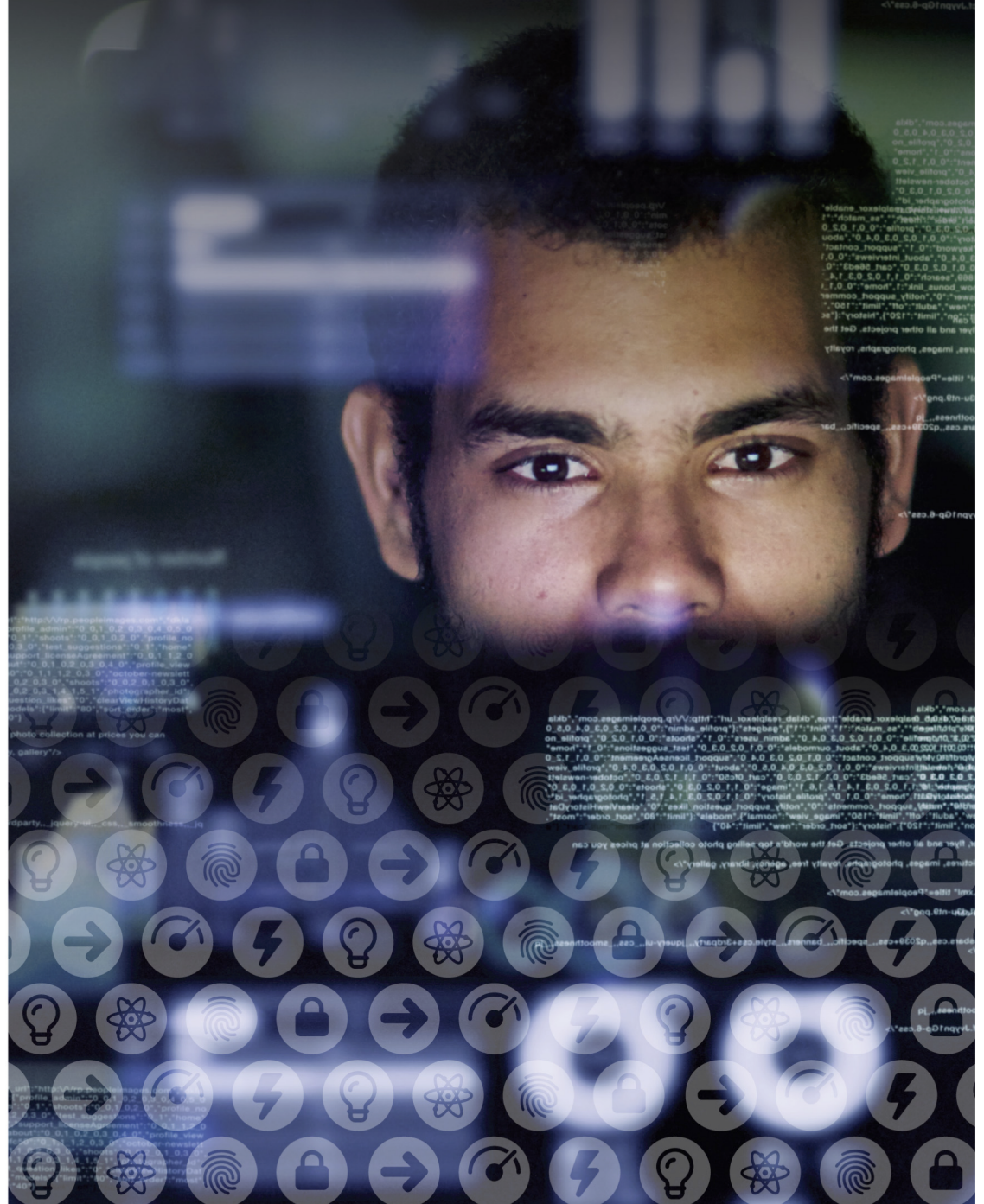




## Agility 2017 Hands-on Lab Guide

# F5 Programmability Training

<https://github.com/f5devcentral/f5-automation-labs/graphs/contributors>





## Contents:

<b>1</b>	<b>Welcome</b>	<b>5</b>
<b>2</b>	<b>Getting Started</b>	<b>7</b>
<b>3</b>	<b>Lab Topology</b>	<b>9</b>
<b>4</b>	<b>Class 1 - Introduction to Automation &amp; Orchestration</b>	<b>11</b>
4.1	Module 1 – REST API Basics & Device Onboarding . . . . .	11
4.2	Module 2 – iWorkflow . . . . .	35
4.3	Module 3 – f5-super-netops-container Toolkit . . . . .	47
4.4	Module 4 – f5-postman-workflows & f5-newman-wrapper . . . . .	55
4.5	Module 5 - Python SDK . . . . .	81
<b>5</b>	<b>HOWTOs: Index</b>	<b>91</b>
5.1	HOWTO - Update Existing iApp templates to Work with iWorkflow v2.1 . . . . .	91





## Welcome

Welcome to F5's Automation, Orchestration and Programmability Training series. The intended audience for these labs are Super NetOps and DevOps engineers that would like to leverage the various programmability tools offered by the F5 platform. If you require a pre-built lab environment please contact your F5 account team and they can provide access to environments on an as-needed basis.

The content contained here leverages a full DevOps CI/CD pipeline and is sourced from the following GitHub repository:

<https://github.com/f5devcentral/f5-automation-labs/>

Bugs and Requests for enhancements can be made using by opening an [Issue](#) within the repository.



## Getting Started

Please follow the instructions provided by the instructor to start your lab and access your jump host.

---

**Note:** All work for this lab will be performed exclusively from the Windows jumphost. No installation or interaction with your local system is required.

---



## Lab Topology

The network topology implemented for this lab is very simple. Since the focus of the lab is Control Plane programmability rather than Data Plane traffic flow we can keep the data plane fairly simple. The following components have been included in your lab environment:

- 2 x F5 BIG-IP VE (v12.1)
- 1 x F5 iWorkflow VE (v2.1)
- 1 x Linux LAMP Webserver (xubuntu 14.04)
- 1 x Linux Docker Server (CentOS 7)
- 1 x Windows Jumpbox

The following table lists VLANs, IP Addresses and Credentials for all components:

Component	VLAN/IP Address(es)	Credentials
Windows Jumphost	<ul style="list-style-type: none"> <li>• <b>Management:</b> 10.1.1.250</li> <li>• <b>Internal:</b> 10.1.10.250</li> <li>• <b>External:</b> 10.1.20.250</li> </ul>	Administrator/ <i>available in instance details</i>
BIG-IP A	<ul style="list-style-type: none"> <li>• <b>Management:</b> 10.1.1.4</li> <li>• <b>Internal:</b> 10.1.10.1</li> <li>• <b>Internal (Float):</b> 10.1.10.3</li> <li>• <b>External:</b> 10.1.20.1</li> </ul>	admin/admin
BIG-IP B	<ul style="list-style-type: none"> <li>• <b>Management:</b> 10.1.1.5</li> <li>• <b>Internal:</b> 10.1.10.2</li> <li>• <b>Internal (Float):</b> 10.1.10.3</li> <li>• <b>External:</b> 10.1.20.2</li> </ul>	admin/admin
iWorkflow	<ul style="list-style-type: none"> <li>• <b>Management:</b> 10.1.1.6</li> </ul>	admin/admin
Linux Server	<ul style="list-style-type: none"> <li>• <b>Management:</b> 10.1.1.7</li> <li>• <b>Internal:</b> 10.1.10.10-13</li> </ul>	root/default
Docker Server	<ul style="list-style-type: none"> <li>• <b>Management:</b> 10.1.1.8</li> </ul>	root/default

## Class 1 - Introduction to Automation & Orchestration

This introductory class covers the following topics:

- Imperative Automation using the F5 BIG-IP iControl REST API
- Declarative Automation using the F5 iWorkflow product
- F5 Automation Tools:
  - The f5-super-netops-container
  - Collections and the f5-postman-workflows extension to Postman
  - Automation Workflows using f5-newman-wrapper

Expected time to complete: **6 hours**

### 4.1 Module 1 – REST API Basics & Device Onboarding

In this module you will learn the basic concepts required to interact with the BIG-IP iControl REST API. Additionally, you will walk through a typical Device Onboarding workflow that results in a fully functional BIG-IP Active/Standby pair. It's important to note that this module will focus on showing an **Imperative** approach to automation.

---

**Note:** The Lab Deployment for this lab includes two BIG-IP devices. For most of the labs we will focus on configuring only the BIG-IP-A device (management IP and licensing have already been completed). BIG-IP-B already has some minimal configuration loaded. In a real-world environment it would be necessary to perform Device Onboarding functions on ALL BIG-IP devices. We are only performing them on a single device in this lab so we are able to cover all topics in the time allotted.

---

---

**Note:** It's beneficial to have GUI/SSH sessions open to BIG-IP and iWorkflow devices while going through this lab. Feel free to verify the actions taken in the lab against the GUI or SSH. You can also watch the following logs:

- BIG-IP:
  - /var/log/ltn
  - /var/log/restjavad.0.log



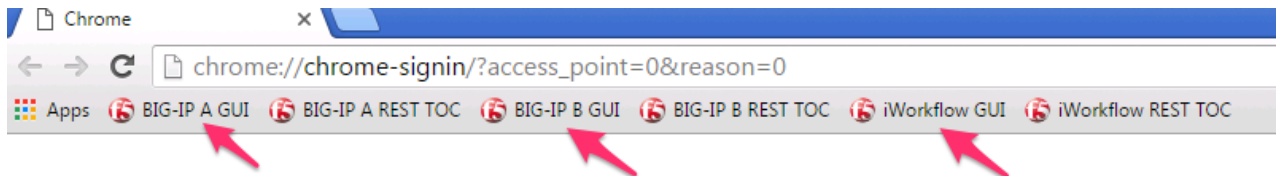
- iWorkflow:
  - /var/log/restjavad.0.log

## 4.1.1 Lab 1.1 – Exploring the iControl REST API

### Task 1 – Explore the API using the TMOS Web Interface

In this lab we will explore the API using an interface that is built-in to TMOS. This utility is useful for understanding how TMOS objects map to the REST API. The interfaces implement full Create, Read, Update and Delete (CRUD) functionality, however, in most practical use cases it's far easier to use this interface as a 'Read' tool rather than trying to Create objects directly from it. It's usually far easier to use TMUI or TMSH to create the object as needed and then use this tool to view the created object with all the correct attributes already populated.

1. Open Google Chrome and navigate to the following bookmarks: BIG-IP A GUI, BIG-IP B GUI and iWorkflow GUI. Bypass any SSL errors that appear and ensure you see the login screen for each bookmark.



2. Navigate to the URL <https://10.1.1.4/mgmt/toc> (or click the BIG-IP A REST TOC bookmark). The '/mgmt/toc' path in the URL is available on all TMOS versions 11.6 or newer.
3. Authenticate to the interface using the default admin/admin credentials.
4. You will now be presented with a top-level list of various REST resources. At the top of the page there is a search box  that can be used to find items on the page. Type 'net' in the search box

#### Table of Contents

#### iControl REST Resources

[net](#)

and then click on the 'net' link under iControl REST Resources: **Traffic Management**

5. Find the `/mgmt/tm/net/route-domain` **Collection** and click it.
6. You will now see a listing of the **Resources** that are part of the route-domain(s) collection. As you can see the default route domain of 0 is listed. You can also create new objects by clicking the **+** button. Additionally resources can be deleted using the button or edited using the button.
7. Click the 0 resource to view the attributes of route-domain 0 on the device:

## /mgmt/tm/net/route-domain/~Common~0

name	0
partition	Common
fullPath	/Common/0
connectionLimit	0
id	0
strict	enabled

Take note of the full path to the resource. Here is how the path is broken down:

```
/ mgmt / tm / net / route-domain / ~Common~0
| Root | OC | OC | Collection | Resource
*OC=Organizing Collection
```


### 4.1.2 Lab 1.2 – REST API Authentication & ‘example’ Templates

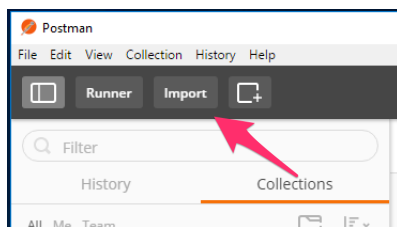
One of the many basic concepts related to interaction with REST API's is how a particular consumer is authenticated to the system. BIG-IP and iWorkflow support two types of authentication: HTTP BASIC and Token based. It's important to understand both of these authentication mechanisms, as consumers of the API will often make use of both types depending on the use case. This lab will demonstrate how to interact with both types of authentication.

#### Task 1 - Import the Postman Collection & Environment

In this task you will Import a Postman Collection & Environment for this lab. Perform the following steps to complete this task:

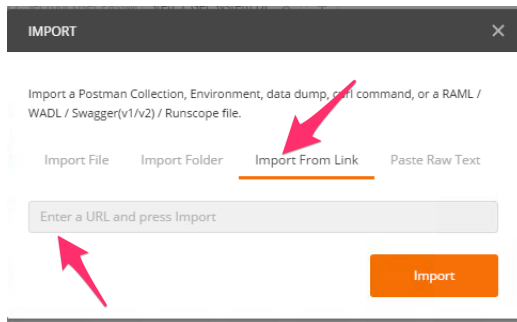


1. Open the Postman tool by clicking the  icon of the taskbar of your Windows Jumphost
2. Click the 'Import' button in the top left of the Postman window

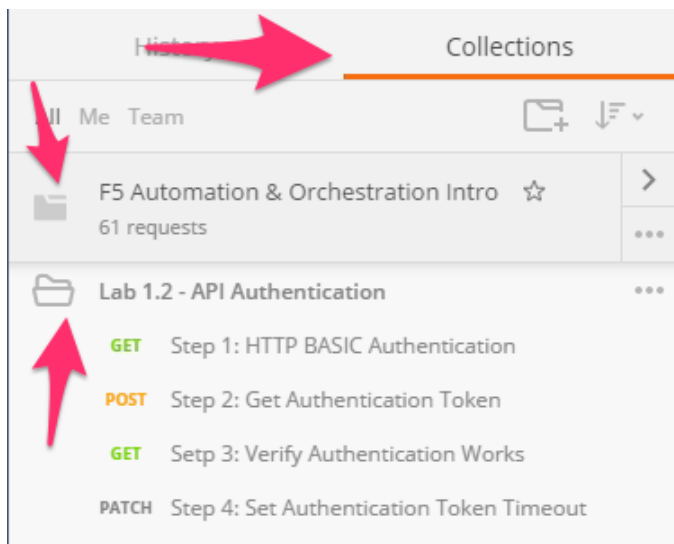


3. Click the 'Import from Link' tab. Paste the following URL into the text box and click 'Import'

```
https://raw.githubusercontent.com/f5devcentral/f5-automation-labs/
ondemand/postman_collections/F5_Automation_Orchestration_Intro.postman_
collection.json
```



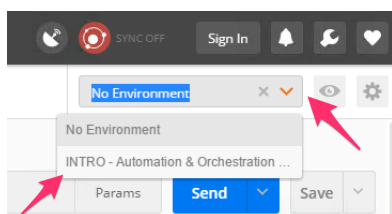
4. You should now see a collection named 'F5 Automation & Orchestration Intro' in your Postman Collections sidebar:



5. Import the Environment file by clicking 'Import' -> 'Import from Link' and pasting the following URL and clicking 'Import':

```
https://raw.githubusercontent.com/f5devcentral/f5-automation-labs/ondemand/postman_collections/INTRO_Automation_Orchestration_Lab.postman_environment.json
```

6. To assist in multi-step procedures we make heavy use of the 'Environments' capability in Postman. This capability allows us to set various global variables that are then substituted into a request before it's sent. Set your environment to 'INTRO - Automation & Orchestration Lab' by using the menu at the top right of your Postman window:



## Task 2 – HTTP BASIC Authentication

In this task we will use the Postman tool to send API requests using HTTP BASIC authentication. As its name implies this method of authentication encodes the user credentials via the existing BASIC authentication.

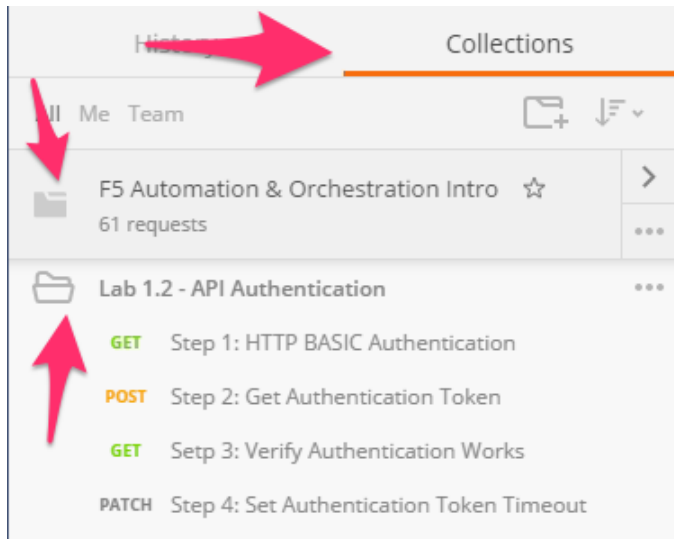
tion method provided by the HTTP protocol. The mechanism this method uses is to insert an HTTP header named 'Authorization' with a value that is built by Base 64 encoding the string <username> : <password>. The resulting header takes this form:

```
Authorization: Basic YWRtaW46YWRtaW4=
```

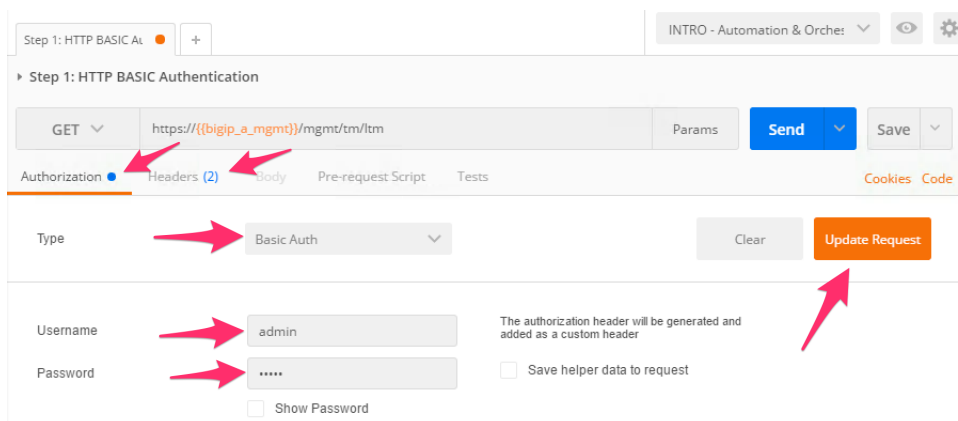
It should be noted that cracking the method of authentication is TRIVIAL; as a result API calls should always be performed using HTTPS (F5 default) rather than HTTP.

Perform the following steps to complete this task:

1. Click the 'Collections' tab on the left side of the screen, expand the 'F5 Automation & Orchestration Intro' collection on the left side of the screen, expand the 'Lab 1.2 – API Authentication' folder:



2. Click the 'Step 1: HTTP BASIC Authentication' item. Click the 'Authorization' tab and select 'Basic Auth' as the Type. Fill in the username and password (admin/admin) and click the 'Update Request' button. Notice that the number of Headers in the Headers tab changed from 1 to 2. This is because Postman automatically created the HTTP header and updated your request to include it. Click the 'Headers' tab and examine the HTTP header:



3. Click the 'Send' button to send the request. If the request succeeds you should be presented with a listing of the /mgmt/tm/ltn Organizing Collection.
4. Update the credentials and specify an INCORRECT password. Send the request again and examine the response:



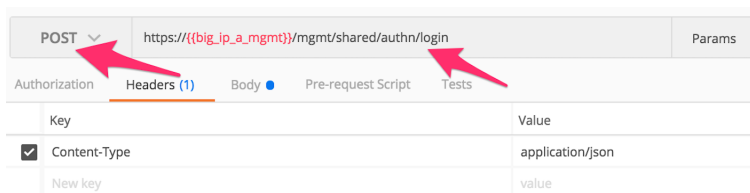
### Task 3 – Token Based Authentication

One of the disadvantages of BASIC Authentication is that credentials are sent with each and every request. This can result in a much greater attack surface being exposed unnecessarily. As a result Token Based Authentication (TBA) is preferred in many cases. This method only sends the credentials once, on the first request. The system then responds with a unique token for that session and the consumer then uses that token for all subsequent requests. Both BIG-IP and iWorkflow support token-based authentication that drops down to the underlying authentication subsystems available in TMOS. As a result the system can be configured to support external authentication providers (RADIUS, TACACS, AD, etc) and those authentication methods can flow through to the REST API. In this task we will demonstrate TBA using the local authentication database, however, authentication to external providers is fully supported.

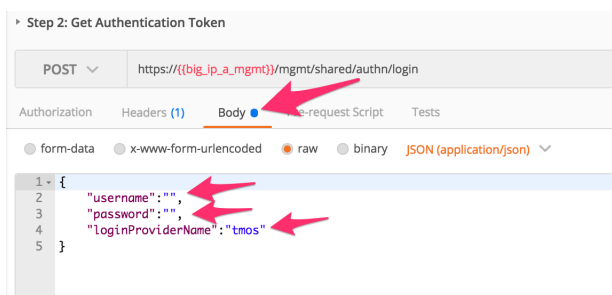
For more information about external authentication providers see the section titled “**About external authentication providers with iControl REST**” in the iControl REST API User Guide available at <https://devcentral.f5.com>

Perform the following steps to complete this task:

1. Click the 'Step 2: Get Authentication Token' item in the Lab 1.2 Postman Collection
2. Notice that we send a POST request to the `/mgmt/shared/authn/login` endpoint.



