed in [brackets]. This list is required for a PARTIAL_EXPORT job. Each item in this list could be either a UUID value or an attribute-value pair matching patterns like "**id=***uuid-value*", "**type**=*object-type*" or "**name**=*object-name*". For example, **"type=networkobject"**.

  The **type** can be either a leaf entity, such as networkobject, or an alias of a set of leaf types. Some typical type aliases are: network (NetworkObject and NetworkObjectGroup), port (all TCP/UDP/ICMP port, protocol and group types), url (URL objects and groups), ikepolicy (IKE V1/V2 policies), ikeproposal (Ike V1/V2 proposals), identitysource (all identity sources), certificate (all certificate types), object (all object/group types that would be listed in the device manager on the Objects page), interface (all network interfaces, s2svpn (all site-to-site VPN related types), ravpn (all RA VPN related types), vpn (both s2svpn and ravpn).

  All of these objects and their outgoing referential descendants will be included in the PARTIAL_EXPORT output file. Note all the unexportable objects will be excluded from the output even if you specify their identities. Use the GET method for the appropriate resource types to obtain the UUIDs, types, or names for the target objects.

  For example, to export all network objects, plus an access rule named myaccessrule, and two objects identified by UUID, you can specify:

  ```
  "entityIds": [
  ```

```
    "type=networkobject",
    "id=bab3e3cd-8c70-11e9-930a-1f12ee87d473",
    "name=myaccessrule",
    "acc2e3cd-8c70-11e9-930a-1f12ee87b286"
    ]",
```

- **jobName**—(Optional.) A name for the export job. Giving the job a name might make it easier to find it when you retrieve job status.
- **type**—The job type, which is always **scheduleconfigexport**.

Example:

The following example performs a full export to the file export-config-1 and accepts the defaults for all other attributes:

```
{
  "diskFileName": "export-config-1",
  "doNotEncrypt": true
  "configExportType": "FULL_EXPORT",
  "type": "scheduleconfigexport"
}
```

Step 2 Post the object.

For example, the curl command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
 -d '{ \
   "configExportType": "FULL_EXPORT", \
   "type": "scheduleconfigexport" \
 }' 'https://10.89.5.38/api/fdm/latest/action/configexport'
```

Step 3 Verify the response.

You should get a response code of 200. A successful response body would look something like the following if you posted the minimum JSON object. If you specify an encryption key, it is masked in the response.

```
{
  "version": null,
  "scheduleType": "IMMEDIATE",
  "user": "admin",
  "forceOperation": false,
  "jobHistoryUuid": "c7a8ba61-629a-11e9-8b8d-0fcc3c9d6d0b",
  "ipAddress": "10.24.5.177",
  "diskFileName": "export-config-1",
  "encryptionKey": null,
  "doNotEncrypt": true
  "configExportType": "FULL_EXPORT",
  "deployedObjectsOnly": false,
  "entityIds": null,
  "jobName": "Config Export",
  "id": "c79be920-629a-11e9-8b8d-85231be77de0",
  "type": "scheduleconfigexport",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest
/action/configexport/c79be920-629a-11e9-8b8d-85231be77de0"
  }
}
```

Check the Status of the Export Job

It takes some time for an export job to complete. The larger the configuration, the more time the job will require. Check the job status to ensure it completes successfully before you try to download the file.

The simplest way to get status is to use GET /jobs/configexportstatus. For example, the curl command would look like the following:

```
curl -X GET --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/jobs/configexportstatus'
```

A successfully completed job would return status similar to the following.

```
{
  "version": "hdy62yf5xp3vf",
  "jobName": "Config Export",
  "jobDescription": null,
  "user": "admin",
  "startDateTime": "2019-04-19 13:14:54Z",
  "endDateTime": "2019-04-19 13:14:56Z",
  "status": "SUCCESS",
  "statusMessage": "The configuration was exported successfully",
  "scheduleUuid": "1ef502ad-62a5-11e9-8b8d-074ebc750708",
  "diskFileName": "export-config-1.zip",
  "messages": [],
  "configExportType": "FULL_EXPORT",
  "deployedObjectsOnly": false,
  "entityIds": null,
  "id": "1f0aad8e-62a5-11e9-8b8d-bb1ebb4d1300",
  "type": "configexportjobstatus",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest
/jobs/configexportstatus/1f0aad8e-62a5-11e9-8b8d-bb1ebb4d1300"
  }
}
```

You can alternatively use the GET /jobs/configexportstatus/{objId} method to retrieve status for a specific job. You would get the object ID from the **id** field in the response object.

Download the Export File

When an export job completes, the export file is written to the system disk and is called a configuration file. You can download this export file to your workstation using the GET /action/downloadconfigfile/{objId} method. To get a list of the available files, use the GET /action/configfiles method.

Note With GET /action/downloadconfigfile/{objId} you typically specify the file name as the object ID. Alternatively, you can specify the ID of the ConfigExportStatus object associated with the file.

Procedure

Step 1 Get a list of the configuration files on the disk.

The list of configuration files includes export files and any files that you uploaded for import.

The curl command would be similar to the following:

```
curl -X GET --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/action/configfiles'
```

The response would show a list of items, each of which is a configuration file. For example, the following list shows 2 files. Note that the **id** for all files is default. Ignore the ID, and use the **diskFileName** instead.

```
{
  "items": [
    {
      "diskFileName": "export-config-2.zip",
      "dateModified": "2019-04-19 13:32:28Z",
      "sizeBytes": 10182,
      "id": "default",
      "type": "configimportexportfileinfo",
      "links": {
        "self": "https://10.89.5.38/api/fdm/latest/action/configfiles/default"
      }
    },
    {
      "diskFileName": "export-config-1.zip",
      "dateModified": "2019-04-19 13:14:56Z",
      "sizeBytes": 10083,
      "id": "default",
      "type": "configimportexportfileinfo",
      "links": {
        "self": "https://10.89.5.38/api/fdm/latest/action/configfiles/default"
      }
    }
  ],
```

Step 2 Download the file using the diskFileName as the object ID.

The curl command would be similar to the following:

```
curl -X GET --header 'Accept: application/octet-stream'
'https://10.89.5.38/api/fdm/latest/action/downloadconfigfile/export-config-2.zip'
```

The file is downloaded to your default downloads folder. If you are issuing the GET method from the API Explorer, and your browser is configured to prompt for download location, you will be prompted to save the file.

A successful download will result in a 200 return code and no response body.

Edit the Exported Configuration File

After you download the configuration file, you can unzip it and open the text file that contains the objects. WordPad formats the content in an easier to read fashion than NotePad. You can also use other text editors that you might have installed. You can even create your own configuration file from scratch, but you will need to export the configuration to understand the file structure.

The following topics explain the requirements for the text file.

Minimum Configuration File Requirements

A configuration file must have the following minimum elements:

- Enclose the objects in the file within [brackets]. The entire file uses standard JSON notation and is an array of objects.
- Enclose each object in {braces}.
- Use commas to separate the objects in the configuration file. That is, the end brace of an object should be followed by a comma except for the final object.
- The first object in the file must be a metadata object. The easiest way to get the right object attributes is to export the configuration from a device of the desired model. For example, following is the metadata object from a Secure Firewall Threat Defense Virtual device. Before importing the device, you can edit the configuration and export types, and if desired, delete the generatedOn attribute.

```
{"hardwareModel":"Cisco Firepower Threat Defense for VMWare",
 "type":"metadata",
 "configType":"FULL_CONFIG",
 "apiVersion":"latest",
 "generatedOn":"Fri Apr 19 13:32:28 UTC 2019",
 "exportType":"FULL_EXPORT",
 "softwareVersion":"6.5.0-10480"}
```

- The metadata object must specify the appropriate configuration type (configType) value.
  - FULL_CONFIG—This text file includes the full device configuration.
  - DELTA_CONFIG—This text file includes a partial configuration, perhaps even just a few objects.
- The exportType is one of the following: FULL_EXPORT, PARTIAL_EXPORT, PENDING_CHANGE_EXPORT.
- If you are doing a full configuration import, the metadata object must specify the following attributes: hardwareModel, softwareVersion, apiVersion.
- You can write objects on one line or on multiple lines, but do not put empty lines or comment lines between the attributes in an object. Comments are not allowed in the file.
- Although objects are exported in dependency order, where an object referred to by another object is defined first, maintaining that order in an import configuration file is not required. The system will automatically resolve relationships during import, assuming the object names and IDs resolve correctly between the dependent objects.

Basic Structure of Identity Wrapper Objects

The configuration file uses identity wrapper objects to define any ConfigEntity or ManagementEntity object that can be exported or imported. Following is the basic structure of an identity wrapper object:

```
{
    "type" : "identitywrapper",
    "data" : {},
    "parentName" : "container-name",
    "oldName" : "old-object-name",
    "action" : "EDIT", //Enum values: CREATE, EDIT or DELETE
```

```
  "index" : integer,
}
```

The object contains the following attributes:

- **type**—This is always **identitywrapper**.
- **data**—This is the collection of attribute-value pairs that define the object from the configuration, such as a network object, access control rule, and so forth. The attributes needed in this collection depend on the model for the specific object type and the action you are taking. Enclose the attribute-value pairs in {braces}. Separate the attributes within the data array with commas.
- **parentName**—(If needed.) A limited number of objects are ContainedObjects, which have a relationship to an object that contains them. Examples include access rules, manual NAT rules, and subinterfaces. For these items, the parentName specifies the name of the containing object (the parent). Specify this attribute for contained objects. Do not specify it for non-contained objects. You might also need to specify index for these objects.

  You can actually omit this attribute if the parent is a single object (that is, you cannot create more than one), such as AccessPolicy, and the system can resolve the reference.
- **oldName**—(If needed.) If you are renaming an existing object, you can specify the old name on this attribute, and the new name in the **name** attribute of the data attributes. The action must be EDIT to use this attribute.
- **action**—The action to take with respect to the defined object. In full exports, the action is always **CREATE**. For pending change or partial exports, other actions might be **EDIT** or **DELETE**.

  When you edit the file for import, specify the desired action. Note that if you specify CREATE but the object already exists, the action is changed to EDIT; if the object does not exist, EDIT is changed to CREATE. The DELETE action is not changed. Object references are resolved based on object type and name, or object type and old name, or object type and parent name.
  - CREATE—This is a new object. You need to specify the data attributes that are required when posting an object. Note that if the **name** matches an existing object of the specified type, the action is automatically changed to EDIT.

    Note that if you create a new object and reference that object from other objects, such as defining a network object and then using it in an access rule, the object **name** must be correct in the reference.
  - EDIT—You are updating an object. You need to specify the data attributes that are required when putting an object, except for version and id. The name and object type are used to determine the object to update, and the version attribute is always ignored.
  - DELETE—You are deleting the object. You must specify the **type** and **name** attributes in the object data.
- **index**—(Optional; integer.) For objects that are part of an ordered list, such as access control and manual NAT rules, the position of the object in the policy. If you are creating a new rule and you do not specify an index value, the rule is added to the end of policy as the last rule. If you are editing the rule, the system will retain the rule's existing position.

Example: Editing a Network Object for Import Into a Different Device

Each object is structured like the following, which is a network host object that defines the IP address of the syslog server:

```
{"type":"identitywrapper",
 "action":"CREATE",
 "data":{
    "version":"lfxdbtbyg4ex6",
    "name":"syslog-host",
    "subType":"HOST",
    "value":"10.100.10.10",
    "isSystemDefined":false,
    "dnsResolution":"IPV4_AND_IPV6",
    "id":"2cd0ea03-62a7-11e9-8b8d-dbf377c781d8",
    "type":"networkobject"}}
```

Suppose you exported this object from a device, and you want to import the object into a different device, but the new device should use a syslog server at a different address, 192.168.5.15. Because you are going to create a new object, remove the **version** and **id** attributes from the data attribute. You can also remove **isSystemDefined** (whose default is false) and **dnsResolution** (which is relevant for an FQDN object only). The resulting new object would look like the following:

```
{"type":"identitywrapper",
 "action":"CREATE",
 "data":{
    "name":"syslog-host",
    "subType":"HOST",
    "value":"192.168.5.15",
    "type":"networkobject"}}
```

At the top of the file, you need to retain (or add) the metadata object. You can also add line returns to make it easier to scan and verify the file content. Thus, the complete configuration file would look like the following:

```
[
{"hardwareModel":"Cisco Firepower Threat Defense for VMWare",
 "type":"metadata",
 "configType":"DELTA_CONFIG",
 "apiVersion":"latest",
 "exportType":"PARTIAL_EXPORT",
 "softwareVersion":"6.5.0-10465"}
,
{"type":"identitywrapper",
 "action":"CREATE",
 "data":{
    "name":"syslog-host",
    "subType":"HOST",
    "value":"192.168.5.15",
    "type":"networkobject"}}
]
```

Upload the Import File

Before you can import a configuration file into a device, you must first upload the file to the device. You can upload either zip or text files. You can include AnyConnect packages and client profiles if you use a zip file.

Use the POST /action/uploadconfigfile resource to upload the file. The name has a maximum length of 60 characters.

- If you use this method from API Explorer, click the **Choose File** button next to the **fileToUpload** attribute to select the file from your workstation drive.

- If you are using the method from your own program, the request payload must contain a single file-item with a file-name field. The file-name extension must be either .txt or .zip and the actual file content format must be consistent with the file extension.

The curl command would look like the following:

```
curl -F 'fileToUpload=@./import-1.txt'
'https://10.89.5.38/api/fdm/latest/action/uploadconfigfile’
```

A successful transfer results in a 200 return code and a response body similar to the following, which shows the file name on the threat defense system (**diskFileName**), which you need for the import job.

```
{
  "diskFileName": "import-1.txt",
  "dateModified": "2019-04-22 10:18:12Z",
  "sizeBytes": 267,
  "id": "default",
  "type": "configimportexportfileinfo",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest/action/uploadconfigfile/default"
  }
}
```

Import the Configuration and Check Job Status

After you upload a configuration file to the threat defense system, you can import the objects defined in the configuration file into the threat defense configuration. Use the POST /action/configimport method.

When importing objects, you also have the option of defining the objects directly in the import command rather than in a configuration file. However, you should directly define objects only in cases where you are importing a small number of changes, such as one or two network objects.

Procedure

Step 1 Create the JSON object body for the import job.

Following is an example of the JSON object to use with this call.

```
{
  "diskFileName": "string",
  "encryptionKey": "*********",
  "preserveConfigFile": true,
  "autoDeploy": true,
  "allowPendingChange": true,
  "excludeEntities": [
     "string"
     ]",
  "inputEntities": [
    {
      "action": "CREATE",
      "oldName": "string",
      "parentId": "string",
      "parentName": "string",
      "index": 0,
      "data": {
          "version": "string",
          "id": "string",
          "type": "identity"
        },
        "id": "string",
        "type": "IdEntityWrapper"
      }
    ],
    "jobName": "string",
    "type": "scheduleconfigimport"
  }
```

The attributes are:

- **diskFileName**—The name of the configuration zip or txt file to be imported.
- **encryptionKey**—The key used to encrypt the zip file, if any. Do not specify a key if the configuration file is not encrypted.
- **preserveConfigFile**—(Optional.) Whether to keep the copy of the configuration file imported on the threat defense disk after a successful import job. Specify true to keep the file, false to have the file deleted from the threat defense disk. The default is false.
- **autoDeploy**—(Optional.) Whether to automatically start a deployment job if the import is successful. Imported objects are pending changes, and they are not active until you successfully deploy the changes. Specify true to start the deployment job automatically. If you specify false, you must manually deploy your changes. The default is false.
- **allowPendingChange**—(Optional.) Whether to allow the import job to start if there are existing pending changes. If you set this attribute to true, and autoDeploy to true, then the automatic deployment job includes all changes, both pre-existing and imported. If you set this attribute to false, then the import job will not run if there are pending changes. The default is false.
- **excludeEntities**—(Optional.) A list of object matching strings that identify objects that should not be imported. You need to specify this attribute only if the import file includes items that you do not want to import (that is, you decided to not delete them from the file you uploaded). Each item in this list has a pattern like "**id=***uuid-value*", "**type**=*object-type*" or "**name**=*object-name*". Input objects that match one of these patterns will be excluded from import.

  The **type** can be either a leaf entity, such as networkobject, or an alias of a set of leaf types. Some typical type aliases are: network (NetworkObject and NetworkObjectGroup), port (all TCP/UDP/ICMP port, protocol and group types), url (URL objects and groups), ikepolicy (IKE V1/V2 policies), ikeproposal (Ike V1/V2 proposals), identitysource (all identity sources), certificate (all certificate types), object (all object/group types that would be listed in the device manager on the Objects page), interface (all network interfaces, s2svpn (all site-to-site VPN related types), ravpn (all RA VPN related types), vpn (both s2svpn and ravpn).

  For example, to exclude all network objects, and two other objects identified by the name myobj and a UUID from being imported, specify:

  ```
  "excludeEntities": [
     "type=networkobject",
     "name=myobj",
     "id=acc2e3cd-8c70-11e9-930a-1f12ee87b286"
     ]",
  ```

- **inputEntities**—If you have a small number of objects to import, you can define them in the inputEntities object list rather than in a configuration file. To use this attribute, you cannot include the diskFileName attribute, or you must set that attribute to null.
- **jobName**—(Optional.) A name for the export job. Giving the job a name might make it easier to find it when you retrieve job status.
- **type**—The job type, which is always **scheduleconfigimport**.

Example:

The following example imports the configuration file named import-1.txt:

```
{
  "diskFileName": "import-2.txt",
  "preserveConfigFile": true,
  "autoDeploy": true,
  "allowPendingChange": true,
  "type": "scheduleconfigimport"
}
```

Step 2 Post the object.

For example, the curl command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
 -d '{ \
   "diskFileName": "import-2.txt", \
   "preserveConfigFile": true, \
   "autoDeploy": true, \
   "allowPendingChange": true, \
   "type": "scheduleconfigimport" \
 }' 'https://10.89.5.38/api/fdm/latest/action/configimport'
```

Step 3 Verify the response.

You should get a response code of 200. A successful response body would look something like the following if you posted the minimum JSON object. If you specify an encryption key, it is masked in the response.

```
{
  "version": null,
  "scheduleType": "IMMEDIATE",
  "user": "admin",
  "forceOperation": false,
  "jobHistoryUuid": "7e360139-6725-11e9-abb5-078014531401",
  "ipAddress": "10.24.127.37",
  "diskFileName": "import-2.txt",
  "encryptionKey": null,
  "preserveConfigFile": true,
  "autoDeploy": true,
  "allowPendingChange": true,
  "jobName": "Config Import",
  "id": "7e2b52d8-6725-11e9-abb5-5dec35337506",
  "type": "scheduleconfigimport",
  "links": {
    "self": "https://10.89.5.38/api/fdm/latest
/action/configimport/7e2b52d8-6725-11e9-abb5-5dec35337506"
  }
}
```

Step 4 Use GET /jobs/configimportstatus to check the status of the import job.

Alternatively, you can use GET /jobs/configimportstatus/{objId} to get status of one import job. For objId, use the jobHistoryUuid value from the response body to your POST /action/configimport call.

The curl command would look like the following:

```
curl -X GET --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/jobs/configimportstatus'
```

The response body might look like the following for a successful import. If the import fails, you might need to edit the file to correct formatting or content errors and try again.

```
{
      "version": "pcgccfnk4hmiz",
      "jobName": "Config Import",
      "jobDescription": null,
      "user": "admin",
      "startDateTime": "2019-04-25 06:43:54Z",
      "endDateTime": "2019-04-25 06:44:01Z",
      "status": "SUCCESS",
      "statusMessage": "The configuration was imported successfully",
      "scheduleUuid": "7e2b52d8-6725-11e9-abb5-5dec35337506",
      "diskFileName": "import-2.txt",
      "messages": [],
      "preserveConfigFile": true,
      "autoDeploy": true,
      "allowPendingChange": true,
      "id": "7e360139-6725-11e9-abb5-078014531401",
      "type": "configimportjobstatus",
      "links": {
        "self": "https://10.89.5.38/api/fdm/latest
/jobs/configimportstatus/7e360139-6725-11e9-abb5-078014531401"
  }
}
```

What to do next

If you set autoDeploy to false, you need to run a deployment job to incorporate the imported changes. Use the POST /operational/deploy method. If you set it to true, the configuration should have been deployed successfully. In the device manager or the API (GET /operational/auditevents), you can check the audit log, and the deployment job is named “Post Configuration Import Deployment.”

Note Some features require particular licenses. For example, a device must have a license for any remote access VPN features. However, the import process does not validate licenses. Thus, if you import objects for a license-controlled feature to a device that does not have the required license, the deployment job will fail. If you encounter this problem, either assign the required licenses to the device, or delete the objects.

Delete Unneeded Import/Export Files

If you no longer need a configuration file, either one created by an export job or one that you uploaded for configuration import, you can delete the file.

Use the DELETE /action/configfiles/{objId} method, using the file name as the objId value.

For example, to delete the file named export-config-2.zip, the curl command would be the following:

```
curl -X DELETE --header 'Accept: application/json'
'https://10.89.5.38/api/fdm/latest/action/configfiles/export-config-2.zip'
```

A successful result is a 204 return code with no response body.

You can use GET /action/configfiles to confirm that the file was deleted.