3. Click the 'Body' tab and examine the JSON that we will send to BIG-IP to provide credentials and the authentication provider:



4. Modify the JSON body and add the required credentials (admin/admin). Then click the 'Send' button.
5. Examine the response status code. If authentication succeeded and a token was generated the response will have a 200 OK status code. If the status code is 401 then check your credentials:

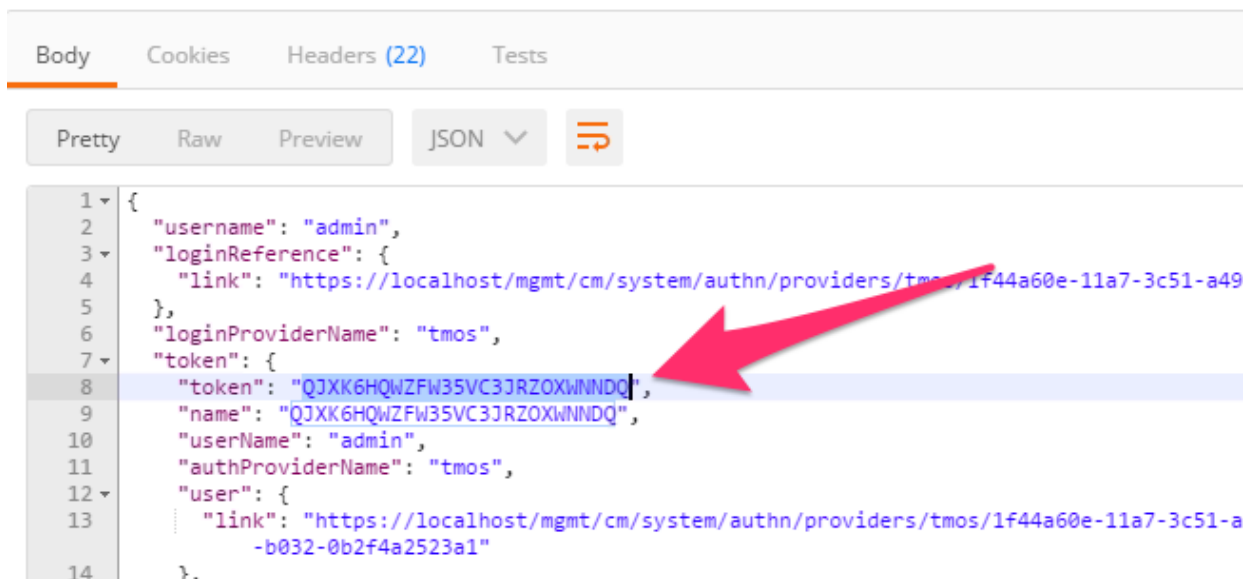
**Successful:**



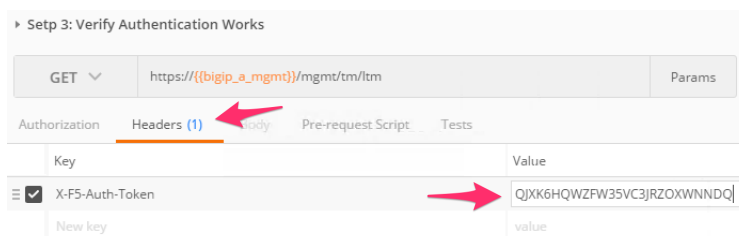
**Unsuccessful:**



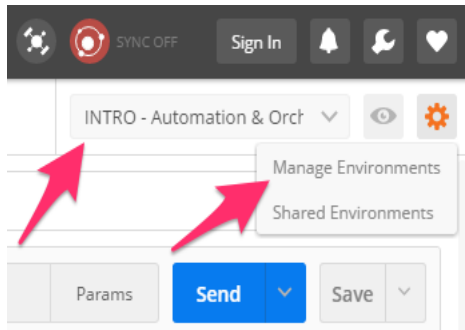
- Once you receive a 200 OK status code examine the response body. The various attributes show the parameters assigned to the particular token. Find the 'token' attribute and copy it into your clipboard (Ctrl+c) for use in the next step:



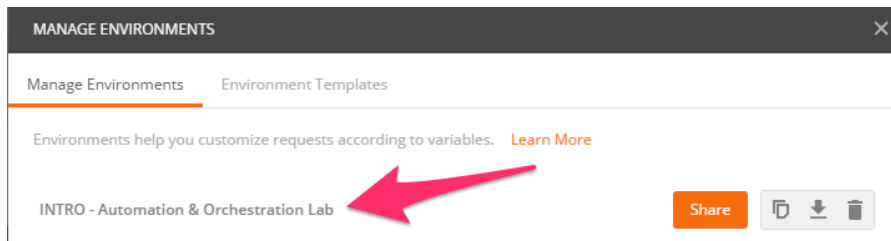
- Click the 'Step 3: Verify Authentication Works' item in the Lab 1.2 Postman collection. Click the 'Headers' tab and paste the token value copied above as the VALUE for the X-F5-Auth-Token header. This header is required to be sent on all requests when using token based authentication.



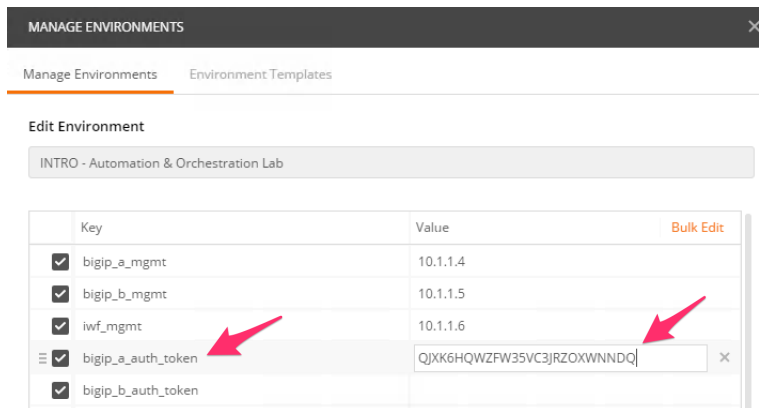
- Click the 'Send' button. If your request is successful you should see a '200 OK' status and a listing of the ltm Organizing Collection.
- We will now update your Postman environment to use this auth token for the remainder of the lab. Click the Environment menu in the top right of the Postman window and click 'Manage Environments':



10. Click the 'INTRO – Automation & Orchestration Lab' item:

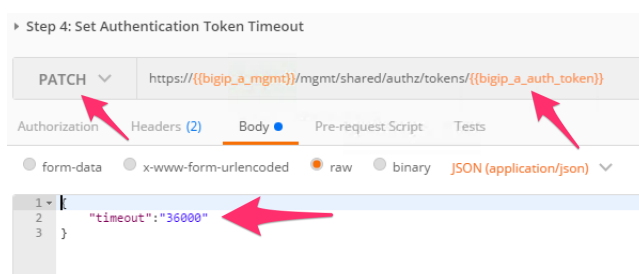


11. Update the value for `bigip_a_auth_token` by Pasting (Ctrl-v) in your auth token:



12. Click the 'Update' button and then close the 'Manage Environments' window. You're subsequent requests will now automatically include the token.

13. Click the 'Step 4: Set Authentication Token Timeout' item in the Lab 1.2 Postman collection. This request will PATCH your token Resource (check the URI) and update the timeout attribute so we can complete the lab easily. Examine the request type and JSON Body and then click the 'Send' button. Verify that the timeout has been changed to '36000' in the response:



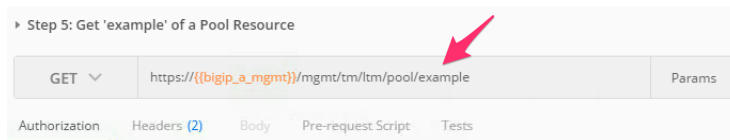


## Task 4 – Get a pool ‘example’ Template

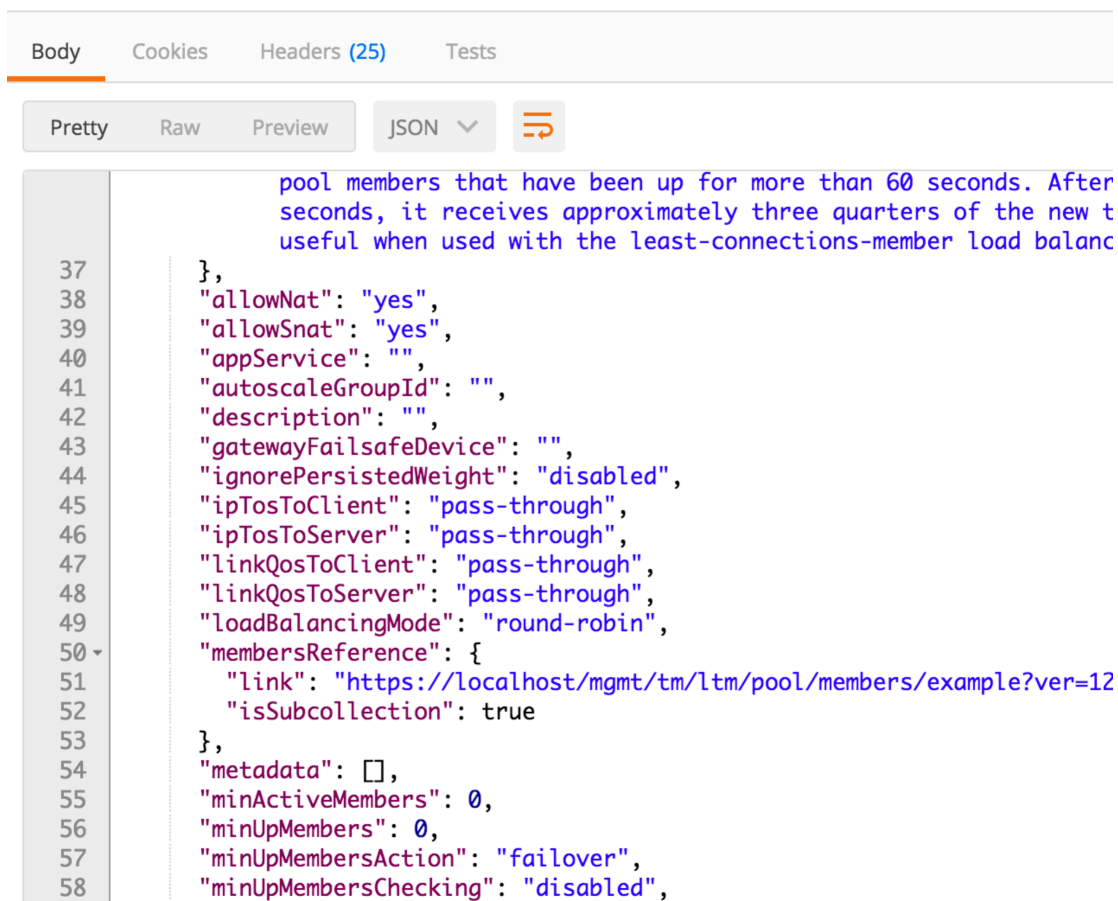
In order to assist with REST API interactions you can request a template of the various attributes of a Resource type in a Collection. This template can then be used as the body of a POST, PUT or PATCH request as needed.

Perform the following steps:

1. Click the ‘Step 5: Get ‘example’ of a Pool Resource’ item in the Lab 1.2 Postman collection
2. Examine the URI. Notice the addition of example at the end of the collection name:



3. Click ‘Send’ and examine the FULL response. You will see descriptions and then all the attributes for the Pool resource type. The response also shows the default values for the attributes if applicable:



### 4.1.3 Lab 1.3 – Review/Set Device Settings

Your BIG-IP-A device is already licensed, so now we can focus on configuring the basic infrastructure related settings to complete the Device Onboarding process. The remaining items include (list not exhaustive):

- Device Settings

- **NTP/DNS Settings**
- Remote Authentication
- **Hostname**
- **Admin Credentials**
- L1-3 Networking
  - Physical Interface Settings
  - L2 Connectivity (**VLAN**, VXLAN, etc.)
  - L3 Connectivity (**Self IPs**, **Routing**, etc.)
- HA Settings
  - **Global Settings**
    - \* **Config Sync IP**
    - \* **Mirroring IP**
    - \* **Failover Addresses**
  - **CMI Device Trusts**
  - **Device Groups**
  - **Traffic Groups**
  - **Floating Self IPs**

We will specifically cover the items in **BOLD** above in the following labs. It should be noted that many permutations of the Device Onboarding process exist due to the nature of customer environments. This class is designed to teach enough information so that you can then apply the knowledge learned and help articulate and/or deliver a specific solution for your environment.

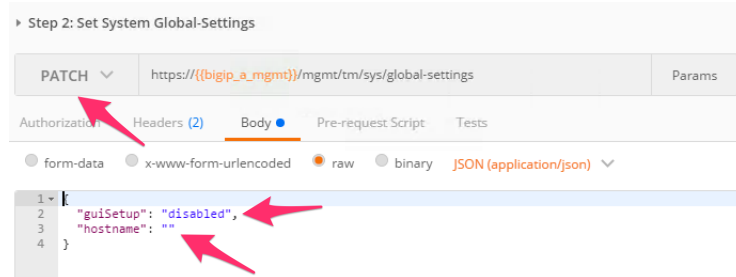
### Task 1 – Set Device Hostname & Disable GUI Setup Wizard

In this task we will modify the device hostname and disable the GUI Setup Wizard. The Resource that contains these settings is `/mgmt/tm/sys/global-settings`.

Perform the following steps to complete this task:

1. Expand the “Lab 1.3 – Review/Set Device Settings” folder in the Postman collection
2. Click the “Step 1: Get System Global-Settings” item. Click the ‘Send’ button and review the response body to see what the current settings on the device are.
3. Click the “Step 2: Set System Global-Settings” item. This item uses a PATCH request to the `global-settings` resource to modify the attributes contained within it. We will update the `guiSetup` and `hostname` attribute.
  - Review the JSON body and modify the ‘hostname’ attribute to set the hostname to `bigip-a.f5.local`

- Also notice that we are disabling the GUI Setup Wizard as part of the same request:



4. Click the 'Send' button and review the response body. You should see that the attributes modified above have been updated. You can also GET the global-settings again to verify they have been updated.

## Task 2 – Modify DNS/NTP Settings

**Note:** This task will make use of JSON arrays. The syntax for defining a JSON array is:

```
myArray: [ Object0, Object1 ... ObjectX ]
```

To define an array consisting of Strings the syntax is:

```
myStringArray: [ "string0", "string1" ... "stringX" ]
```

Much like the previous task we can update system DNS and NTP settings by sending a PATCH request to the correct resource in the 'sys' Organizing Collection. The relevant Resources for this task are:

URL	Type
/mgmt/tm/sys/dns	DNS Settings
/mgmt/tm/sys/ntp	NTP Settings

Perform the following steps to complete this task:

1. Click the "Step 3: Get System DNS Settings" item in the collection. Click 'Send' and review the current settings
2. Click the "Step 4: Set System DNS Settings" item in the collection. Review the JSON body to verify the name server IPs 4.2.2.2 and 8.8.8.8 are listed. Additionally add a search domain of 'f5.local'. You will modify a JSON array for both of these attributes.
3. Click the 'Send' button and verify the requested changes were successfully implemented
4. Click the "Step 5: Get System NTP Settings" item in the collection. Click 'Send' and review the current settings
5. Click the "Step 6: Set System NTP Settings" item in the collection. Review the JSON body to verify the NTP servers with hostnames 0.pool.ntp.org and 1.pool.ntp.org are contained in the servers attribute (another JSON array!)
6. Click the 'Send' button and verify the requested changes were successfully implemented

## Task 3 – Update default user account passwords

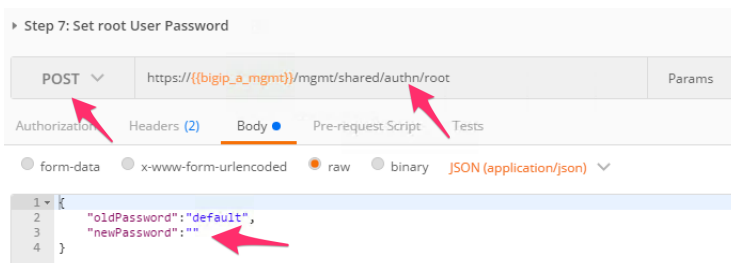
In this task we will update the passwords for the 'root' and 'admin' accounts. The process for updating the root account is different than other system accounts due to the special nature of the root account.

To update the root account password we will use a POST to a shared REST worker at `/mgmt/shared/authn/root`

To update all other system accounts we will PATCH the `/mgmt/auth/user/<username>` Resource

Perform the following steps to change the **root** user password:

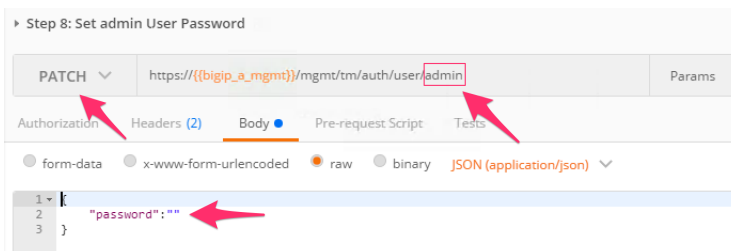
1. Click the “Step 7: Set root User Password” item in the collection.
2. Notice that we are performing a POST operation to a shared REST worker. Modify the JSON body to update the password to the value “newdefault” and click the ‘Send’ button.



3. You can verify the password was changed by opening an SSH session using PuTTY to BIG-IP-A.
4. **Repeat the procedure above to change the password back to “default”**

Perform the following steps to change the **admin** user password:

1. Click the “Step 8: Set admin User Password” item in the collection.
2. Notice that we are performing a PATCH operation to admin user Resource. Modify the JSON body to update the password to the value “newadmin” and click the ‘Send’ button.



3. You can verify the password was changed by opening an SSH session using PuTTY to BIG-IP-A OR by logging into TMUI in a Chrome browser tab.
4. **Repeat the procedure above to change the password back to “admin”**

#### 4.1.4 Lab 1.4 – Basic Network Connectivity

This lab will focus on configuration of the following items:

- L1-3 Networking
  - Physical Interface Settings
  - L2 Connectivity (**VLAN**, VXLAN, etc.)
  - L3 Connectivity (**Self IPs**, **Routing**, etc.)

We will specifically cover the items in **BOLD** above in the following labs. It should be noted that many permutations of the Device Onboarding process exist due to the nature of customer environments. This

class is designed to teach enough information so that you can then apply the knowledge learned and help articulate and/or deliver a specific solution to your customer.

The following table lists the L2-3 network information used to configure connectivity for BIG-IP-A:

Type	Name	Details
VLAN	Internal	Interface: 1.1 Tag: 10
VLAN	External	Interface: 1.2 Tag: 20
Self IP	Self-Internal	Address: 10.1.10.1/24 VLAN: Internal
Self IP	Self-External	Address: 10.1.20.1/24 VLAN: External
Route	Default	Network: 0.0.0.0/0 GW: 10.1.20.254

### Task 1 – Create VLANs

---

**Note:** This lab shows how to configure VLAN tags, but does not deploy tagged interfaces. To use tagged interfaces the `tagged` attribute needs to have the value `true`

---

Perform the following steps to configure the VLAN objects/resources:

1. Expand the “Lab 1.4 – Basic Network Connectivity” folder in the Postman collection.
2. Click the “Step 1: Create a VLAN” item in the collection. Examine the JSON body; the values for creating the Internal VLAN have already been populated.
3. Click the ‘Send’ button to create the VLAN
4. Repeat Step 1, however, this time modify the JSON body to create the External VLAN using the parameters in the table above.
5. Click the “Step 2: Get VLANs” item in the collection. Click the ‘Send’ button to GET the VLAN collection. Examine the response to make sure both VLANs have been created.

### Task 2 – Create Self IPs

Perform the following steps to configure the Self IP objects/resources:

1. Click the “Step 3: Create a Self IP” item in the collection. Examine the JSON body; the values for creating the Self-Internal Self IP have already been populated.
2. Click the ‘Send’ button to create the Self IP
3. Repeat Step 1, however, this time modify the JSON body to create the Self-External Self IP using the parameters in the table above.
4. Click the “Step 4: Get Self IPs” item in the collection. Click the ‘Send’ button to GET the Self IP collection. Examine the response to make sure both Self IPs have been created.

### Task 3 – Create Routes

Perform the following steps to configure the Route object/resource:

1. Click the “Step 5: Create a Route” item in the collection. Examine the JSON body; the values for creating the Default Route have already been populated.
2. Click the ‘Send’ button to create the Route
3. Click the “Step 6: Get Routes” item in the collection. Click the ‘Send’ button to GET the routes collection. Examine the response to make sure the route has been created.

### 4.1.5 Lab 1.5 – Build a BIG-IP Cluster

In this lab we will build a active-standby cluster between BIG-IP-A and BIG-IP-B. As mentioned previously, to save time, BIG-IP-B already has already been licensed and had its device level settings configured. This lab will walk you through creating the cluster step by step. As you will see complex operation such as this start to become less effective using the **Imperative** model of automation. Clustering is one of the ‘transition’ points for most customers to move into the **Declarative** model (if not already done) due to the need to abstract device/vendor level specifics from Automation consumers.

The high-level procedure required to create the cluster is:

1. Rename the CMI ‘Self’ Device name to match the hostname of the Device
2. Set BIGIP-A & BIGIP-B CMI Parameters (Config Sync IP, Failover IPs, Mirroring IP)
3. Add BIG-IP-B as a trusted peer on BIGIP-A
4. Check the device\_trust\_group Sync Group Status
5. Create a sync-failover Device Group
6. Check the status of the created Device Group
7. Perform initial sync of the Device Group
8. Check status (again)
9. Change the Traffic Group to use HA Order failover (not required but shown as an example)
10. Create Floating Self IPs

### Task 1 – Rename objects and Setup CMI Global Parameters

In this task we will complete Items 1&2 from the list high-level procedure at the beginning of the lab. One of the idiosyncrasies of BIG-IP is that when you use the GUI Setup Wizard to set the hostname of the device, the wizard automatically renames the CMI ‘Self’ device to match the hostname. Since we configured the hostname via a REST call earlier this action did not take place.

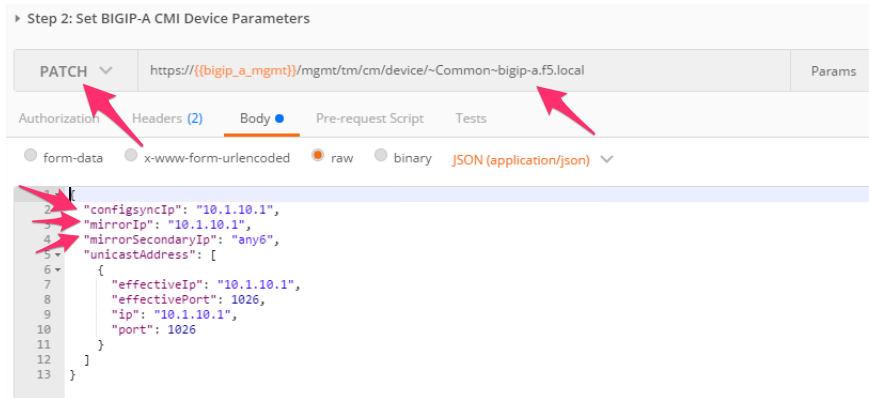
Perform the following steps to rename the CMI ‘Self’ device:

1. Expand the “Lab 1.5 – Build a Cluster” folder in the Postman collection
2. Click the “Step 1: Rename the CMI Self Device” item in the collection
3. Examine the URI and JSON body. We are sending a POST request to execute the equivalent of a `tmsh mv` command to rename the existing object to the `/mgmt/tm/cm/device` Collection. The `name` attribute specifies the current name of the object (the factory default name), while the `target` attribute specifies the new name of the object.
4. Click the ‘Send’ button to rename the Resource.

5. Change the request type from a POST to a GET and click 'Send'. Examine the response to make sure the name was changed successfully.

Perform the following steps to set CMI Device Parameters

1. Click the "Step 2: Set BIGIP-A CMI Device Parameters" item in the collection. Examine the operation (PATCH), URI and JSON body. We will PATCH the newly renamed object (from the previous step) and assign the Config Sync IP, Unicast Failover Address/Port and Mirroring IPs:



2. Click the 'Send' button and examine the response to ensure the settings were changed
3. Click the "Step 3: Set BIGIP-B CMI Device Parameters" item in the collection. Examine the operation (PATCH), URI and JSON body. We will PATCH and assign the Config Sync IP, Unicast Failover Address/Port and Mirroring IPs.

*EXTRA CREDIT: How is authentication to BIG-IP-B working if we never got an authentication token?  
(Hint: we cheated)*

4. Click the 'Send' button and examine the response to ensure the settings were changed

## Task 2 – Add BIG-IP-B as a Trusted Peer

The CMI subsystem relies on a PKI based device trust model to establish relationships between BIG-IP systems. In this task we will add BIG-IP-B as a trusted peer of BIG-IP-A. Establishing a trust relationship is automatically a bi-directional operation. As a result, when we establish the trust relationship, BIG-IP-B will automatically establish a trust relationship with BIG-IP-A. This task corresponds to items 3&4 in the high-level procedure.

Perform the following steps to complete this task:

1. Click the "Step 4: Add BIGIP-B Device to CMI Trust on BIGIP-A" item in the collection
2. Examine the operation (POST), URI and JSON body. We are using a special REST worker to add the device to the CMI trust. Additionally the JSON body must be specified in a very specific manner to ensure this step completes successfully. To minimize the chance for error the values have been completed for you already. You should, however, review and understand this step fully before continuing.
3. Click the 'Send' button. The response for this request does NOT indicate success, only that the command is running.
4. To check for success we have to check the status of the Sync Group named "device\_trust\_group". To do this click the "Step 5: Check Sync Group Status" item in the collection. This request will GET the sync status for all sync groups on the system



- Click the 'Send' button and examine the response. The should indicate a color of 'green', that bigip-b.f5.local is connected and 'In Sync' (please notify an instructor of any issue):

```
1 {
2   "kind": "tm:cm:sync-status:sync-statusstats",
3   "selfLink": "https://localhost/mgmt/tm/cm/sync-status?ver=12.1.2",
4   "entries": {
5     "https://localhost/mgmt/tm/cm/sync-status/0": {
6       "nestedStats": {
7         "entries": {
8           "color": {
9             "description": "green"
10          },
11          "https://localhost/mgmt/tm/cm/syncStatus/0/details": {
12            "nestedStats": {
13              "entries": {
14                "https://localhost/mgmt/tm/cm/syncStatus/0/details/0": {
15                  "nestedStats": {
16                    "entries": {
17                      "details": {
18                        "description": "bigip-b.f5.local: connected"
19                      }
20                    }
21                  },
22                  "https://localhost/mgmt/tm/cm/syncStatus/0/details/1": {
23                    "nestedStats": {
24                      "entries": {
25                        "details": {
26                          "description": "device_trust_group (In Sync): All devices in the device group are in sync"
27                        }
28                      }
29                    }
30                  },
31                  "https://localhost/mgmt/tm/cm/syncStatus/0/details/2": {
32                    "nestedStats": {
33                      "entries": {
34                        "details": {
35                          "description": "Optional action: Add a device group"
36                        }
37                      }
38                    }
39                  }
40                }
41              }
42            }
43          }
44        }
45      }
46    }
47  }
48 }
```

### Task 3 – Create a sync-failover Device Group

This task will create a Device Group object that will contain the two BIG-IP systems. The type of device-group will be a 'sync-failover' group, however, 'sync-only' groups can also be created with the same procedure but different attribute values. This task corresponds to items 5-8 in the high-level procedure.

Perform the following steps to complete this task

- Click the "Step 6: Create Device Group" item in the collection. Examine the request type, URL and JSON body. We will POST to the '/mgmt/tm/cm/device-group' collection and create a new Resource called DeviceGroup1 that includes both BIG-IP devices and is set to 'sync-failover' type. We are also setting the device-group to 'autosync' so manual syncing is not required when configuration changes occur:

```
1 POST https://localhost/mgmt/tm/cm/device-group
2 {
3   "name": "DeviceGroup1",
4   "type": "sync-failover",
5   "autoSync": "enabled",
6   "devices": [ "bigip-a.f5.local", "bigip-b.f5.local" ]
7 }
```

- Click the 'Send' button and examine the response.
- To check the status of the device-group we have to check the status of the underlying sync group on the system. Click the 'Step 7: Check Sync Group Status' item in the collection and click 'Send'. Examine the response and take note that the system is 'Awaiting Initial Sync':

```

1 {
2   "kind": "tm:cm:sync-status:sync-statusstats",
3   "selfLink": "https://localhost/mgmt/tm/cm/sync-status?ver=12.1.2",
4   "entries": {
5     "https://localhost/mgmt/tm/cm/sync-status/0": {
6       "nestedStats": {
7         "entries": {
8           "color": {
9             "description": "blue"
10          },
11          "https://localhost/mgmt/tm/cm/syncStatus/0/details": {
12            "nestedStats": {
13              "entries": {
14                "https://localhost/mgmt/tm/cm/syncStatus/0/details/0": {
15                  "nestedStats": {
16                    "entries": {
17                      "details": {
18                        "description": "bigip-b.f5.local: connected"
19                      }
20                    }
21                  },
22                  "https://localhost/mgmt/tm/cm/syncStatus/0/details/1": {
23                    "nestedStats": {
24                      "entries": {
25                        "details": {
26                          "description": "DeviceGroup1 (Awaiting Initial Sync): The device group is awaiting the initial config sync"
27                        }
28                      }
29                    },
30                    "https://localhost/mgmt/tm/cm/syncStatus/0/details/2": {
31                      "nestedStats": {
32                        "entries": {
33                          "details": {
34                            "description": "- Recommended action: Synchronize one of the devices to the group"
35                          }
36                        }
37                      }
38                    }
39                  }
40                }
41              }
42            }
43          }
44        }
45      }
46    }
47  }
48 }

```

- We will now manually sync DeviceGroup1 to fulfill the need for the Initial Sync. Click the 'Step 8: Manually Sync DeviceGroup1' item in the collection. Examine the request type, URL and JSON body. We will POST the the '/mgmt/tm/cm/config-sync' worker and tell it to 'run' a config-sync of BIG-IP-A 'to-group' DeviceGroup1:

```

1 {
2   "command": "run",
3   "options": [{"to-group": "DeviceGroup1"}]
4 }

```

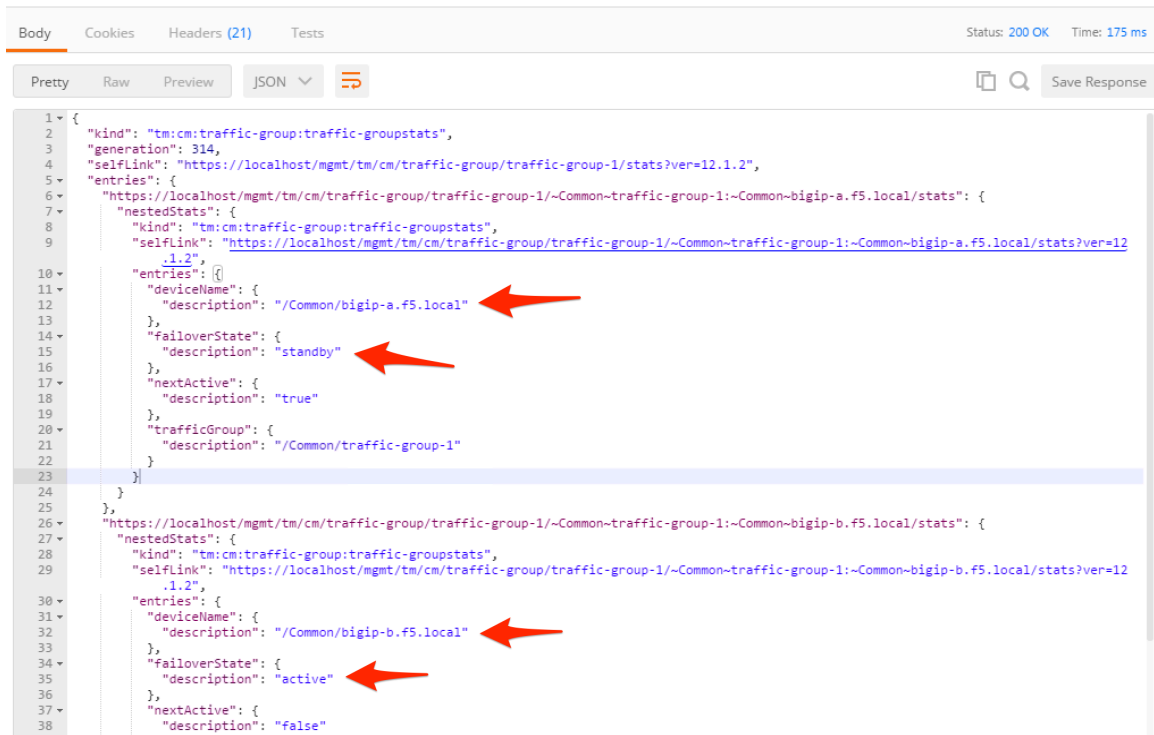
- Click 'Send' to initiate the sync
- Click the 'Step 9: Check Sync Group Status' item in the collection and click the 'Send' button. Examine the response to make sure that DeviceGroup1 is 'In Sync'. You may have to click 'Send' multiple times as the sync operation can take a while to complete.

## Task 4 – Perform Additional Operations

The remainder of the steps show how to manipulate various common items related to the HA config. In this task we will change the Traffic Group to use the 'HA Order' failover method. We will then initiate a failover and show how to view the status of the traffic-group.

Perform the following steps to complete this task:

1. Click the “Step 10: Get Traffic Group Properties” item in the collection. Examine the URL, we will GET the attributes of the ‘traffic-group-1’ resource from the traffic-group collection. Click the ‘Send’ button and review the response.
2. Click the “Step 11: Change Traffic Group to use HA Order” item in the collection. Examine the request type, URL and JSON body. We will PATCH the existing resource and specify an ‘haOrder’ attribute to change the traffic-group behavior.
3. Click the ‘Send’ button and examine the response to verify the change was successful.
4. Click the “Step 12: Get Traffic Group Failover States” item in the collection and click the ‘Send’ button. Examine the response and determine which device is ‘active’ for the traffic-group:



```
1 {
2   "kind": "tm:cm:traffic-group:traffic-groupstats",
3   "generation": 314,
4   "selfLink": "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/stats?ver=12.1.2",
5   "entries": {
6     "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1::~Common-bigip-a.f5.local/stats": {
7       "nestedStats": {
8         "kind": "tm:cm:traffic-group:traffic-groupstats",
9         "selfLink": "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1::~Common-bigip-a.f5.local/stats?ver=12.1.2",
10        "entries": {
11          "deviceName": {
12            "description": "/Common/bigip-a.f5.local"
13          },
14          "failoverState": {
15            "description": "standby"
16          },
17          "nextActive": {
18            "description": "true"
19          },
20          "trafficGroup": {
21            "description": "/Common/traffic-group-1"
22          }
23        }
24      },
25      "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1::~Common-bigip-b.f5.local/stats": {
26        "nestedStats": {
27          "kind": "tm:cm:traffic-group:traffic-groupstats",
28          "selfLink": "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1::~Common-bigip-b.f5.local/stats?ver=12.1.2",
29          "entries": {
30            "deviceName": {
31              "description": "/Common/bigip-b.f5.local"
32            },
33            "failoverState": {
34              "description": "active"
35            },
36            "nextActive": {
37              "description": "false"
38            }
39          }
40        }
41      }
42    }
43  }
```

5. Click EITHER the “Step 13A” or “Step 13B” item in the collection depending on which device is ACTIVE for the traffic group. Notice that we are sending the request to the ACTIVE device for the traffic group. Examine the JSON body and click the ‘Send’ button.
6. Click the “Step 14: Get Traffic Group Failover States” item in the collection and click the ‘Send’ button. Examine the response to determine that the failover occurred properly:

```
1 {
2   "kind": "tm:cm:traffic-group:traffic-groupstats",
3   "generation": 331,
4   "selfLink": "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/stats?ver=12.1.2",
5   "entries": {
6     "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1:~Common-bigip-a.f5.local/stats": {
7       "nestedStats": {
8         "kind": "tm:cm:traffic-group:traffic-groupstats",
9         "selfLink": "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1:~Common-bigip-a.f5.local/stats?ver=12.1.2",
10        "entries": {
11          "deviceName": {
12            "description": "/Common/bigip-a.f5.local"
13          },
14          "failoverState": {
15            "description": "active"
16          },
17          "nextActive": {
18            "description": "false"
19          },
20          "trafficGroup": {
21            "description": "/Common/traffic-group-1"
22          }
23        }
24      },
25    },
26    "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1:~Common-bigip-b.f5.local/stats": {
27      "nestedStats": {
28        "kind": "tm:cm:traffic-group:traffic-groupstats",
29        "selfLink": "https://localhost/mgmt/tm/cm/traffic-group/traffic-group-1/~Common-traffic-group-1:~Common-bigip-b.f5.local/stats?ver=12.1.2",
30        "entries": {
31          "deviceName": {
32            "description": "/Common/bigip-b.f5.local"
33          },
34          "failoverState": {
35            "description": "standby"
36          },
37          "nextActive": {
38            "description": "true"
39          }
40        }
41      },
42    }
43  }
44 }
```

## Task 5 – Create Floating Self IPs

To complete the HA config we will now create a Floating Self IP on the Internal VLAN.

Perform the following steps to complete this task:

1. Click the “Step 15: Create a Floating Self IP” item in the collection. Examine the request type, URL and JSON body. We will create a new resource in the `/mgmt/tm/net/self` collection named ‘Self-Internal-Floating’ and an IP address of 10.1.10.3.
2. Click the ‘Send’ button and examine the response
3. Click the “Step 16: Get Self IPs” item in the collection and click ‘Send’. Examine the response and verify the Self IP was created.

### 4.1.6 Lab 1.6 – Build a Basic LTM Config

In this lab we will build a basic LTM Config using the Imperative automation model. While this lab may seem simple for basic configurations, the complexity involved with rich L4-7 services quickly makes the Imperative approach untenable for advanced configurations. The Imperative model relies on the user having in-depth knowledge of device specifics such as:

- Object types and their attributes
  - How many different objects/profiles/options do we have?
- Order of operations
  - Monitor before pool before profiles before virtual servers, etc.
  - What about L7 use cases like WAF?

- \* WAF Policy -> HTTP Policy -> Virtual Server
- How does this all get deleted?
  - You have to reverse the order of operations and 'undo' the whole config
  - \* TMOS has lots of issues here

As a result of this it's recommended for customers to use Imperative automation only for legacy environments. New environments should shift to a Declarative model.

### Task 1 – Build a Basic LTM Config

Perform the following steps to complete this task:

1. Expand the “Lab 1.6 – Build a Basic LTM Config” folder in the Postman collection
2. Click each Step in the folder and ‘Send’ the request. Verify each component is created on the BIG-IP device using the GUI.
3. After the steps are completed you should be able to connect to <http://10.1.20.129> in your browser.

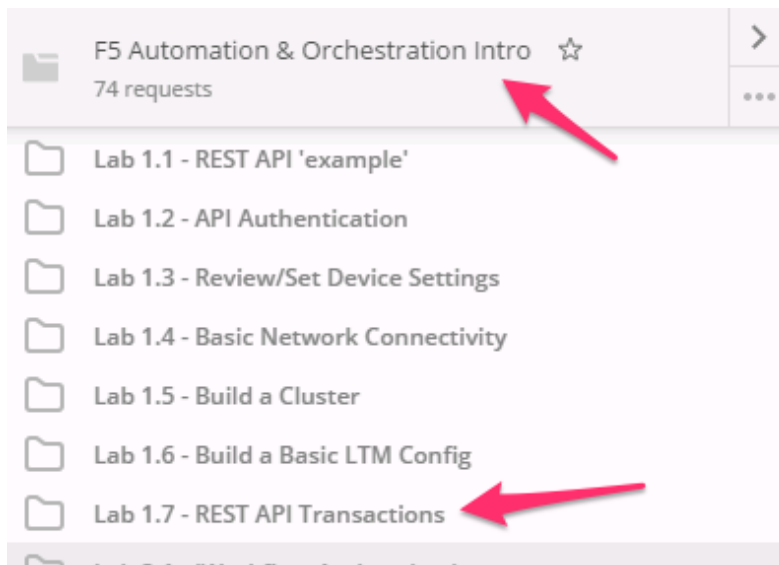
## 4.1.7 Lab 1.7 – REST API Transactions

### Task 1 – Create a Transaction

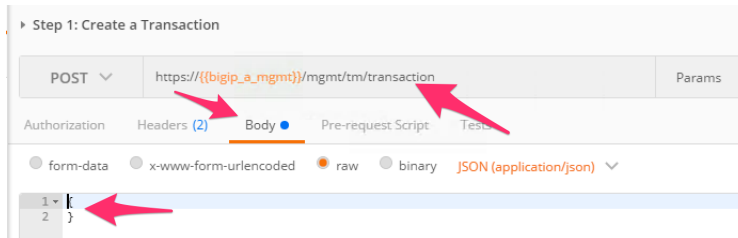
In this lab we will create a transaction using the REST API. Transactions are very useful in cases where you would want discreet REST operations to act as a batch operation. As a result the nature of a transaction is that either all the operations succeed or none of them do. This is very useful when creating a configuration that is linked together because it allows the roll back of operations in case one fails. All the commands issued, are queued one after one in the transaction. We will also review how to change the order of a queued command or remove a single command from the queued list before committing.

Perform the following steps to complete this task:

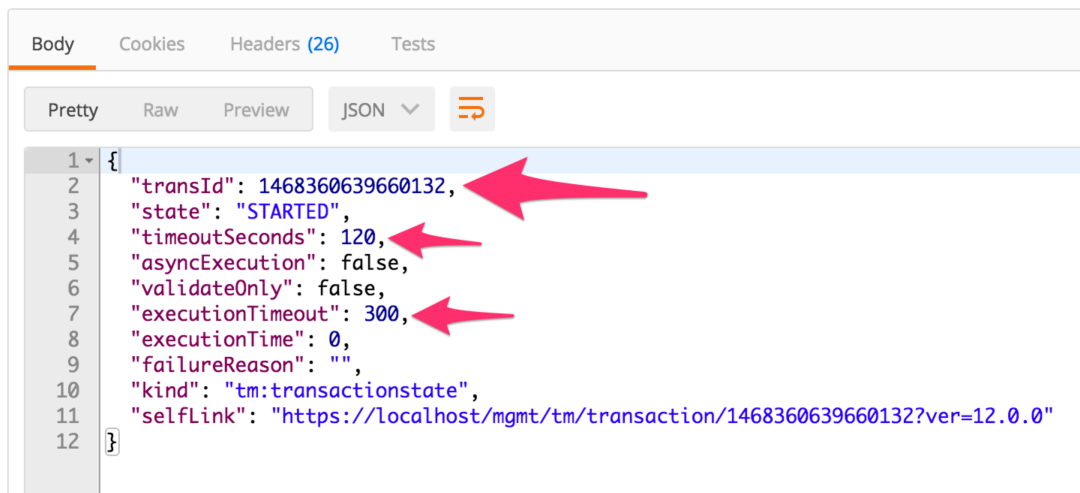
1. Expand the ‘Lab 1.7 – Rest API Transactions’ folder in the Postman collection:



- Click the 'Step 1: Create a Transaction' item. Examine the URL and JSON body. We will send a POST to the /mgmt/tm/transaction worker with an empty JSON body to create a new transaction.



- Click the 'Send' button to send the request. Examine the response and find the 'transId' attribute.



- Save the value of this attribute in the `bigip_transaction_id` environment variable. Additionally notice that there are timeouts for both the submission of the transaction and how long it should take to execute. Be aware that after the 'timeoutSeconds' value, this transId will be silently removed:

MANAGE ENVIRONMENTS

Manage Environments
Environment Templates

Edit Environment

INTRO - Automation & Orchestration Lab

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	bigip_a_mgmt	10.1.1.4	
<input checked="" type="checkbox"/>	bigip_b_mgmt	10.1.1.5	
<input checked="" type="checkbox"/>	iwf_mgmt	10.1.1.6	
<input checked="" type="checkbox"/>	bigip_a_auth_token		
<input checked="" type="checkbox"/>	bigip_b_auth_token		
<input checked="" type="checkbox"/>	bigip_transaction_id		
<input checked="" type="checkbox"/>	iwf_auth_token		

- Click the 'Step 2: Add to Transaction: Create a HTTP Monitor' item in the Postman collection. This request is the same as a non-transaction enabled request in terms of the request type (POST), URI and JSON body. The difference is we add a X-F5-REST-Coordination-Id header with a value of the transId attribute to add it to the transaction:

Step 2: Add To Transaction: Create a HTTP Monitor

POST
https://{{bigip\_a\_mgmt}}/mgmt/tm/ltn/monitor/http
Params

Authorization
Headers (3)
Body
Pre-request Script
Tests

	Key	Value
<input checked="" type="checkbox"/>	Content-Type	application/json
<input checked="" type="checkbox"/>	X-F5-REST-Coordination-Id	{{bigip_transaction_id}}
<input checked="" type="checkbox"/>	X-F5-Auth-Token	{{bigip_a_auth_token}}
	New key	value



- Click the 'Send' button and examine the response
- Examine and click 'Send' on Steps 3-6 in the collection
- Click 'Step 7: View the Transaction queue'. Examine the request type and URI and click 'Send'. This request allows you to see the current list of commands (ordered) that are in the transaction.

## Task 2 – Modify a Transaction


- Click the 'Step 8: View queued command 4 from Transaction' item in the collection. Examine the request type and URI. We will GET the queued command number 4 from the transaction list.





Step 8: View queued command 4 from Transaction

GET  https://{bigip\_a\_mgmt}/mgmt/tm/transaction/{bigip\_transaction\_id}/commands/4  Params


Authorization Headers (2) Body Pre-request Script Tests

Type No Auth 

Body Cookies Headers (21) Tests Status: 200 OK



Pretty Raw Preview JSON  

```
1 {
2   "method": "POST",
3   "uri": "https://localhost/mgmt/tm/ltm/profile/tcp",
4   "body": {
5     "name": "Lab1.7_tcp_clientside",
6     "nagle": "disabled",
7     "sendBufferSize": "16000"
8   },
9   "evalOrder": 4,
10  "commandId": 4,
11  "kind": "tm:transaction:commandsstate",
12  "selfLink": "https://localhost/mgmt/tm/transaction/1494358032450450/commands/4?ver=12.1.1"
13 }
```



2. Click the 'Step 9: Change Eval Order 4 ->1' item in the collection. Examine the request type, URI and JSON body. We will PATCH our transaction resource and change the value of the 'evalOrder' attribute, from 4 to 1, to move at the first position of the transaction queue:


Step 9: Change Eval Order 4 -> 1


PATCH  https://{{bigip\_a\_mgmt}}/mgmt/tm/transaction/{{bigip\_transaction\_id}}/commands/4  Params

Authorization Headers (2) Body ● Pre-request Script Tests

Type No Auth

Body Cookies Headers (21) Tests Status: 200 C

Pretty Raw Preview JSON 



```
1 {
2   "method": "POST",
3   "uri": "https://localhost/mgmt/tm/ltm/profile/tcp",
4   "body": {
5     "name": "Lab1.7_tcp_clientside",
6     "nagle": "disabled",
7     "sendBufferSize": "16000"
8   },
9   "evalOrder": 1, 
10  "commandId": 4,
11  "kind": "tm:transaction:commandsstate",
12  "selfLink": "https://localhost/mgmt/tm/transaction/1494358032450450/commands/4?ver=12.1.1"
13 }
```

3. Click the 'Step 10: View the Transaction queue changes' item in the collection. Examine that the transaction number 4 has moved into position 1 and all other transactions eval order has moved accordingly.


### Task 3 – Commit a Transaction


1. Click the 'Step 11: Commit the Transaction' item in the collection. Examine the request type, URI and JSON body. We will PATCH our transaction resource and change the value of the 'state' attribute to submit the transaction:

Step 11: Commit the Transaction

PATCH  https://{{bigip\_a\_mgmt}}/mgmt/tm/transaction/{{bigip\_transaction\_id}}  Params

Authorization Headers (2) Body ● Pre-request Script Tests

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) 

```
1 {
2   "state": "VALIDATING" 
3 }
```

2. Click the 'Send' button and examine the response.
3. Verify the config was created using TMUI or REST requests.

**Warning:** When sending the Header `X-F5-REST-Coordination-Id`, the system assumes you want to add an entry in the transaction queue. You **MUST** remove this header if you want to issue transaction queue changes (like deleting an entry from the queue, changing the order, committing a transaction). If you don't remove the header in that specific case, the system will send a 400 with the following type of error: "message": "Transaction XXXXX operation . . . is not allowed to be added to transaction."

## 4.2 Module 2 – iWorkflow

In this module we will explore how to use F5's iWorkflow platform to further abstract application services and deliver those services to tenants. iWorkflow has two main purposes in the Automation & Orchestration toolchain:

- Provide simplified but customizable Device Onboarding workflows
- Provide a tenant/provider interface for L4 – L7 service delivery

When moving to an iWorkflow based toolchain it's important to understand that L1-3 Automation (Device Onboarding, Networking, etc) and L4-7 (Deployment of Virtual Servers, Pools, etc) are separated and delivered by different features.

L1-3 Networking and Device Onboarding are delivered by 'Cloud Connectors' that are specific to the third party technology ecosystem (e.g. vCMP, AWS, Cisco APIC, VMware NSX, BIG-IP, etc).

L4-7 service delivery is accomplished by:

- Declarative: Consuming F5 iApp templates from BIG-IP devices and creating a Service Catalog.
- Imperative: Consuming the iWorkflow REST Proxy to drive API calls to BIG-IP devices

The labs in the module will focus on the high level features in place to achieve full L1-7 automation. As mentioned above, iApps are a key component of this toolchain. For our purposes we will use the `f5.http` iApp to create simple examples. For more advanced use cases it's often required to use a 'Declarative' or 'Deployment-centric' iApp template. A supported template of this nature called the App Services Integration iApp is available at <https://github.com/F5Networks/f5-application-services-integration-iApp> for this purpose.

### 4.2.1 Lab 2.1 – iWorkflow Authentication

iWorkflow supports the same authentication mechanisms as BIG-IP (HTTP BASIC, Token Based Auth). In this lab we will quickly review TBA on iWorkflow.

#### Task 1 – Token Based Authentication

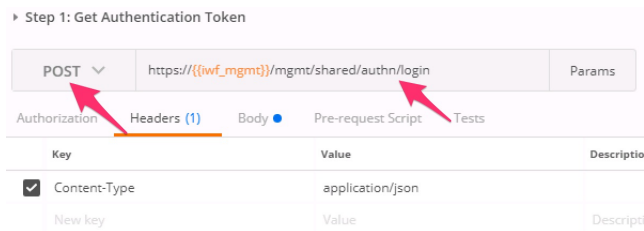
In this task we will demonstrate TBA using the local authentication database, however, authentication to external providers is fully supported.

For more information about external authentication providers see the section titled "**About external authentication providers with iControl REST**" in the iControl REST API User Guide available at <https://devcentral.f5.com>

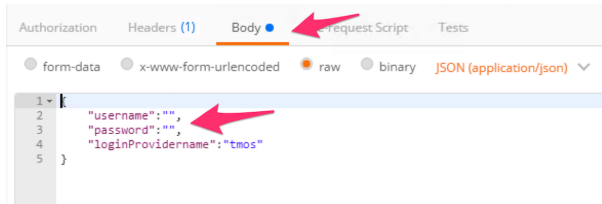
Perform the following steps to complete this task:

1. Click the 'Step 1: Get Authentication Token' item in the Lab 2.1 Postman Collection

2. Notice that we are sending a POST request to the `/mgmt/shared/authn/login` endpoint.



3. Click the 'Body' tab and examine the JSON that we will send to iWorkflow to provide credentials:



4. Modify the JSON body and add the required credentials (admin/admin). Then click the 'Send' button.
5. Examine the response status code. If authentication succeeded and a token was generated the response will have a 200 OK status code. If the status code is 401 then check your credentials:

#### Successful:



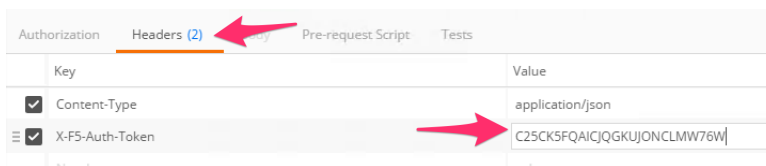
#### Unsuccessful:



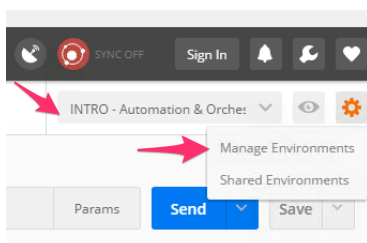
6. Once you receive a 200 OK status code examine the response body. The various attributes show the parameters assigned to the particular token. Find the 'token' attribute and copy it into your clipboard (Ctrl+c) for use in the next step:

```
1 {
2   "username": "admin",
3   "loginReference": {
4     "link": "https://localhost/mgmt/cm/system/authn/providers/local/login"
5   },
6   "loginProviderName": "local",
7   "token": {
8     "token": "PX5Z4NE2KDYTJGGRBOAYYAUJ4J",
9     "name": "PX5Z4NE2KDYTJGGRBOAYYAUJ4J",
10    "userName": "admin",
11    "authProviderName": "local",
12    "user": {
13      "link": "https://localhost/mgmt/shared/authz/users/admin"
14    },
15    "groupReferences": [],
16    "timeout": 1200,
17    "refreshToken": "2016-07-06T03:03:40-07:00"
```

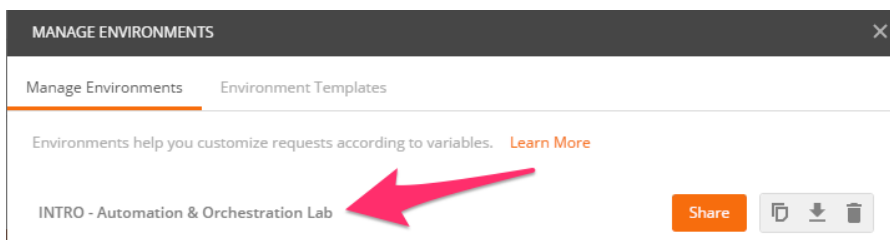
7. Click the 'Step 2: Verify Authentication Works' item in the Lab 2.1 Postman collection. Click the 'Headers' tab and paste the token value copied above as the VALUE for the X-F5-Auth-Token header. This header is required to be sent on all requests when using token based authentication.



8. Click the 'Send' button. If your request is successful you should see a '200 OK' status and a listing of the 'Ibm' Organizing Collection.
9. We will now update your Postman environment to use this auth token for the remainder of the lab. Click the Environment menu in the top right of the Postman window and click 'Manage Environments':



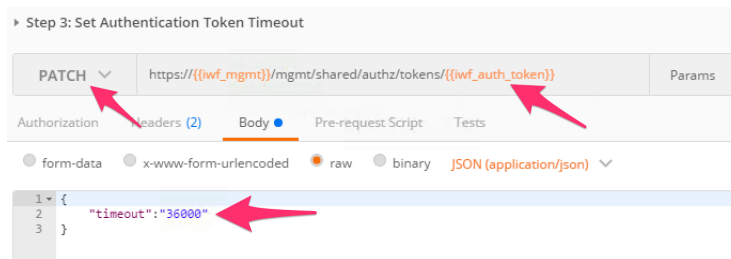
10. Click the 'INTRO – Automation & Orchestration Lab' item:



11. Update the value for 'iwf\_auth\_token' by Pasting (Ctrl-v) in your auth token:

MANAGE ENVIRONMENTS		
Manage Environments Environment Templates		
Edit Environment		
INTRO - Automation & Orchestration Lab		
Key	Value	Bulk Edit
<input checked="" type="checkbox"/> bigip_a_mgmt	10.1.1.4	
<input checked="" type="checkbox"/> bigip_b_mgmt	10.1.1.5	
<input checked="" type="checkbox"/> iwf_mgmt	10.1.1.6	
<input checked="" type="checkbox"/> bigip_a_auth_token	QJXK6HQWZFW35VC3JRZOXWNNNQ	
<input checked="" type="checkbox"/> bigip_b_auth_token		
<input checked="" type="checkbox"/> transaction_id	1491161090035223	
<input checked="" type="checkbox"/> iwf_auth_token	C25CK5FQAICJQGKUJONCLMW76W	
<input checked="" type="checkbox"/> iwf_pool_uuid		
<input checked="" type="checkbox"/> iwf_bigip_a_uuid		

- Click the 'Update' button and then close the 'Manage Environments' window. Your subsequent requests will now automatically include the token.
- Click the 'Step 3: Set Authentication Token Timeout' item in the Lab 1.2 Postman collection. This request will PATCH your token Resource (check the URI) and update the timeout attribute so we can complete the lab easily. Examine the request type and JSON Body and then click the 'Send' button. Verify that the timeout has been changed to '36000' in the response:



## 4.2.2 Lab 2.2 – Discover BIG-IP Devices

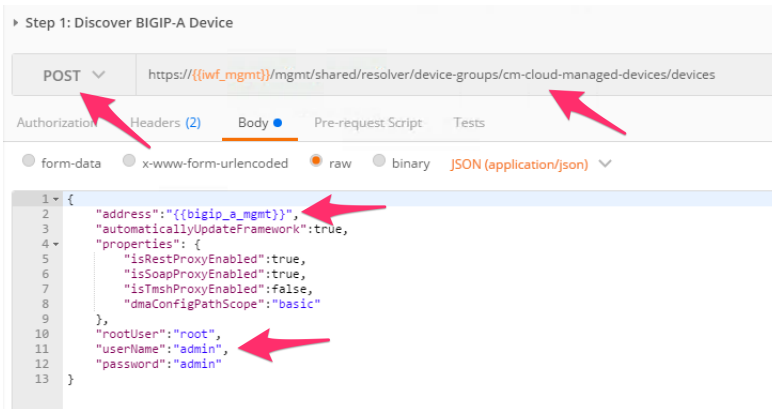
In order for iWorkflow to interact with a BIG-IP device it must be discovered by iWorkflow. The device discovery process leverages the existing CMI Device Trust infrastructure on BIG-IP. Currently there is a limitation that a single BIG-IP device can only be 'discovered' by ONE of iWorkflow or BIG-IQ CM at a time. In this lab we will discover the existing BIG-IP devices from your lab environment.

### Task 1 – Discover BIG-IP Devices

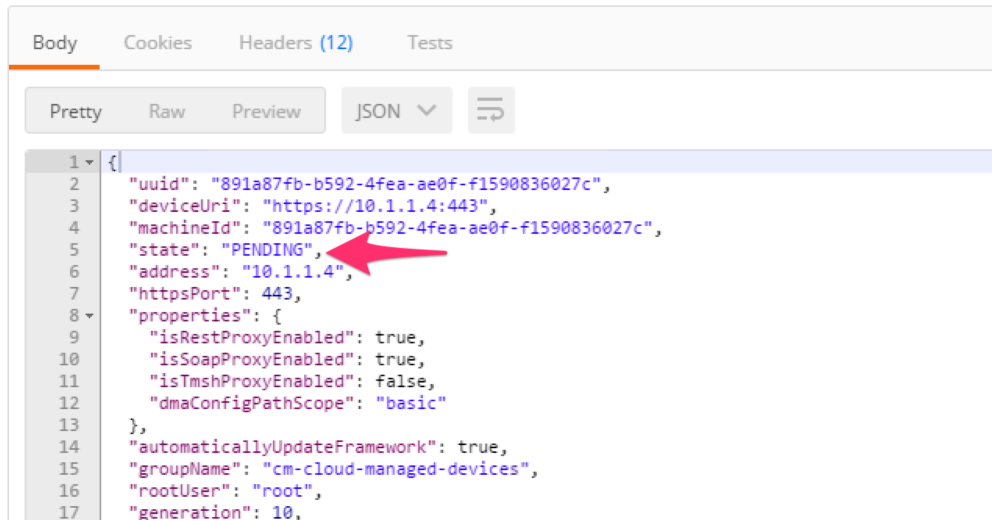
Perform the following steps to complete this task:

- Expand the "Lab 2.2: Discover & License BIG-IP Devices" folder in the Postman collection
- Open a Google Chrome window/tab to your iWorkflow device (<https://10.1.1.6>) and login with default credentials (admin/admin). You can use this window to monitor actions while they are being performed in Postman. Find the 'Devices' pane and make it viewable if it isn't already.
- Click the "Step 1: Discover BIGIP-A Device" item in the Postman collection. This request will perform a POST to the `/mgmt/shared/resolver/device-groups/`

cm-cloud-managed-devices/devices worker to perform the device discovery process. Examine the JSON body so you understand what data is sent to perform the discovery process:



4. Click the 'Send' button. Examine the response and monitor the iWorkflow Chrome window you opened previously.



5. Copy the 'uuid' attribute for BIGIP-A and populate the 'iwf\_bigip\_a\_uuid' Postman environment variable with the value:

```

1 {
2   "uuid": "a53991f9-a0de-4cd5-a342-feb435d7b1d",
3   "deviceUri": "https://10.1.1.4:443",
4   "machineId": "a53991f9-a0de-4cd5-a342-feb435d7b1d",
5   "state": "PENDING",
6   "address": "10.1.1.4",
7   "httpsPort": 443,
8   "properties": {
9     "isRestProxyEnabled": true,
10    "isSoapProxyEnabled": true,
11    "isTmshProxyEnabled": false,
12    "dmaConfigPathScope": "basic"
13  },
14  "automaticallyUpdateFramework": true,
15  "groupName": "cm-cloud-managed-devices",
16  "rootUser": "root",
17  "generation": 1,
18  "lastUpdateHicross": 1491161906297581,
19  "kind": "shared:resolver:device-groups:restdeviceresolverdevicestate",
20  "selfLink": "https://localhost/mgmt/shared/resolver/device-groups/cm-clo
21 }

```

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> bigip_a_mgmt	10.1.1.4	
<input checked="" type="checkbox"/> bigip_b_mgmt	10.1.1.5	
<input checked="" type="checkbox"/> iwf_mgmt	10.1.1.6	
<input checked="" type="checkbox"/> bigip_a_auth_token	QJXK6HQWZFW35VC3JRZOXWNNQ	
<input checked="" type="checkbox"/> bigip_b_auth_token		
<input checked="" type="checkbox"/> transaction_id	1491161090035223	
<input checked="" type="checkbox"/> iwf_auth_token	CXJV40BFJZI3UJAHJISIMYD773	
<input checked="" type="checkbox"/> iwf_pool_uuid		
<input checked="" type="checkbox"/> iwf_bigip_a_uuid	a53991f9-a0de-4cd5-a342-feb435d7b1d	
<input checked="" type="checkbox"/> iwf_connector_uuid		

- Click the “Step 2: Discover BIGIP-B Device” item in the collection.
- Click the “Step 3: Get Discovered Devices” item in the collection. We will GET the devices collection and verify that both BIG-IP devices show a ‘state’ of ‘ACTIVE’:

```

1 {
2   "items": [
3     {
4       "uuid": "a53991f9-a0de-4cd5-a342-feb435d7b1d",
5       "deviceUri": "https://10.1.1.4:443",
6       "machineId": "a53991f9-a0de-4cd5-a342-feb435d7b1d",
7       "state": "ACTIVE",
8       "address": "10.1.1.4",
9       "httpsPort": 443,
10      "hostname": "ip-10-1-1-4.us-west-2.compute.internal",
11      "version": "12.1.1",
12      "product": "BIG-IP",
13      "edition": "Hotfix HF1",
14      "build": "1.0.196",
15      "restFrameworkVersion": "13.1.0-0.0.5807",
16      "managementAddress": "10.1.1.4",
17      "mcpDeviceName": "/Common/bigip1",
18      "trustDomainGuid": "0a5dd62c-b92f-4b18-8c1d0a65904a6233",

```

## 4.2.3 Lab 2.3 – Create Tenant & BIG-IP Connector

iWorkflow implements a Tenant/Provider interface to enable abstracted deployment of L4-7 into various environment. In conjunction iWorkflow Connectors serve as the L1-3 Network and Device Onboarding automation component in the automation toolchain. iWorkflow supports Connectors for various vendor integrations (F5 vCMP, F5 BIG-IP, Cisco APIC, vmWare NSX, etc.) In this lab we will create a ‘BIG-IP Connector’ for the BIG-IP devices in the lab deployment. This connector will then allow you to drive a fully automated deployment from the iWorkflow Service Catalog.

### Task 1 – Create a Tenant and Tenant User

In this task we will create a Local Connector that is linked to our BIG-IP devices. The Local Cloud Connector is DSC aware and will automatically detect that the BIG-IP devices are clustered and configure itself accordingly.

Perform the following steps to complete this task:

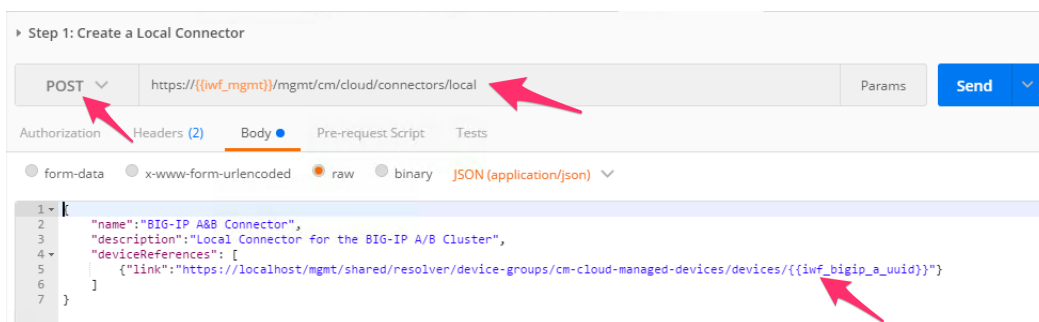
- Expand the “Lab 2.3 – Create Tenant & Local Connector” folder in the Postman collection.



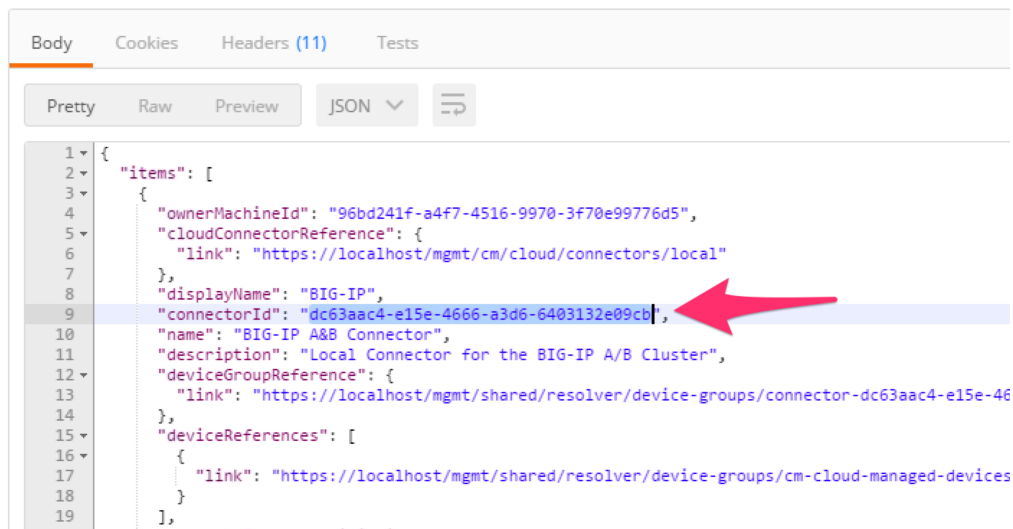
2. Click the “Step 1: Create iWorkflow Tenant” item in the collection and click ‘Send’. This request will create a tenant name `MyTenant`.
3. Click the “Step 2: Create Tenant User” item in the collection and click ‘Send’. This request will create a **tenant** user.
4. Click the “Step 3: Assign User to Tenant Admin Role” item in the collection and click ‘Send’. This request will assign the Admin role for the `MyTenant` tenant to the `tenant` user.

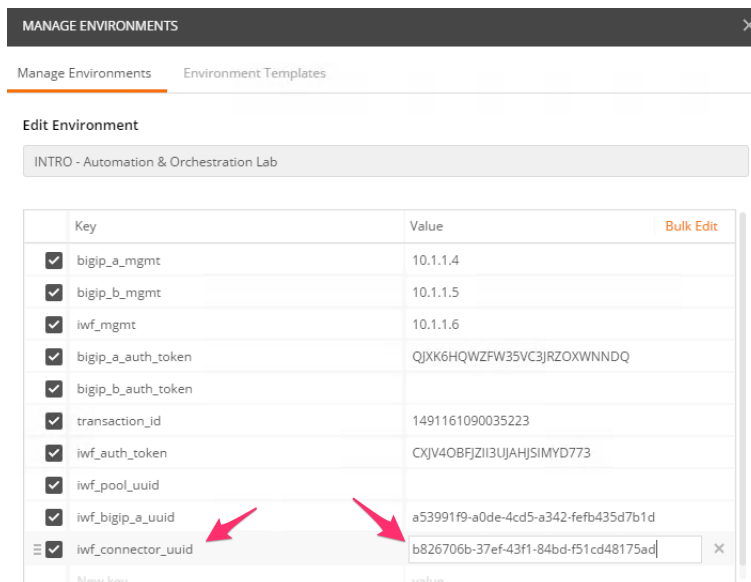
## Task 2 - Create a Local Connector

1. Click the “Step 4: Create a Local Connector” item in the collection. We will create a new connector by performing a POST to the local connector collection. If you examine the JSON body you can see we are providing a reference to the URL for the BIG-IP-A device (using the UUID environment variable we populated earlier):



2. Click the ‘Send’ button to create the connector.
3. Click the “Step 5: Get Local Connectors” item in the collection and click ‘Send’. Examine the output to see how the connector was configured. Take note of the reference to the ‘device-group’. This is how the connector determines the HA state of the underlying BIG-IP devices. Find the ‘connectorId’ of the connector and update your Postman environment to include the ‘connectorId’ as the value of the ‘iwf\_connector\_uuid’ variable:





- Click the “Step 6: Assign Connector to Tenant” item in the collection. This request will assign this connector to to the ‘MyTenant’ tenant allowing service deployments from that tenant. Click the ‘Send’ button and examine the response.

## 4.2.4 Lab 2.4 – Deploy L4-7 Services

To drive iApp automation-based L4-7 deployments, iWorkflow includes the capability to create a Tenant Service Catalog via L4 – L7 Service Templates. This model of deployment enables Declarative automation of F5 L4-7 services provided the underlying iApp templates are designed with a declarative presentation layer in mind. To demonstrate this capability we will create a simple Service Catalog Template and deploy and application from a tenant on our BIG-IP devices using the App Services iApp.

### Task 1 - Install the App Services iApp on iWorkflow

iWorkflow serves as the Source-of-Truth for iApp templates. As a result iApp templates that will be used to automate deployments on BIG-IP must be installed on iWorkflow first. Once installed, iWorkflow will automatically determine when a template needs to be installed on BIG-IP and perform the needed actions.

**Note:** iApp template installation on BIG-IP devices occurs during the **first** service deployment to a device.

To assist in deployment of the App Services iApp template and it’s associated sample service templates a Postman collection has been created. We will first import the collection into Postman and then use it to install the template into iWorkflow.

Perform the following steps to complete this task:

- Import the following collection URL using ‘Import’ -> ‘Import from Link’:

```
https://raw.githubusercontent.com/f5devcentral/f5-automation-labs/ondemand/postman_collections/AppSvcs_iApp_Workflows.postman_collection.json
```

- Expand the AppSvcs\_iApp\_Workflows collection. Then open the 2\_Install\_on\_iWorkflow folder and click the Install AppSvcs Template on iWorkflow item.

3. You can examine the Body of this request, however, understand that it contains the minified code that comprises the iApp and will not be very readable. This collection uses the underlying variables that have already been set (`iwf_mgmt` and `iwf_auth_token`) to make installation simple.
4. Click the 'Send' button to install the iApp.

## Task 2 – Create the f5-http-lb L4–7 Service Template

An L4-7 Service Deployment on iWorkflow is driven by the creation of an L4 – L7 Service Template. These templates allow a provider (administrator) to specify the values of specific fields from an origin iApp presentation layer. Additionally, the provider also defines the tenant interface to the service by marking which fields are '**Tenant Editable**' and therefore visible during service deployment from the tenant. You can think of a Service Catalog Template and a filter that allows the vast majority of fields to be filled in or defaulted while only exposing the minimal set of fields required to deploy a service.

In this task we will create a Service Catalog Template that utilizes the App Services iApp you just installed.

Perform the following steps to complete this task:

1. Expand the `3_iWorkflow_Service_Templates_Examples` folder of the `AppSvc_IApp_Workflows` collection
2. Click the "f5-http-lb Template" item in the collection. This request is pre-built and will create a new Service Template using the App Services iApp. Click the 'Send' button to create the template.
3. Open a Chrome tab to iWorkflow (<https://10.1.1.6>) and login with admin/admin credentials. Expand the 'Service Templates' pane and double-click the "f5-http-lb" template. Notice various defaults have been populated (e.g. port '80' for the `pool__port` variable) and some fields have been marked as 'Tenant Editable':

The screenshot shows the 'Service Templates' interface for the 'f5-http-lb' template. The interface is divided into several sections:

- Properties:**
  - Name: f5-http-lb
  - Input Parameters: Common Options (selected), All Options
  - Cloud: All Clouds
  - Base Template: appsvcs\_integration\_v2.0.003
- Sections:**
  - Virtual Server Listener & Pool Configuration:**

Name	Description	Default Value	Tenant Editable
pool__addr	Virtual Server: Address		<input checked="" type="checkbox"/>
pool__port	Virtual Server: Port	80	<input checked="" type="checkbox"/>
pool__Members	Pool: Members		
  - Virtual Server Configuration:**

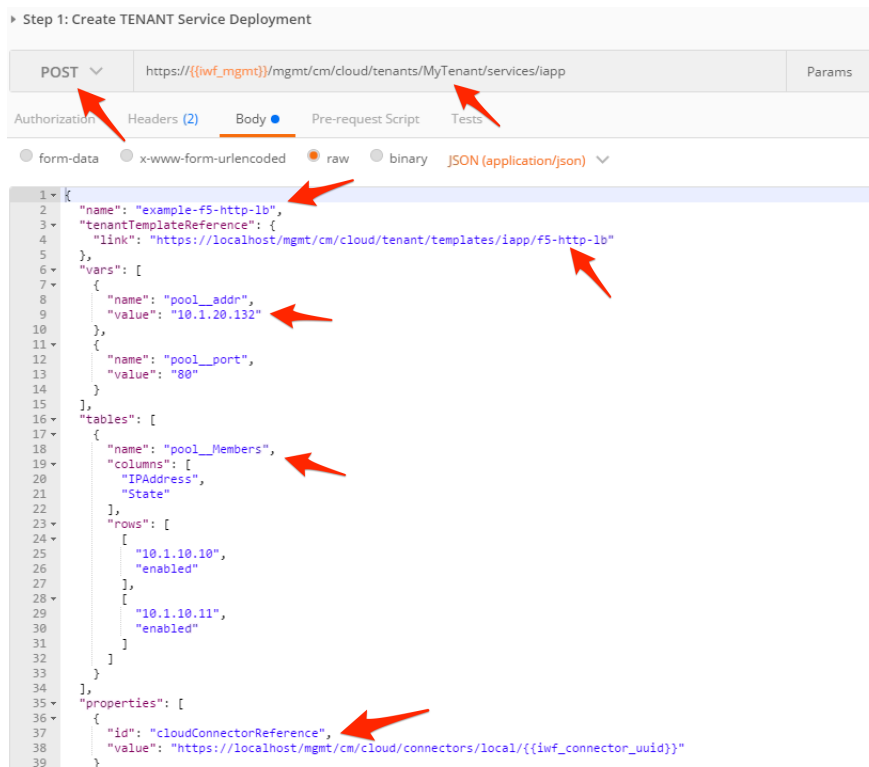
Name	Description	Default Value	Tenant Editable
vs__ProfileClientS...	Virtual Server: Client SSL Certificate		<input type="checkbox"/>
vs__ProfileClientS...	Virtual Server: Client SSL Key		<input type="checkbox"/>

## Task 2 – Tenant L4-7 Service Deployment

In this task we will perform CRUD operations based on a deployment of the Service Catalog Template created in the previous task.

Perform the following steps to complete this task:

1. Open a new Chrome tab to iWorkflow (<https://10.1.1.6>) and login with the credentials Username: tenant, Password: tenant. Expand the 'L4-L7 Services' pane.
2. Switch back to F5 Automation & Orchestration Intro Postman Collection and click the "Step 1: Create TENANT Service Deployment" item in Lab 2.4. Examine the URL and JSON body. We will be creating a new Tenant Service Deployment under 'MyTenant' with the properties marked as 'Tenant Editable' provided:



3. Click the 'Send' button to create the Service Deployment. Examine the response. The iWorkflow GUI in your Chrome tab will also reflect a new item in the Services pane:

**L4-L7 Services** + **example-f5-http-lb** Save Delete Cancel

1 item total

**example-f5-http-lb**  
Mytenant  
clientside-bits in: 0  
clientside-bits out: 0

Properties Statistics

General Properties

Name	example-f5-http-lb
Status	Application Service is healthy.
L4-L7 Service Template	f5-http-lb
Cloud	BIG-IP A&B Connector

Customize L4-L7 Server Template

Pool Addr	10.1.20.132
Pool Port	80

	Ipaddress	State		
Pool: Members	10.1.10.10	enabled	+	x
	10.1.10.11	enabled	+	x

- Open a Chrome tab to BIGIP-A. Click on iApps -> Application Services -> Applications -> example-f5-http-lb to view the config that was deployed on BIG-IP:

Main Help About

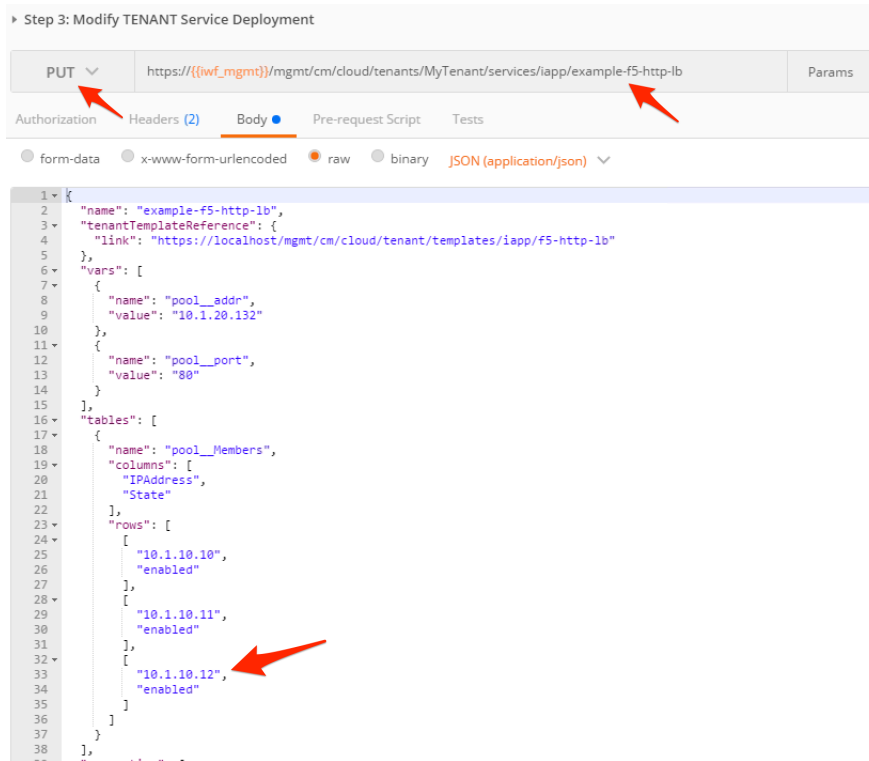
iApps » Application Services : Applications » **example-f5-http-lb**

Properties Reconfigure **Components**

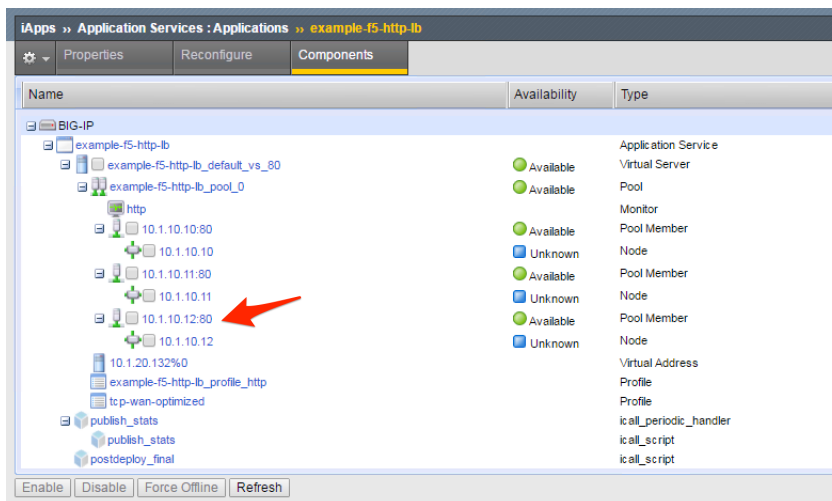
Name	Availability	Type
BIG-IP		
example-f5-http-lb	Available	Application Service
example-f5-http-lb_default_vs_80	Available	Virtual Server
example-f5-http-lb_pool_0	Available	Pool
http	Available	Monitor
10.1.10.10:80	Available	Pool Member
10.1.10.10	Unknown	Node
10.1.10.11:80	Available	Pool Member
10.1.10.11	Unknown	Node
10.1.20.132%0		Virtual Address
example-f5-http-lb_profile_http		Profile
tcp-wan-optimized		Profile
publish_stats		icall_periodic_handler
publish_stats		icall_script
postdeploy_final		icall_script

Enable Disable Force Offline Refresh

- Go back to Postman and click the “Step 2: Get TENANT Service Deployment” item in the collection and click ‘Send’. This item is example of a GET of the service definition. The response should match what you see in the iWorkflow GUI when viewing the properties of a deployment.
- Click the “Step 3: Modify TENANT Service Deployment” item in the collection. This request is an example of an Update operation. Notice that we are sending a PUT request to the URL representing the service deployment. Examine the JSON body and note that in the ‘pool\_\_Members’ table there is an additional pool member with an IP of 10.1.10.12 that will be added. Click the ‘Send’ button to re-deploy the service:



7. Verify that the pool member was added on BIG-IP:



8. Go back to Postman and click the “Step 4: Delete TENANT Service Deployment” item. This item will send a DELETE request to the URL for the service deployment. Click ‘Send’ and verify that the deployment has been removed in the iWorkflow and BIG-IP GUIs.

## 4.2.5 Lab 2.5 – iWorkflow REST Proxy

In order to enable Imperative automation use cases, iWorkflow includes a REST proxy that allows pass-through of REST requests to devices managed by iWorkflow. The REST proxy feature allows customers to simplify automation by:

- Providing a centralized API endpoint for BIG-IP infrastructure

- No need to communicate with individual BIG-IP devices, only with iWorkflow
- Simplified authentication
  - Strong authentication can be implemented at iWorkflow rather than on each BIG-IP
- Simplified RBAC
  - RBAC can be implemented at iWorkflow for all devices rather than on individual devices in the environment

The rest proxy works by passing data sent to a specific URL through to the BIG-IP device. The root URL for a particular device's REST proxy is:

```
/mgmt/shared/resolver/device-groups/cm-cloud-managed-devices/devices/
<device\_uuid>/rest-proxy/
```

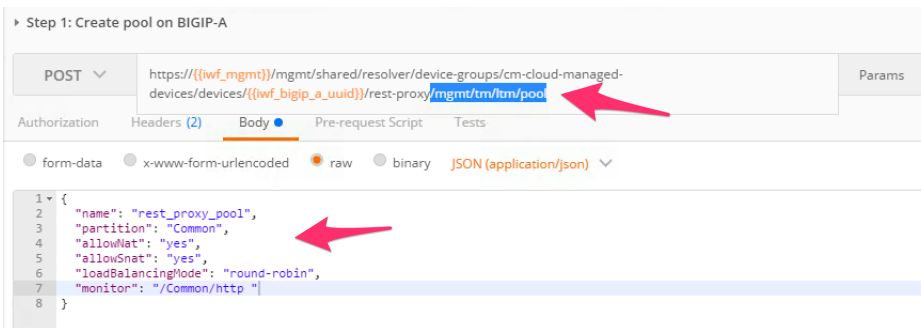
Any URL segments included after `.../rest-proxy/` are forwarded unaltered to the BIG-IP device. Query parameters (e.g. `?expandSubcollections=true`) are also passed unaltered along with the request type and request body.

### Task 1 – Perform REST operations via the REST Proxy

In this task we will perform a sample CRUD operation utilizing the REST Proxy. The intent of this task is to show the basic mechanism used to perform these tasks. Simply changing the URL to include the iWorkflow REST Proxy root for that device could easily change all the Imperative operations we have completed in this lab to use the REST Proxy.

Perform the following steps to complete this task:

1. Expand the “Lab 2.5 – iWorkflow REST Proxy” folder in the Postman collection.
2. Click the “Step 1: Create pool on BIGIP-A”. Examine the request type, URL and JSON body. Essentially we are performing a POST to the `/mgmt/tm/ltn/pool` collection on BIGIP-A. The last part of the URL includes this URI path (the part after `.../rest-proxy/`). The JSON body and all other parameters are passed unaltered. Also, notice that we are still using our iWorkflow Token to authenticate, not the BIG-IP one.



3. Click the “Send” button and examine the response.
4. Complete steps 2-5 for the remaining items in the “Lab 2.5 – iWorkflow REST Proxy” collection. Examine each request carefully so you understand what is happening.

## 4.3 Module 3 – f5-super-netops-container Toolkit

In this module we will explore how to use the **f5-super-netops** container toolkit to easily consume various F5 Automation, Orchestration Super Netops and DevOps tools and frameworks.

The f5-super-netops-container is meant to provide a simple way for users to quickly duplicate a standard automation and orchestration environment in your local machine/lab environment. The container is continuously updated to include the latest tools and documentation.

The labs in this module will show you how to install the f5-super-netops-container image, start it in your local environment and access various tools and documentation.

To install the f5-super-netops-container you need to be sure your system support running Docker Community Edition (CE). Please refer to <https://docs.docker.com/engine/installation/#platform-support-matrix> for more information.

This toolkit is fully open source and is on GitHub at <https://github.com/f5devcentral/f5-super-netops-container>

### 4.3.1 Lab 3.1 – Install Docker Community Edition (CE)

To use the f5-super-netops-container you first need to install Docker Community Edition on your system.

---

**Note:** If you are using an F5 provided lab environment, Docker CE has already been installed on the host named 'Docker Server'. Please SSH to that host and execute all `docker` commands there.

---

#### Task 1 – Install Docker CE

Please follow the instructions at <https://docs.docker.com/engine/installation/> to install Docker CE.

Once you have completed installing and successfully run the `hello-world` test you can continue to the next lab.

To test your setup with the `hello-world` container, you just need to run the following command

```
docker run hello-world
```

Example output:

```
$ sudo docker run --rm hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:



```
https://cloud.docker.com/
```

For more examples and ideas, visit:  
<https://docs.docker.com/engine/userguide/>

---

**Note:** The `--rm` option will delete the container as soon as it stops running.

If you see this message: *Cannot connect to the Docker daemon. Is the docker daemon running on this host?*, it is likely that you don't have enough privileges with your user, try to use `sudo` when executing docker commands.

If you want to remove the hello-world container, you can run the command `sudo docker rmi hello-world`. If your container is running, you cannot remove the image. You can issue the following commands in that case (this will stop ALL your container instances): `sudo docker stop $(docker ps -aq)`

---

### 4.3.2 Lab 3.2 – Obtain & Start the f5-super-netops-container Image

In this lab we will use the `docker` cli tools to obtain and start the f5-super-netops-container image.

#### Task 1 – Obtain the container image

Perform the following steps to complete this task:

1. Open a Command Prompt

---

**Note:** If you are using an F5 provided lab environment please SSH to the 'Docker Server' host and execute the following commands.

---

2. Execute `docker pull f5devcentral/f5-super-netops-container:base`

Example output:

```
$ docker pull f5devcentral/f5-super-netops-container:base
base: Pulling from f5devcentral/f5-super-netops-container
cfc728c1c558: Pull complete
d87c258a5fa6: Pull complete
c65d1b487eef: Pull complete
8dbc9686aafd: Pull complete
8780a91a51b1: Pull complete
adf738b585dc: Pull complete
03b3481bc590: Pull complete
8fc57fb32b1a: Pull complete
8f73f7c22240: Pull complete
7d94bd4c05e6: Pull complete
a0b407bf28b5: Pull complete
b97bd4f3c99d: Pull complete
f37519ea449c: Pull complete
Digest: sha256:20f501b4c46948d3e69ffd7793cbbf08ac18da5f89c6665f36af10bc7c2a89b4
Status: Downloaded newer image for f5devcentral/f5-super-netops-container:base
```

3. Execute `docker images`

Example output:

\$ docker images			
REPOSITORY		TAG	IMAGE ID
↪CREATED	SIZE		
f5devcentral/f5-super-netops-container		base	7712f3d38f6b
↪7 days ago	206 MB		

## Task 2 – Start the container image

To start the container we will execute the command:

```
docker run -p 8080:80 -p 2222:22 -it f5devcentral/f5-super-netops-container:base
```

The `-p` option publishes a L4 port from the container to the host. For example the `-p 8080:80` option will redirect port 8080 on the host system to port 80 in the container.

The `-it` option will make the session interactive and allocate a pseudo-TTY

The `f5devcentral/f5-super-netops-container:base` option is the name associated with the image we obtained in Task 1.

Perform the following steps to complete this task:

1. Execute `docker run -p 8080:80 -p 2222:22 -it f5devcentral/f5-super-netops-container:base`

---

**Note:** The image requires Internet connectivity to download the latest versions of tools and documentation. Please ensure you have proper connectivity from your host prior to starting the image. If you need to use a proxy please refer to the documentation at <https://docs.docker.com>

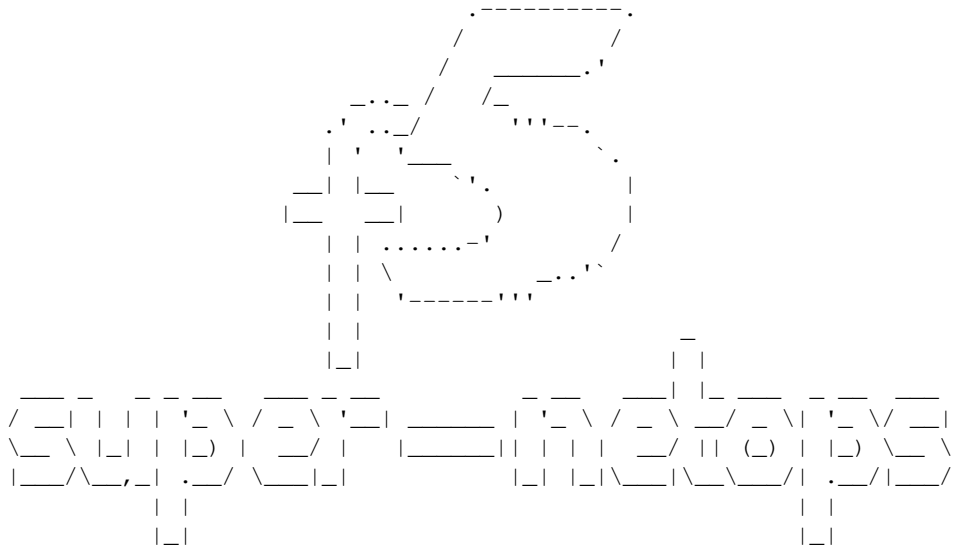
---

The image will now start and load resources from the Internet. This process may take a while depending on the speed of your connection. When the startup process is complete you will be presented with a `root` user prompt. You can interact with the image with standard Linux commands. In the next lab we will connect to the image via SSH and HTTP.

Example startup output:

```
$ docker run -p 8080:80 -p 2222:22 -it f5devcentral/f5-super-netops-container:base
[s6-init] making user provided files available at /var/run/s6/etc...exited 0.
[s6-init] ensuring user provided files have correct perms...exited 0.
[fix-attrs.d] applying ownership & permissions fixes...
[fix-attrs.d] done.
[cont-init.d] executing container initialization scripts...
[cont-init.d] done.
[services.d] starting services
[services.d] done.
Reticulating splines...
Becoming self-aware...
[cloneGitRepos] Retrieving repository list from https://github.com/f5devcentral/
↪f5-super-netops-container.git#master
[updateRepos] Processing /tmp/snops-repo/images/base/fs/etc/snopsrepo.d/base.json
[updateRepos] Processing /tmp/user_repos.json
[cloneGitRepos] Loading repositories from /home/snops/repos.json
[cloneGitRepos] Found 5 repositories to clone...
[cloneGitRepos][1/5] Cloning f5-sphinx-theme#master from https://github.com/
↪f5devcentral/f5-sphinx-theme.git
[cloneGitRepos][1/5] Installing f5-sphinx-theme#master
```

```
[cloneGitRepos][2/5] Cloning f5-super-netops-container#master from https://github.
↪com/f5devcentral/f5-super-netops-container.git
[cloneGitRepos][2/5] Installing f5-super-netops-container#master
[cloneGitRepos][3/5] Cloning f5-application-services-integration-iApp#develop_
↪from https://github.com/F5Networks/f5-application-services-integration-iApp.git
[cloneGitRepos][3/5] Installing f5-application-services-integration-iApp#develop
[cloneGitRepos][4/5] Cloning f5-postman-workflows#develop from https://github.com/
↪0xHiteshPatel/f5-postman-workflows.git
[cloneGitRepos][4/5] Installing f5-postman-workflows#develop
[cloneGitRepos][5/5] Cloning f5-automation-labs#master from https://github.com/
↪f5devcentral/f5-automation-labs.git
[cloneGitRepos][5/5] Installing f5-automation-labs#master
```



Welcome to the f5-super-netops-container. This image has the following services running:

```
SSH tcp/22
HTTP tcp/80
```

To access these services you may need to remap ports on your host to the local container using the command:

```
docker run -p 8080:80 -p 2222:22 -it f5devcentral/f5-super-netops-container:base
```

From the HOST perspective, this results in:

```
localhost:2222 -> f5-super-netops-container:22
localhost:8080 -> f5-super-netops-container:80
```

You can then connect using the following:

```
HTTP: http://localhost:8080
SSH: ssh -p 2222 snops@localhost
```

Default Credentials:

```
snops/default
root/default
```

```
Go forth and automate!

(you can now detach by using Ctrl+p+q)

[root@f5-super-netops] [/] #
```

### Task 3 - Detach/Re-attach the Container

When running containers it's important to understand that it will exit if the foreground process (in this case the shell) exits. For example, if you typed the `exit` command in the running container it will shutdown. In order to avoid this you should detach from the container once it has completed booting. You can still perform functions by using SSH to access the container as explained in the next lab.

#### Detach the Container

1. Enter `Ctrl+p+q` in the running TTY.

Example output:

```
[root@f5-super-netops] [/] #
[root@f5-super-netops] [/] #
[root@f5-super-netops] [/] # <enter Ctrl+p+q>
hostname:~ user$
```

2. Verify the container is still running by entering `docker ps`

Example output:

```
hostname:~ user$ docker ps
CONTAINER ID        IMAGE                                     COMMAND
↪                  CREATED            STATUS              PORTS
↪                  NAMES
b8c86fe5c7f1       f5devcentral/f5-super-netops-container:base  "/init /
↪snopsboot/..."   2 minutes ago      Up 2 minutes        0.0.0.0:2222->22/tcp,
↪0.0.0.0:8080->80/tcp  keen_ritchie
```

#### Re-attach the Container

1. Execute `docker ps`

Example output:

```
hostname:~ user$ docker ps
CONTAINER ID        IMAGE                                     COMMAND
↪                  CREATED            STATUS              PORTS
↪                  NAMES
b8c86fe5c7f1       f5devcentral/f5-super-netops-container:base  "/init /
↪snopsboot/..."   2 minutes ago      Up 2 minutes        0.0.0.0:2222->22/tcp,
↪0.0.0.0:8080->80/tcp  keen_ritchie
|-----|
^~ YOUR CONTAINER ID
```

2. Copy the value under the `CONTAINER ID` column that correspond to the `f5devcentral/f5-super-netops-container:base` image.

3. Execute `docker attach <container_id>`
4. You may have to hit `<Enter>` to display the command prompt
5. Detach the container again by entering `<Ctrl+p+q>`

### 4.3.3 Lab 3.3 – Connect to f5-super-netops-container

In the previous lab we started the container image and were presented with a root command prompt. In order to support use the container and its associated tools properly you connect via SSH and/or HTTP.

#### Task 1 – Connect via SSH

To connect to the image via SSH we must use the published port specified in the `docker run` command. To review, the command used to start the container was:

```
docker run -p 8080:80 -p 2222:22 -it f5devcentral/f5-super-netops-container:base
```

This will publish the standard SSH service on TCP/22 to TCP/2222 on the Docker host. In the case of the SSH service the following mapping applies:

```
localhost:2222 -> f5-super-netops-container:22
```

---

**Note:** If you are using an F5 provided lab environment please use the SSH client and connect to the 'f5-super-netops-container SSH' item

---

Additionally the container includes the `snops` user with a password of `default`. To connect to the container execute the following command or it's OS-specific equivalent:

```
ssh -p 2222 snops@localhost
```

---

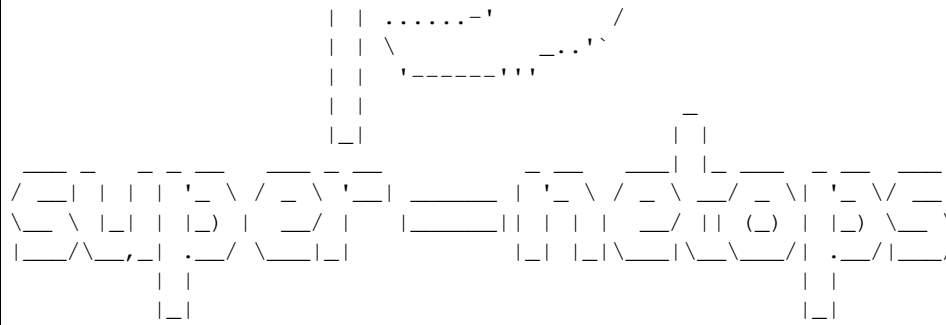
**Note:** The host SSH keys are regenerated each time the container boots. As a result you may receive an error when trying to connect indicating the host key has changed. This error is safe to ignore in this case and can be resolved by removing the key from `~/.ssh/known_hosts`. You can also configure your local SSH config by adding the following to `~/.ssh/config`:

```
Host localhost
  Port 2222
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
```

Example output:

```
$ ssh -p 2222 snops@localhost
Warning: Permanently added '[localhost]:2222' (ECDSA) to the list of known hosts.
snops@localhost's password:
```





Welcome to the f5-super-netops-container. This image has the following services running:

```
SSH  tcp/22
HTTP tcp/80
```

To access these services you may need to remap ports on your host to the local container using the command:

```
docker run -p 8080:80 -p 2222:22 -it f5devcentral/f5-super-netops-container:base
```

From the HOST perspective, this results in:

```
localhost:2222 -> f5-super-netops-container:22
localhost:8080 -> f5-super-netops-container:80
```

You can then connect using the following:

```
HTTP: http://localhost:8080
SSH:  ssh -p 2222 snops@localhost
```

Default Credentials:

```
snops/default
root/default
```

Go forth and automate!

```
[snops@f5-super-netops] [~] $
```

## Task 2 – Connect via HTTP

To connect to the image via HTTP we must use the published port specified in the `docker run` command. To review the command used to start the container was:

```
docker run -p 8080:80 -p 2222:22 -it f5devcentral/f5-super-netops-container:base
```

This will publish the standard HTTP service on TCP/80 to TCP/8080 on the Docker host. In the case of the HTTP service the following mapping applies:

```
localhost:8080 -> f5-super-netops-container:80
```

---

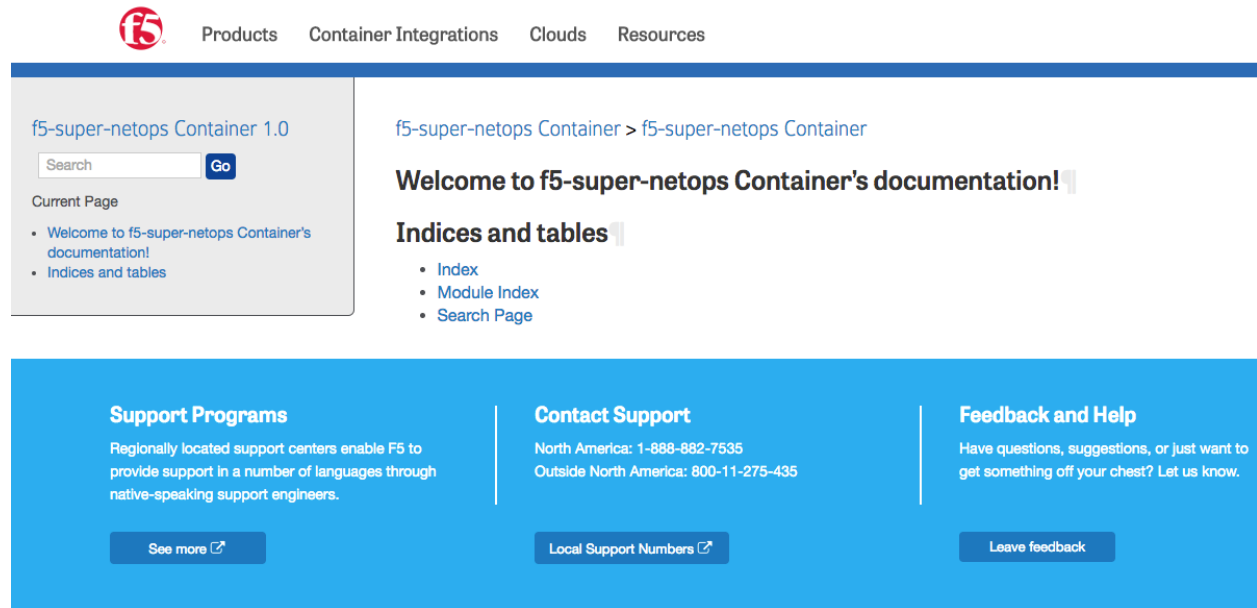
**Note:** If you are using an F5 provided lab environment please use the browser and click the ‘Super Netops Container’ bookmark.

---

To connect via HTTP open a web browser and enter the URL:

`http://localhost:8080/start`

You should see a page like this:



## 4.4 Module 4 – f5-postman-workflows & f5-newman-wrapper

In the previous modules you may have found the tasks associated with checking various response values and populating environment variables very tedious. In addition to being tedious these tasks are not fundamentally automatable due to the requirement for human interaction.

In order to assist users with automating the F5 BIG-IP platform we have developed a set of tools that can be used with the Postman REST Client (<http://getpostman.com>). The purpose of the tools are:

- f5-postman-workflows
  - Provide re-usable JavaScript functions that ease testing of API responses and populating environment variables
  - Implement a delay-based polling mechanism
- f5-newman-wrapper
  - Allow users to easily assemble Postman collections into workflows
  - Enabled integration with third-party tools such as Ansible, Chef & Puppet

The framework allows collection developers to create automatable collections that include full testing of response values, population of environment variables to establish chains of requests and time-based polling to allow long-lived API processes time to complete.

Users can then interact with these collections via the Postman GUI client, run the collections with the Postman Runner or the Newman CLI client.

This lab module will walk you through using the tools. If you are interested in developing collections using the f5-postman-workflows framework please visit the official GitHub repository at <https://github.com/0xHiteshPatel/f5-postman-workflows>

### 4.4.1 Lab 4.1 – Install the f5-postman-workflows Framework

In this lab you will walk through installing the f5-postman-workflows framework into the Postman REST Client.

#### Task 1 - Import the f5-postman-workflows Postman Collection

In this task you will import a Postman collection that contains installation helpers, Examples and a automated test framework. The collection is installed from the f5-postman-workflows GitHub repository.

Perform the following steps to complete this task:



1. Open the Postman Client on your jumphost by clicking the  icon
2. Click the 'Import' button in the top left of the Postman window
3. Click the 'Import from Link' tab. Paste the following URL into the text box and click 'Import'  

```
https://raw.githubusercontent.com/0xHiteshPatel/f5-postman-workflows/master/F5_Postman_Workflows.postman_collection.json
```
4. You should now see a collection named 'F5\_Postman\_Workflows' in your Postman Collections sidebar

#### Task 2 - Install f5-postman-workflows into your Postman Client

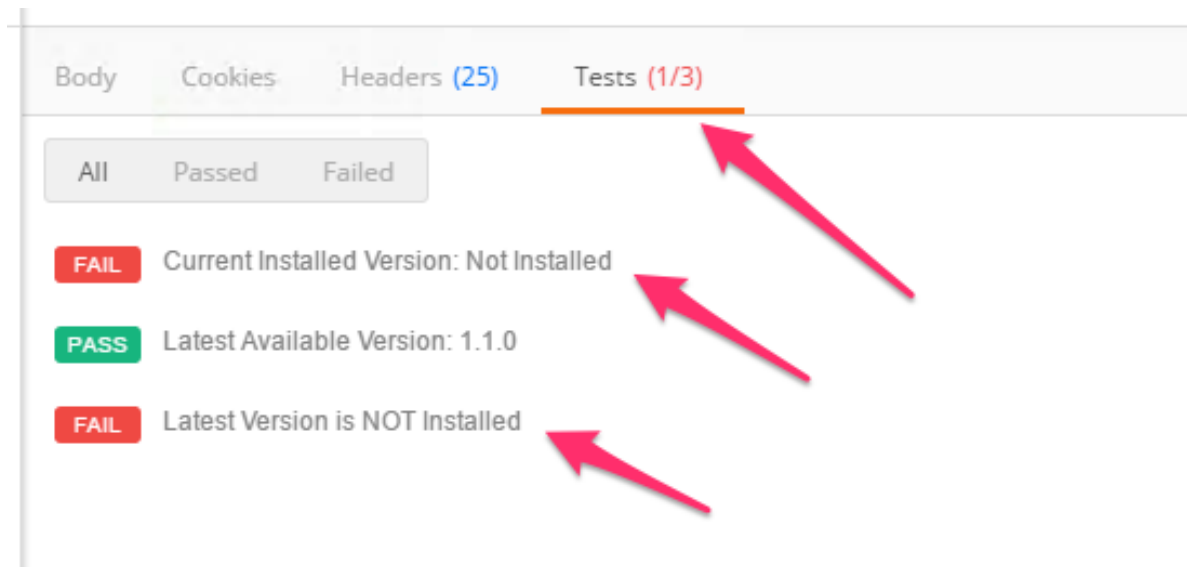
To utilize the helper functions the framework include we must install those functions into the Postman Client. The installation helpers perform the following tasks:

1. Determine the most current version of the framework
2. Dynamically minify the JavaScript code from the f5-postman-workflows GitHub repository using Google's Closure Compiler
3. Install the minified JS code into a Postman Global Variable
4. Set a number of Global variables that allow you to configure various options

To install the framework complete the following tasks:

1. Open the `F5_Postman_Workflows` collection
2. Open the `Install` folder
3. Select the `Check f5-postman-workflows Version` item and click 'Send'
4. Examine the 'Tests' portion of the **RESPONSE**:

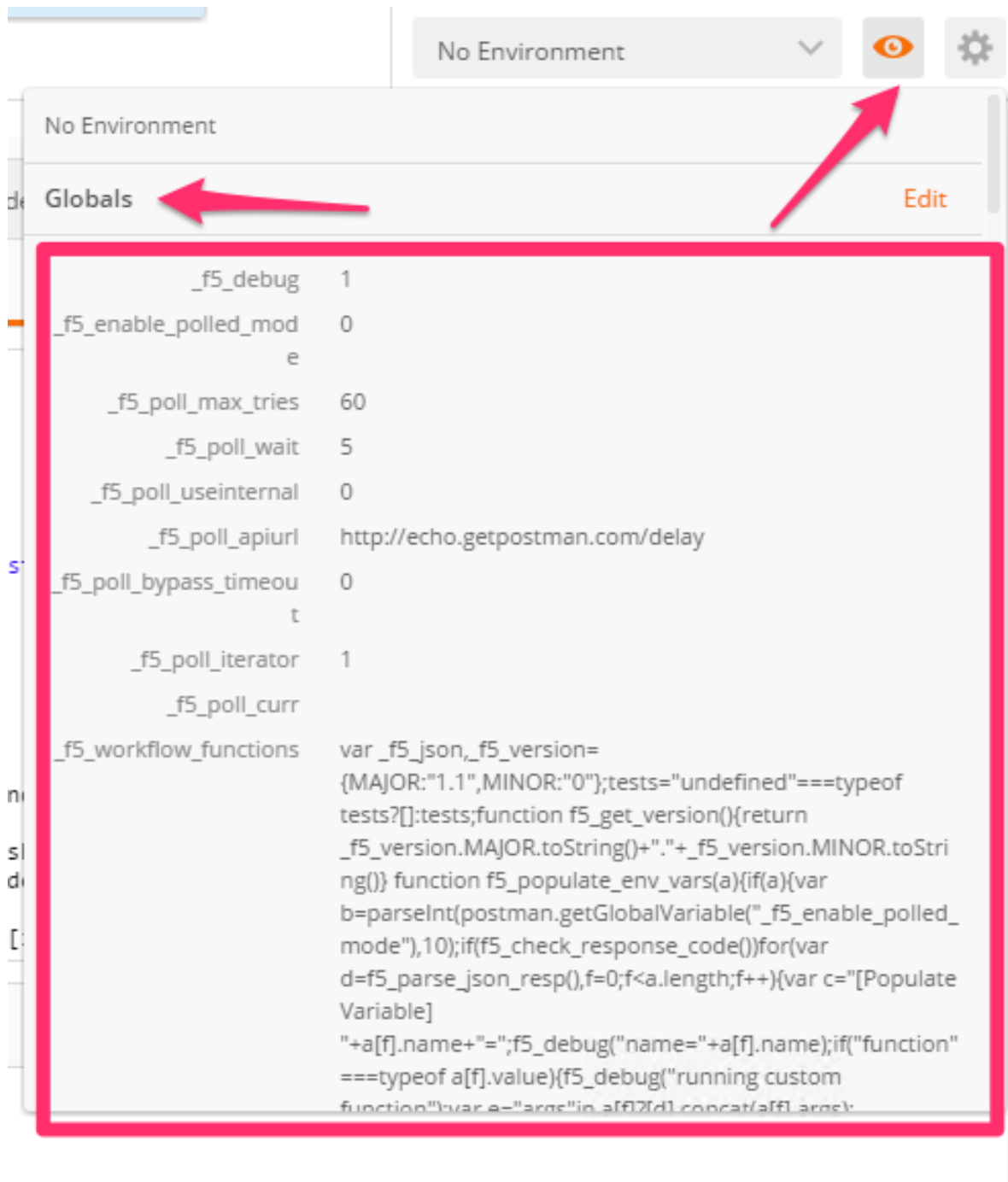




5. Select the `Install/Upgrade f5-postman-workflows` item and click 'Send'
6. Examine the 'Tests' again and ensure that Installation was successful:



7. Click the 'Eye' button in the top right of the Postman window and examine the Global variables that have been populated



The f5-postman-workflows framework is now installed in your Postman client.

#### 4.4.2 Lab 4.2 – Manually Execute a Workflow

In this lab we will walk through how to obtain two collections that use the f5-postman-workflows framework and execute a simple workflow using the Postman GUI client. The f5-postman-workflows GitHub repository is continually updated with new collections that can be used as is, or customized, to automate the F5 platform. Additionally the f5-super-netops-container automatically downloads these and other tools so users can rapidly execute workflows in their environments.

Postman collections also serve as a reference example of how various tasks can be accomplished using an **Imperative** process. When executing a collection you are actually providing a **Declarative** input to an **Imperative** process.

Collections are self-documenting and we will explore how to access the included documentation to assemble a workflow from start to end. In the next lab we will use this base knowledge to create workflows as JSON templates that can be executed by the f5-newman-wrapper on the f5-super-netops-container image (or any system that has Newman installed)

## Task 1 - Import and Explore BIG-IP Collections

We will import two collections to Postman using the same steps in the previous labs. The collections will allow us to perform REST API Authentication to BIG-IP devices and then execute Operational actions on the BIG-IP device.

Execute the following steps to complete this task:

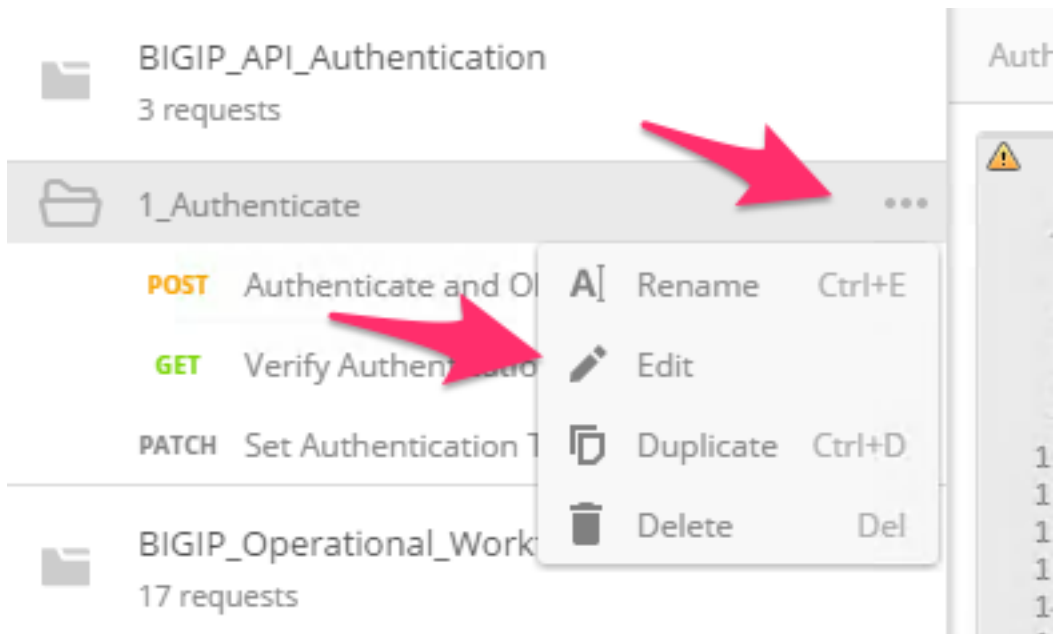
1. Click Import -> Import from Link and import these collection URLs:

- [https://raw.githubusercontent.com/0xHiteshPatel/f5-postman-workflows/master/collections/BIG\\_IP/BIGIP\\_API\\_Authentication.postman\\_collection.json](https://raw.githubusercontent.com/0xHiteshPatel/f5-postman-workflows/master/collections/BIG_IP/BIGIP_API_Authentication.postman_collection.json)
- [https://raw.githubusercontent.com/0xHiteshPatel/f5-postman-workflows/master/collections/BIG\\_IP/BIGIP\\_Operational\\_Workflows.postman\\_collection.json](https://raw.githubusercontent.com/0xHiteshPatel/f5-postman-workflows/master/collections/BIG_IP/BIGIP_Operational_Workflows.postman_collection.json)

2. You should now have two additional Collections in the sidebar:

- BIGIP\_API\_Authentication
- BIGIP\_Operational\_Workflows

3. Expand the BIGIP\_API\_Authentication collection. Within the collection you will see one folder named 1\_Authenticate. Folders are used to organize various workflows within a collection. In this case this collection performs exactly one task, authentication, therefore one folder is present.
4. Expand the 1\_Authenticate folder. Notice there are three requests in the folder. These three requests will be executed in a synchronous manner (one-after-another).
5. Click the ... icon on the 1\_Authenticate folder, then click Edit



6. In the following window you will see documentation explaining what the requests in this folder accomplish. Additionally you will see a series of Input and Output variables. Unless marked otherwise it is assumed that all Input variables are required. In this case the `bigip_token_timeout` variable is optional.

Folders may also contain output variables that are set to pass data between requests in different collections. In this case the output variable is named `bigip_token` and contains the authentication token that can be sent in the `X-F5-Auth-Token` HTTP header to perform authentication.

7. Close the window by clicking 'Cancel'
8. Repeat the steps above and explore the `BIGIP_Operational_Workflows` collection, specifically the `4A_Get_BIGIP_Version` folder.

## Task 2 - Manually Chain Folders into a Workflow

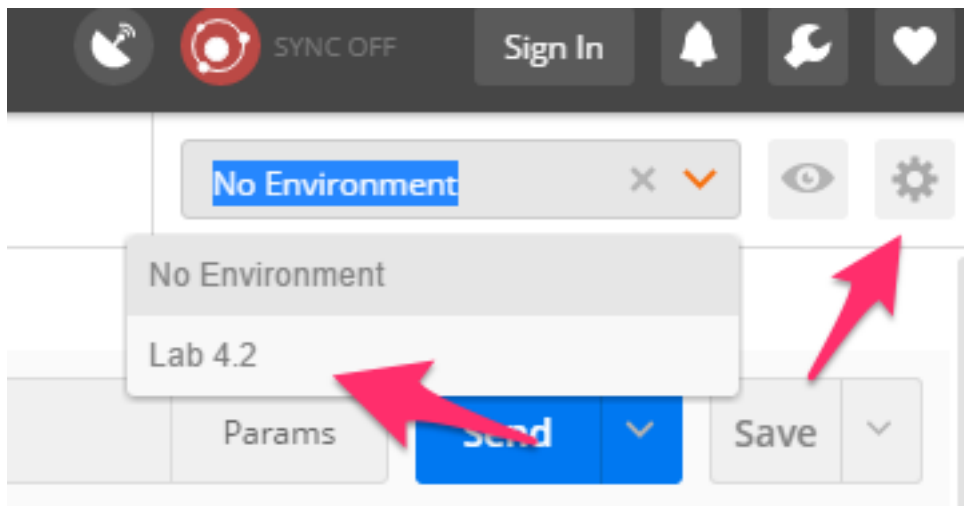
In this task we will explore how to chain two folders together and manually execute a workflow. This example is simple, but should help illustrate how we can use folders as building blocks that can be assembled or chained together to construct a workflow.

We will use the `1_Authenticate` folder in the `BIGIP_API_Authentication` collection and then pass the authentication token to the `4A_Get_BIGIP_Version` folder in the `BIGIP_Operational_Workflows` collection.

Execute the following steps to complete this task:

1. Create a new Postman environment by clicking the Gear icon -> Manage Environments -> Add.
2. Name the environment `Lab 4.2` and populate the following key/value pairs:
  - **`bigip_mgmt`**: 10.1.1.4
  - **`bigip_username`**: admin
  - **`bigip_password`**: admin
3. Click the 'Add' button, then close the 'Manage Environments' window.

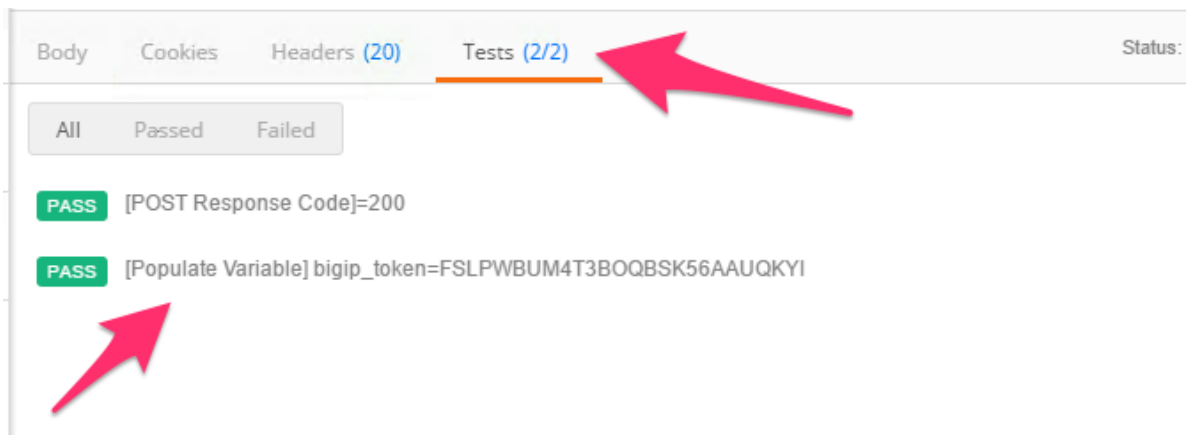
4. Select the Lab 4.2 environment:



The preceding steps configured the Input Variables required for all the folders that comprise our workflow. We will now manually execute all the requests in the folders.

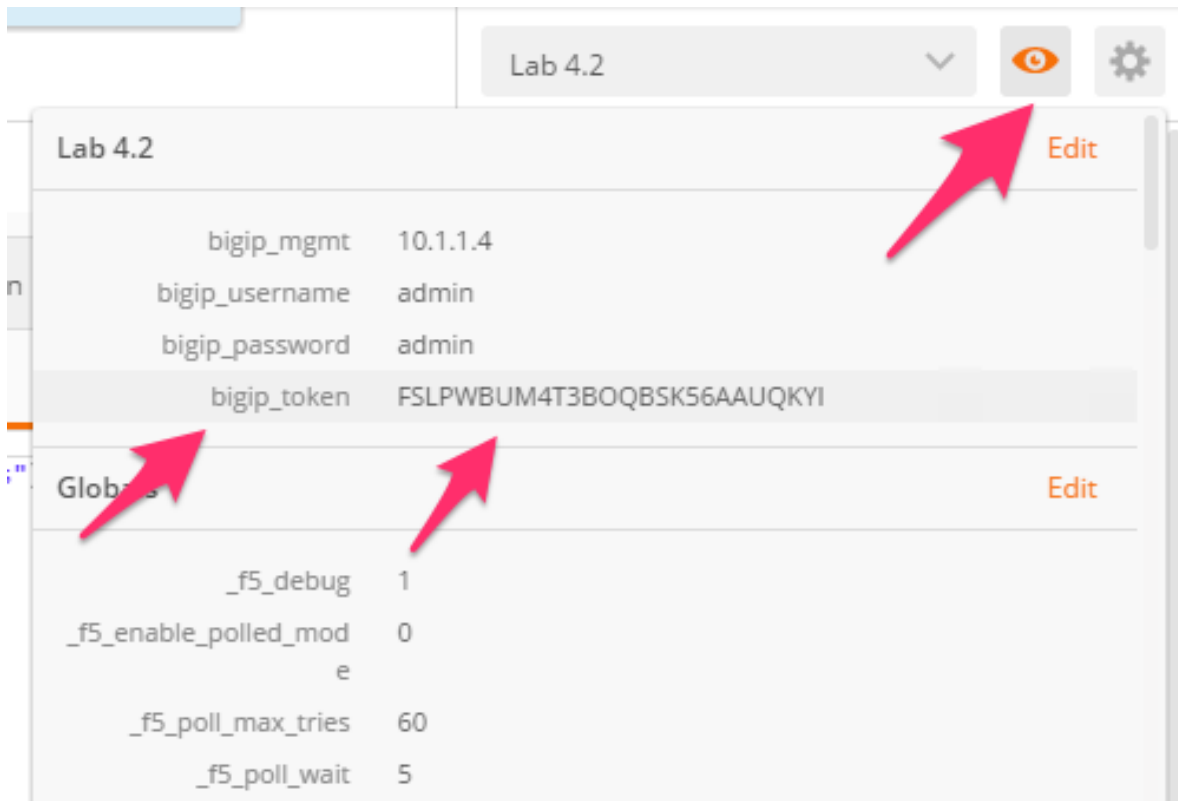
1. Expand the BIGIP\_API\_Authentication -> 1\_Authenticate folder.
2. Select the Authenticate and Obtain Token item and click Send
3. Examine the Tests in the response portion of the request. All the tests should be passing. Additionally you should see a test similar to:

[Populate Variable] bigip\_token=....



These Tests items and there corresponding actions (populating a variable in this case) are generated by the f5-postman-workflows framework.

4. Examine your Postman Environment variables and confirm that the bigip\_token variable is present and populated.



5. Select the `Verify Authentication Works` request in the folder and click 'Send'. Examine the Tests and ensure they are all passing
6. Select the `Set Authentication Token Timeout` request, click *Send* and verify all Tests pass.

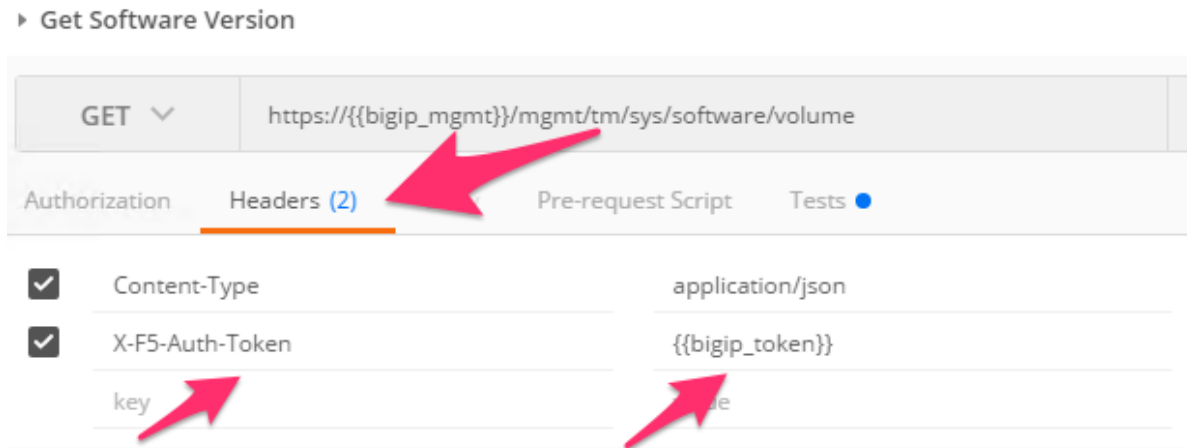
At this point we have successfully authenticated to our device and stored the authentication token in the `bigip_token` environment variable. We will now execute a request in a different collection and folder that uses the `bigip_token` variable value to authenticate and perform its actions.

1. Expand the `BIGIP_Operational_Workflows` -> `4A_Get_BIGIP_Version` folder.
2. Click the `Get Software Version` request.
3. Click the 'Headers' tab. Notice that the value for the `X-F5-Auth-Token` header is populated with the `bigip_token` variable value.

---

**Note:** Postman uses the `{{variable_name}}` syntax to perform variable value substitution.

---



4. Click 'Send' to send the request. Examine the Tests and ensure all tests have passed.
5. Examine your environment variables and note that the `bigip_version` and `bigip_build` variables are now populated.

While the example above was simple it should show how we can chain together different collections and folders to assemble custom workflows. The key concepts to understand are:

- The `f5-postman-workflows` framework and collection test code perform unit tests on the response data and verify the request executed successfully.
- The framework also populates Output Variables as documented so they can be used in subsequent requests as Inputs to assemble a workflow

Next, we will explore how to use this base knowledge to assemble various collections and folders into workflows using Newman and the `f5-newman-wrapper`.

#### 4.4.3 Lab 4.3 – `f5-newman-wrapper` Introduction

As shown in the previous lab we can manually execute collections and folders using the Postman GUI to achieve some end result on BIG-IP devices. While this capability is important in a test/prototyping phase we need to ensure we can execute these manual steps as an automated process.

To achieve this goal we can use the `f5-newman-wrapper` tool. This tool allows a user to specify a workflow in a JSON formatted file that includes Input Variables, the collections and folders to execute and various output options to provide feedback and run details in a programmatic fashion.

The core element of a workflow that can be executed by `f5-newman-wrapper` is a JSON formatted input file. In this lab we will introduce the file format.

##### Task 1 - Explore the workflow JSON format

To introduce the format of the workflow file we will use an example that recreates the simple workflow we executed manually in the previous lab. We will explore the file in sections followed by showing the whole file.

## Define Name and Description

```
1 {
2   "name": "Wrapper_Demo_1",
3   "description": "Execute a chained workflow that authenticates to a BIG-IP and_
↪retrieves it's software version"
4 }
```

## Define Global Settings for the Run

This section defines how f5-newman-wrapper will run this workflow. The attributes are explained in the table below.

```
1 {
2   "globalEnvVars": "../framework/f5-postman-workflows.postman_globals.json",
3   "globalOptions": {
4     "insecure": true,
5     "reporters": ["cli"]
6   },
7   "saveEnvVars": true,
8   "outputFile": "Wrapper_Demo_1-run.json",
9   "envOutputFile": "Wrapper_Demo_1-env.json"
10 }
```

Attribute	Description
globalEnvVars	This is the file that contains the Global environment variables used by Newman. This file is generated by the f5-postman-workflows build scripts and contains the same global variables as we saw in the previous lab that installed the framework into the Postman GUI client
globalOptions	Specify the global options for newman. Available options are documented at: <a href="https://github.com/postmanlabs/newman#api-reference">https://github.com/postmanlabs/newman#api-reference</a>  <b>Note:</b> Removing the <code>cli</code> option from the <code>reporters</code> array will disable verbose CLI output
saveEnvVars	Save the environment variables at the end of the run to a file
outputFile	The file to save the run details to.
envOutputFile	The file to save the environment variables at the end of the run to.

## Define Input Variables

This section specifies the Input Variables for the workflow. The name `globalVars` conveys that the variables defined here will be present for each request in the defined workflow (the global scope from a workflow perspective). Variables can also be defined within each item in a workflow (the local scope from a item perspective). In the case of a global and local variable that is named identically, the local scope variable will take precedence.

```
1 {
2   "globalVars": {
3     "bigip_mgmt": "10.1.1.4",
```



```

4      "bigip_username": "admin",
5      "bigip_password": "admin"
6  }
7  }

```

## Define the Workflow Collections and Ordering

This section defines the workflow and collections and folders that it is comprised of. The `workflow` attribute is an ordered array that contains objects defining each collection and folder to run.

```

1  {
2      "workflow": [
3          {
4              "name": "Authenticate to BIG-IP",
5              "options": {
6                  "collection": "../collections/BIG_IP/BIGIP_API_Authentication.postman_
↵collection.json",
7                  "folder": "1_Authenticate"
8              }
9          },
10         {
11             "name": "Get BIG-IP Software Version",
12             "options": {
13                 "collection": "../collections/BIG_IP/BIGIP_Operational_Workflows.
↵postman_collection.json",
14                 "folder": "4A_Get_BIGIP_Version"
15             }
16         }
17     ]
18 }

```

Lets look at the item in the workflow that performs authentication:

```

1      {
2          "name": "Authenticate to BIG-IP",
3          "options": {
4              "collection": "../collections/BIG_IP/BIGIP_API_
↵Authentication.postman_collection.json",
5              "folder": "1_Authenticate"
6          }
7      }

```

The `name` attribute specifies the name for this item. The `options` object specifies the parameters used to execute this particular item. In our case the `collection` attribute refers to the file containing the `BIGIP_API_Authentication` collection. The `folder` attribute specifies the name of the folder to run in the collection.

By default all output variables from a collection or folder are passed to the next item in the workflow. This allows us to chain collections together as needed to build workflows.

## Final Workflow JSON

```

1  {
2      "name": "Wrapper_Demo_1",

```

```

3      "description": "Execute a chained workflow that authenticates to a BIG-IP
↪and retrieves it's software version",
4      "globalEnvVars": "../framework/f5-postman-workflows.postman_globals.json",
5      "globalOptions": {
6          "insecure": true,
7          "reporters": ["cli"]
8      },
9      "globalVars": {
10         "bigip_mgmt": "10.1.1.4",
11         "bigip_username": "admin",
12         "bigip_password": "admin"
13     },
14     "saveEnvVars": true,
15     "outputFile": "Wrapper_Demo_1-run.json",
16     "envOutputFile": "Wrapper_Demo_1-env.json",
17     "workflow": [
18         {
19             "name": "Authenticate to BIG-IP",
20             "options": {
21                 "collection": "../collections/BIG_IP/BIGIP_API_
↪Authentication. postman_collection.json",
22                 "folder": "1_Authenticate"
23             }
24         },
25         {
26             "name": "Get BIG-IP Software Version",
27             "skip": false,
28             "options": {
29                 "collection": "../collections/BIG_IP/BIGIP_
↪Operational_Workflows. postman_collection.json",
30                 "folder": "4A_Get_BIGIP_Version"
31             }
32         }
33     ]
34 }

```

#### 4.4.4 Lab 4.4 – Run a workflow with f5-newman-wrapper

In this lab we will use the f5-super-netops-container to run the workflow we reviewed in the previous lab. The advantage of using the f5-super-netops Container is that all the tools, collections and frameworks are pre-installed and ready to use.

##### Task 1 - Run a f5-newman-wrapper Workflow

1. Open an SSH session as described in the *previous lab*
2. Run `cd f5-postman-workflows/local`
3. Run `cp ../workflows/Wrapper_Demo_1.json .`
4. Edit the `Wrapper_Demo_1.json` file with `vim` and enter the `10.1.1.4` for the value of the `bigip_mgmt` variable

```

"globalVars": {
    "bigip_mgmt": "10.1.1.4",
    "bigip_username": "admin",

```

```
    "bigip_password": "admin"
  },
```

5. Run `f5-newman-wrapper Wrapper_Demo_1.json`
6. Examine the output to see how the workflow was executed. Notice that the same tests that we saw when using Postman are present during this run.

Example output:

```
[snops@f5-super-netops] [~/f5-postman-workflows/local] $ f5-newman-wrapper
↳ Wrapper_Demo_1.json
[Wrapper_Demo_1-2017-03-30-04-08-12] starting run
[Wrapper_Demo_1-2017-03-30-04-08-12] [runCollection][Authenticate to BIG-IP]
↳ running...
newman

BIGIP_API_Authentication

? 1_Authenticate
? Authenticate and Obtain Token
  POST https://10.1.1.4/mgmt/shared/authn/login [200 OK, 1.41KB, 108ms]
  ✓ [POST Response Code]=200
  ✓ [Populate Variable] bigip_token=WYKIVPHCNASNVEC55ZDVNH5002

? Verify Authentication Works
  GET https://10.1.1.4/mgmt/shared/authz/tokens/WYKIVPHCNASNVEC55ZDVNH5002 [200
↳ OK, 1.23KB, 8ms]
  ✓ [GET Response Code]=200
  ✓ [Current Value] token=WYKIVPHCNASNVEC55ZDVNH5002
  ✓ [Check Value] token == WYKIVPHCNASNVEC55ZDVNH5002

? Set Authentication Token Timeout
  PATCH https://10.1.1.4/mgmt/shared/authz/tokens/WYKIVPHCNASNVEC55ZDVNH5002 [200
↳ OK, 1.23KB, 14ms]
  ✓ [PATCH Response Code]=200
  ✓ [Current Value] timeout=1200
  ✓ [Check Value] timeout == 1200

?-----?-----?-----?
|                                     | executed |   failed |
?-----?-----?-----?
|               iterations |         1 |         0 |
?-----?-----?-----?
|               requests  |         3 |         0 |
?-----?-----?-----?
|             test-scripts |         3 |         0 |
?-----?-----?-----?
|    prerequest-scripts   |         1 |         0 |
?-----?-----?-----?
|             assertions   |         8 |         0 |
?-----?-----?-----?
| total run duration: 297ms |         |
?-----?-----?-----?
| total data received: 1.71KB (approx) |         |
?-----?-----?-----?
| average response time: 43ms |         |
?-----?-----?-----?
[Wrapper_Demo_1-2017-03-30-04-08-12] [runCollection][Get BIG-IP Software Version]
↳ running...
```

```

newman

BIGIP_Operational_Workflows

? 4A_Get_BIGIP_Version
? Get Software Version
  GET https://10.1.1.4/mgmt/tm/sys/software/volume [200 OK, 1.32KB, 16ms]
  ✓ [GET Response Code]=200
  ✓ [Populate Variable] bigip_version=12.1.1
  ✓ [Populate Variable] bigip_build=1.0.196
[Wrapper_Demo_1-2017-03-30-04-08-12] run completed

?-----?-----?-----?
|                                     | executed |    failed |
?-----?-----?-----?
|             iterations |          1 |          0 |
?-----?-----?-----?
|             requests  |          1 |          0 |
?-----?-----?-----?
|           test-scripts |          1 |          0 |
?-----?-----?-----?
|    prerequisite-scripts |          0 |          0 |
?-----?-----?-----?
|             assertions |          3 |          0 |
?-----?-----?-----?
| total run duration: 58ms |          |
?-----?-----?-----?
| total data received: 611B (approx) |          |
?-----?-----?-----?
| average response time: 16ms |          |
?-----?-----?-----?

```

7. Examine the environment variables that were saved at the end of the run by executing `cat Wrapper_Demo_1-env.json`

Example output:

```

1 {
2   "id": "c0550892-36d4-4412-bf35-ald9aa8d2efe",
3   "values": [
4     {
5       "type": "any",
6       "value": "10.1.1.4",
7       "key": "bigip_mgmt"
8     },
9     {
10      "type": "any",
11      "value": "admin",
12      "key": "bigip_username"
13    },
14    {
15      "type": "any",
16      "value": "admin",
17      "key": "bigip_password"
18    },
19    {
20      "type": "any",
21      "value": "WYKIVPHCNASNVEC55ZDVNH5002",

```

```

22     "key": "bigip_token"
23   },
24   {
25     "type": "any",
26     "value": "1200",
27     "key": "bigip_token_timeout"
28   },
29   {
30     "type": "any",
31     "value": "12.1.1",
32     "key": "bigip_version"
33   },
34   {
35     "type": "any",
36     "value": "1.0.196",
37     "key": "bigip_build"
38   }
39 ]
40 }

```

Notice that the `bigip_version` and `bigip_build` variables were saved. This file is JSON formatted and can easily be used directly by other tools to drive further automation.

## 4.4.5 Lab 4.5 – Building Complex Workflows

In the previous lab we reviewed and ran a very simple workflow. To support more complex use cases `f5-newman-wrapper` includes features to help build more complex workflows.

These features allow users to:

- Create infinitely nested items
- Rename/remap variables name pre and post run of an item
- Load variables from a saved environment file
- Define variables in the global (workflow) or local (item) scope

To explore all the available options currently implemented please refer to <https://raw.githubusercontent.com/0xHiteshPatel/f5-postman-workflows/master/framework/f5-newman-wrapper/workflow-schema.json>

### Task 1 - Explore Nested Workflows & Variable Remapping

By using the 'children' array within an item in a workflow you can create nested items. In this task, we will create a more advanced version of the workflow we used in the previous lab. This workflow will perform authentication to two BIG-IP devices and then retrieve the software version running on each.

We will implement a workflow that is best depicted by the following branch diagram:

```

Start
|
|- Authenticate
|   |- Authenticate to BIG-IP A
|   |- Authenticate to BIG-IP B
|
|- Get BIGIP Version
|   |- Get BIGIP Version on BIG-IP A
|   |- Get BIGIP Version on BIG-IP B

```

```
|
Stop
```

To implement this workflow we need to consider how Input Variables are passed to each item in the workflow. Previously, we saw that the following variables are required to the the `1_Authenticate` folder in the `BIGIP_API_Authentication` collection:

- `bigip_mgmt`
- `bigip_username`
- `bigip_password`

The issue we encounter when building this workflow is that we, at a minimum, have different values for `bigip_mgmt` because we are trying to communicate with two BIG-IP devices. To address this issue, we could define our input variables as follows:

- `bigip_a_mgmt = 10.1.1.4`
- `bigip_b_mgmt = 10.1.1.5`
- `bigip_username = admin`
- `bigip_password = admin`

This solves the problem of providing both BIG-IP management addresses, however, it introduces another issue. The `1_Authenticate` folder requires that the management IP address be passed in the `bigip_mgmt` input variable. To solve this issue, we will use variable name remapping to remap a globalVar to a different name before the `1_Authenticate` folder is run for each BIG-IP device. To illustrate this, we will add more information to our diagram:

```
Start
|
|- Define globalVars
|   |- bigip_a_mgmt = 10.1.1.4
|   |- bigip_b_mgmt = 10.1.1.5
|   |- bigip_username = admin
|   |- bigip_password = admin
|
|- Authenticate
|   |- Authenticate to BIG-IP A
|   |   | Pre-run: Remap bigip_a_mgmt -> bigip_mgmt
|   |   | Run: 1_Authenticate folder
|   |
|   |- Authenticate to BIG-IP B
|   |   | Pre-run: Remap bigip_b_mgmt -> bigip_mgmt
|   |   | Run: 1_Authenticate folder
|   |
|- Get BIGIP Version
|   |- Get BIGIP Version on BIG-IP A
|   |- Get BIGIP Version on BIG-IP B
|
Stop
```

We've now addressed our issues regarding defining and passing the BIG-IP management address, but have to consider one last problem. The **output variable** of the `1_Authenticate` folder is `bigip_token`. By default `f5-newman-wrapper` will store all output variables from one folder and automatically pass them to the next item. In this case, an issue occurs because the `Authenticate to BIG-IP B` item will overwrite the `bigip_token` variable that was outputted by the `Authenticate to BIG-IP A` item. To resolve this

issue, we can remap variables **AFTER** or post-run of an item. We can modify our diagram to handle this issue like this:

```
Start
|
|- Define globalVars
|   |- bigip_a_mgmt = 10.1.1.4
|   |- bigip_b_mgmt = 10.1.1.5
|   |- bigip_username = admin
|   |- bigip_password = admin
|
|- Authenticate
|   |- Authenticate to BIG-IP A
|   |   | Pre-run: Remap bigip_a_mgmt -> bigip_mgmt
|   |   | Run: 1_Authenticate folder
|   |   | Post-run: Remap bigip_token -> bigip_a_token
|   |
|   |- Authenticate to BIG-IP B
|   |   | Pre-run: Remap bigip_b_mgmt -> bigip_mgmt
|   |   | Run: 1_Authenticate folder
|   |   | Post-run: Remap bigip_token -> bigip_b_token
|   |
|- Get BIGIP Version
|   |- Get BIGIP Version on BIG-IP A
|   |- Get BIGIP Version on BIG-IP B
|
Stop
```

The last step is to perform some additional pre-run remapping to pass the correct token to the 4A\_Get\_BIGIP\_Version folder to get our BIG-IP software version. Additionally, we will perform some post-run remaps so we can save the output variables for each device:

```
Start
|
|- Define globalVars
|   |- bigip_a_mgmt = 10.1.1.4
|   |- bigip_b_mgmt = 10.1.1.5
|   |- bigip_username = admin
|   |- bigip_password = admin
|
|- Authenticate
|   |- Authenticate to BIG-IP A
|   |   | Pre-run: Remap bigip_a_mgmt -> bigip_mgmt
|   |   | Run: 1_Authenticate folder
|   |   | Post-run: Remap bigip_token -> bigip_a_token
|   |
|   |- Authenticate to BIG-IP B
|   |   | Pre-run: Remap bigip_b_mgmt -> bigip_mgmt
|   |   | Run: 1_Authenticate folder
|   |   | Post-run: Remap bigip_token -> bigip_b_token
|   |
|- Get BIGIP Version
|   |- Get BIGIP Version on BIG-IP A
|   |   | Pre-run: Remap bigip_a_mgmt -> bigip_mgmt
|   |   | Pre-run: Remap bigip_a_token -> bigip_token
|   |   | Run: 4A_Get_BIGIP_Version folder
|   |   | Post-run: Remap bigip_version -> bigip_a_version
|   |   | Post-run: Remap bigip_build -> bigip_a_build
```

```

| |
| |- Get BIGIP Version on BIG-IP B
| | | Pre-run: Remap bigip_b_mgmt -> bigip_mgmt
| | | Pre-run: Remap bigip_b_token -> bigip_token
| | | Run: 4A_Get_BIGIP_Version folder
| | | Post-run: Remap bigip_version -> bigip_b_version
| | | Post-run: Remap bigip_build -> bigip_b_build
|
| |- Save globalVars to file
|
Stop

```

---

**Note:** Collections and folders that are designed to act on multiple devices are designed to automatically use the `bigip_a...` and `bigip_b...` syntax to avoid having to remap variables. In this case the `BIGIP_Operational_Workflows` collection is designed to perform actions on **one** device at a time, thus the need for remapping of the `bigip_token` input variables.

---



---

**Note:** Another option that is available to solve this issue is to define all variables in the local scope for each item. This method is not preferred because it decreases portability and increases complexity in definition of input variables.

---

## Task 2 - Build Complex Workflow JSON

### Define Global Settings & Variables:

```

1 {
2   "name": "Wrapper_Demo_2",
3   "description": "Execute a chained workflow that authenticates to two BIG-IPs and
↪ retrieves their software version",
4   "globalEnvVars": "../framework/f5-postman-workflows.postman_globals.json",
5   "globalOptions": {
6     "insecure": true,
7     "reporters": ["cli"]
8   },
9   "globalVars": {
10    "bigip_a_mgmt": "10.1.1.4",
11    "bigip_b_mgmt": "10.1.1.5",
12    "bigip_username": "admin",
13    "bigip_password": "admin"
14  },
15  "saveEnvVars": true,
16  "outputFile": "Wrapper_Demo_2-run.json",
17  "envOutputFile": "Wrapper_Demo_2-env.json"
18 }

```

### Define Authentication Items

---

**Note:** As shown below, we can use the `skip: true` attribute to signal `f5-newman-wrapper` to not run that particular item. The items children will still be processed. The `skip` attribute can be used to create

---



a container for similar requests.

```
1 {
2   "workflow": [
3     {
4       "name": "Authenticate to BIG-IPs",
5       "skip": true,
6       "children": [
7         {
8           "name": "Authenticate to BIG-IP A",
9           "options": {
10            "collection": "../collections/BIG_IP/BIGIP_API_Authentication.postman_
↵collection.json",
11            "remapPreRun": {
12              "bigip_a_mgmt": "bigip_mgmt"
13            },
14            "folder": "1_Authenticate",
15            "remapPostRun": {
16              "bigip_token": "bigip_a_token"
17            }
18          }
19        },
20        {
21          "name": "Authenticate to BIG-IP B",
22          "options": {
23            "collection": "../collections/BIG_IP/BIGIP_API_Authentication.postman_
↵collection.json",
24            "remapPreRun": {
25              "bigip_b_mgmt": "bigip_mgmt"
26            },
27            "folder": "1_Authenticate",
28            "remapPostRun": {
29              "bigip_token": "bigip_b_token"
30            }
31          }
32        }
33      ]
34    }
35  ]
36 }
```

The JSON above implements the following part of our branch diagram:

```
| - Authenticate
|   | - Authenticate to BIG-IP A
|   |   | Pre-run: Remap bigip_a_mgmt -> bigip_mgmt
|   |   |   Run: 1_Authenticate folder
|   |   | Post-run: Remap bigip_token -> bigip_a_token
|   |
|   | - Run: Authenticate to BIG-IP B
|   |   | Pre-run: Remap bigip_b_mgmt -> bigip_mgmt
|   |   |   Run: 1_Authenticate folder
|   |   | Post-run: Remap bigip_token -> bigip_b_token
```

Specifically, note the use of the `skip` attribute on line 5 to create a container to group the items together.

## Define Get Software Version Items

```
1 {
2   "workflow": [
3     {
4       "name": "Get BIG-IP Software Versions",
5       "skip": true,
6       "children": [
7         {
8           "name": "Get BIG-IP A Software Version",
9           "options": {
10            "collection": "../collections/BIG_IP/BIGIP_Operational_Workflows.postman_
↪collection.json",
11            "remapPreRun": {
12              "bigip_a_mgmt": "bigip_mgmt",
13              "bigip_a_token": "bigip_token"
14            },
15            "folder": "4A_Get_BIGIP_Version",
16            "remapPostRun": {
17              "bigip_version": "bigip_a_version",
18              "bigip_build": "bigip_a_build"
19            }
20          }
21        },
22        {
23          "name": "Get BIG-IP B Software Version",
24          "options": {
25            "collection": "../collections/BIG_IP/BIGIP_Operational_Workflows.postman_
↪collection.json",
26            "remapPreRun": {
27              "bigip_b_mgmt": "bigip_mgmt",
28              "bigip_b_token": "bigip_token"
29            },
30            "folder": "4A_Get_BIGIP_Version",
31            "remapPostRun": {
32              "bigip_version": "bigip_b_version",
33              "bigip_build": "bigip_b_build"
34            }
35          }
36        }
37      ]
38    }
39  ]
40 }
```

The JSON above implements the following part of our branch diagram:

```
| - Get BIGIP Version
|   | - Get BIGIP Version on BIG-IP A
|   |   | Pre-run: Remap bigip_a_mgmt -> bigip_mgmt
|   |   | Pre-run: Remap bigip_a_token -> bigip_token
|   |   |   Run: 4A_Get_BIGIP_Version folder
|   |   | Post-run: Remap bigip_version -> bigip_a_version
|   |   | Post-run: Remap bigip_build -> bigip_a_build
|   |
|   | - Get BIGIP Version on BIG-IP B
|   |   | Pre-run: Remap bigip_b_mgmt -> bigip_mgmt
|   |   | Pre-run: Remap bigip_b_token -> bigip_token
```

```
| Run: 4A_Get_BIGIP_Version folder
| Post-run: Remap bigip_version -> bigip_b_version
| Post-run: Remap bigip_build -> bigip_b_build
```

## Final Workflow JSON

```
{
  "name": "Wrapper_Demo_2",
  "description": "Execute a chained workflow that authenticates to two BIG-IPs and
↳ retrieves their software version",
  "globalEnvVars": "../framework/f5-postman-workflows.postman_globals.json",
  "globalOptions": {
    "insecure": true,
    "reporters": ["cli"]
  },
  "globalVars": {
    "bigip_a_mgmt": "",
    "bigip_b_mgmt": "",
    "bigip_username": "admin",
    "bigip_password": "admin"
  },
  "saveEnvVars": true,
  "outputFile": "Wrapper_Demo_2-run.json",
  "envOutputFile": "Wrapper_Demo_2-env.json",
  "workflow": [
    {
      "name": "Authenticate to BIG-IPs",
      "skip": true,
      "children": [
        {
          "name": "Authenticate to BIG-IP A",
          "options": {
            "collection": "../collections/BIG_IP/BIGIP_API_Authentication.postman_
↳ collection.json",
            "remapPreRun": {
              "bigip_a_mgmt": "bigip_mgmt"
            },
            "folder": "1_Authenticate",
            "remapPostRun": {
              "bigip_token": "bigip_a_token"
            }
          }
        },
        {
          "name": "Authenticate to BIG-IP B",
          "options": {
            "collection": "../collections/BIG_IP/BIGIP_API_Authentication.postman_
↳ collection.json",
            "remapPreRun": {
              "bigip_b_mgmt": "bigip_mgmt"
            },
            "folder": "1_Authenticate",
            "remapPostRun": {
              "bigip_token": "bigip_b_token"
            }
          }
        }
      ]
    }
  ]
}
```

```

48     }
49   ]
50 },
51 {
52   "name": "Get BIG-IP Software Versions",
53   "skip": true,
54   "children": [
55     {
56       "name": "Get BIG-IP A Software Version",
57       "options": {
58         "collection": "../collections/BIG_IP/BIGIP_Operational_Workflows.postman_
↪collection.json",
59         "remapPreRun": {
60           "bigip_a_mgmt": "bigip_mgmt",
61           "bigip_a_token": "bigip_token"
62         },
63         "folder": "4A_Get_BIGIP_Version",
64         "remapPostRun": {
65           "bigip_version": "bigip_a_version",
66           "bigip_build": "bigip_a_build"
67         }
68       }
69     },
70     {
71       "name": "Get BIG-IP B Software Version",
72       "options": {
73         "collection": "../collections/BIG_IP/BIGIP_Operational_Workflows.postman_
↪collection.json",
74         "remapPreRun": {
75           "bigip_b_mgmt": "bigip_mgmt",
76           "bigip_b_token": "bigip_token"
77         },
78         "folder": "4A_Get_BIGIP_Version",
79         "remapPostRun": {
80           "bigip_version": "bigip_b_version",
81           "bigip_build": "bigip_b_build"
82         }
83       }
84     }
85   ]
86 }
87 ]
88 }

```

### Task 3 - Run the Workflow

1. Open an SSH session as described in the *previous lab*
2. Run `cd f5-postman-workflows/local`
3. Run `cp ../workflows/Wrapper_Demo_2.json .`
4. Edit the `Wrapper_Demo_2.json` file and enter you BIG-IP managment addresses

```

1 {
2   "globalVars": {
3     "bigip_a_mgmt": "10.1.1.4",
4     "bigip_b_mgmt": "10.1.1.5",

```

```

5     "bigip_username": "admin",
6     "bigip_password": "admin"
7   }
8 }

```

5. Run `f5-newman-wrapper Wrapper_Demo_2.json`

6. Examine the output to see how the workflow was executed.

Example output:

```

[snops@f5-super-netops] [~/f5-postman-workflows/local] $ f5-newman-wrapper_
↳ Wrapper_Demo_2.json
[Wrapper_Demo_2-2017-03-30-19-22-52] starting run
[Wrapper_Demo_2-2017-03-30-19-22-52] [runCollection][Authenticate to BIG-IP A]_
↳ running...
newman

BIGIP_API_Authentication

? 1_Authenticate
? Authenticate and Obtain Token
  POST https://10.1.1.4/mgmt/shared/authn/login [200 OK, 1.41KB, 570ms]
  ✓ [POST Response Code]=200
  ✓ [Populate Variable] bigip_token=UE7W5CXWM5SJ6SZE5A7KTAI5Q

? Verify Authentication Works
  GET https://10.1.1.4/mgmt/shared/authz/tokens/UE7W5CXWM5SJ6SZE5A7KTAI5Q [200_
↳ OK, 1.23KB, 9ms]
  ✓ [GET Response Code]=200
  ✓ [Current Value] token=UE7W5CXWM5SJ6SZE5A7KTAI5Q
  ✓ [Check Value] token == UE7W5CXWM5SJ6SZE5A7KTAI5Q

? Set Authentication Token Timeout
  PATCH https://10.1.1.4/mgmt/shared/authz/tokens/UE7W5CXWM5SJ6SZE5A7KTAI5Q [200_
↳ OK, 1.23KB, 13ms]
  ✓ [PATCH Response Code]=200
  ✓ [Current Value] timeout=1200
  ✓ [Check Value] timeout == 1200

?-----?-----?-----?
|               | executed |   failed |
?-----?-----?-----?
|       iterations |         1 |         0 |
?-----?-----?-----?
|       requests  |         3 |         0 |
?-----?-----?-----?
|    test-scripts |         3 |         0 |
?-----?-----?-----?
|prerequisite-scripts |         1 |         0 |
?-----?-----?-----?
|       assertions |         8 |         0 |
?-----?-----?-----?
| total run duration: 740ms
?-----?-----?-----?
| total data received: 1.71KB (approx)
?-----?-----?-----?
| average response time: 197ms
?-----?-----?-----?

```

```
[Wrapper_Demo_2-2017-03-30-19-22-52] [runCollection][Authenticate to BIG-IP B]
↪running...
newman

BIGIP_API_Authentication

? 1_Authenticate
? Authenticate and Obtain Token
  POST https://10.1.1.5/mgmt/shared/authn/login [200 OK, 1.41KB, 350ms]
  ✓ [POST Response Code]=200
  ✓ [Populate Variable] bigip_token=ONQXOQPNCVOHZELKIFSPHETL3I

? Verify Authentication Works
  GET https://10.1.1.5/mgmt/shared/authz/tokens/ONQXOQPNCVOHZELKIFSPHETL3I [200
↪OK, 1.23KB, 9ms]
  ✓ [GET Response Code]=200
  ✓ [Current Value] token=ONQXOQPNCVOHZELKIFSPHETL3I
  ✓ [Check Value] token == ONQXOQPNCVOHZELKIFSPHETL3I

? Set Authentication Token Timeout
  PATCH https://10.1.1.5/mgmt/shared/authz/tokens/ONQXOQPNCVOHZELKIFSPHETL3I [200
↪OK, 1.23KB, 12ms]
  ✓ [PATCH Response Code]=200
  ✓ [Current Value] timeout=1200
  ✓ [Check Value] timeout == 1200

?-----?-----?-----?
|                               | executed |   failed |
?-----?-----?-----?
|           iterations |         1 |         0 |
?-----?-----?-----?
|           requests  |         3 |         0 |
?-----?-----?-----?
|         test-scripts |         3 |         0 |
?-----?-----?-----?
|    prerequest-scripts |         1 |         0 |
?-----?-----?-----?
|           assertions |         8 |         0 |
?-----?-----?-----?
| total run duration: 472ms |         |
?-----?-----?-----?
| total data received: 1.71KB (approx) |         |
?-----?-----?-----?
| average response time: 123ms |         |
?-----?-----?-----?

[Wrapper_Demo_2-2017-03-30-19-22-52] [runCollection][Get BIG-IP A Software
↪Version] running...
newman

BIGIP_Operational_Workflows

? 4A_Get_BIGIP_Version
? Get Software Version
  GET https://10.1.1.4/mgmt/tm/sys/software/volume [200 OK, 1.32KB, 207ms]
  ✓ [GET Response Code]=200
  ✓ [Populate Variable] bigip_version=12.1.1
  ✓ [Populate Variable] bigip_build=1.0.196
```

```

?-----?-----?-----?
|                               | executed |   failed |
?-----?-----?-----?
|             iterations |         1 |         0 |
?-----?-----?-----?
|             requests  |         1 |         0 |
?-----?-----?-----?
|          test-scripts |         1 |         0 |
?-----?-----?-----?
|    prerequest-scripts |         0 |         0 |
?-----?-----?-----?
|             assertions |         3 |         0 |
?-----?-----?-----?
| total run duration: 250ms                               |
?-----?-----?-----?
| total data received: 611B (approx)                       |
?-----?-----?-----?
| average response time: 207ms                             |
?-----?-----?-----?
[Wrapper_Demo_2-2017-03-30-19-22-52] [runCollection][Get BIG-IP B Software_
↪Version] running...
newman

BIGIP_Operational_Workflows

? 4A_Get_BIGIP_Version
? Get Software Version
  GET https://10.1.1.5/mgmt/tm/sys/software/volume [200 OK, 1.32KB, 191ms]
  ✓ [GET Response Code]=200
  ✓ [Populate Variable] bigip_version=12.1.1
  ✓ [Populate Variable] bigip_build=1.0.196

?-----?-----?-----?
|                               | executed |   failed |
?-----?-----?-----?
|             iterations |         1 |         0 |
?-----?-----?-----?
|             requests  |         1 |         0 |
?-----?-----?-----?
|          test-scripts |         1 |         0 |
?-----?-----?-----?
|    prerequest-scripts |         0 |         0 |
?-----?-----?-----?
|             assertions |         3 |         0 |
?-----?-----?-----?
| total run duration: 230ms                               |
?-----?-----?-----?
| total data received: 611B (approx)                       |
?-----?-----?-----?
| average response time: 191ms                             |
?-----?-----?-----?
[Wrapper_Demo_2-2017-03-30-19-22-52] run completed in 3s, 316.921 ms

```

- Examine the environment variables that were saved at the end of the run by executing `cat Wrapper_Demo_2-env.json`. The resulting BIG-IP software versions are now present and have been highlighted below.

Example output:

```

1  {
2  "id": "d459e491-4936-4be7-a910-567f711a636a",
3  "values": [
4      {
5          "type": "any",
6          "value": "10.1.1.4",
7          "key": "bigip_a_mgmt"
8      },
9      {
10         "type": "any",
11         "value": "10.1.1.5",
12         "key": "bigip_b_mgmt"
13     },
14     {
15         "type": "any",
16         "value": "10.1.1.5",
17         "key": "bigip_mgmt"
18     },
19     {
20         "type": "any",
21         "value": "admin",
22         "key": "bigip_username"
23     },
24     {
25         "type": "any",
26         "value": "admin",
27         "key": "bigip_password"
28     },
29     {
30         "type": "any",
31         "value": "UE7W5CXWM5SJ6SZEVS7KTAI5Q",
32         "key": "bigip_a_token"
33     },
34     {
35         "type": "any",
36         "value": "ONQXOQPNCVOHZELKIFSPHETL3I",
37         "key": "bigip_b_token"
38     },
39     {
40         "type": "any",
41         "value": "ONQXOQPNCVOHZELKIFSPHETL3I",
42         "key": "bigip_token"
43     },
44     {
45         "type": "any",
46         "value": "12.1.1",
47         "key": "bigip_a_version"
48     },
49     {
50         "type": "any",
51         "value": "1.0.196",
52         "key": "bigip_a_build"
53     },
54     {
55         "type": "any",
56         "value": "1200",
57         "key": "bigip_token_timeout"
58     },
59 ]

```



```

59     {
60         "type": "any",
61         "value": "12.1.1",
62         "key": "bigip_b_version"
63     },
64     {
65         "type": "any",
66         "value": "1.0.196",
67         "key": "bigip_b_build"
68     }
69 ]
70 }

```

## 4.5 Module 5 - Python SDK

This module will cover the newly released F5 Python SDK. This SDK is released and maintained as a public GitHub repository at <https://github.com/F5Networks/f5-common-python>

The goal of the Python SDK is to provide a simple interface that abstracts many of the F5-specific nuances of the iControl REST API away from the user. As you learned in Module 1, when interacting directly with the API, it's often necessary to build out requests in a very manual fashion. In order to provide a simpler interface, the SDK was developed to abstract away many of the eccentricities of the API and provide a clean, Pythonic interface.

For example, when creating a pool in, an Imperative automation model, without the SDK you would be required to do something like the following (this code is not complete):

```

import requests
import sys
base_url = "https://10.1.1.4/mgmt/tm/ltm/pool/"

pool_attributes = {
    "name": "test_pool",
    "partition": "Common",
    "loadBalancingMode": "least-connections-member",
    "minUpMembers": 1
}

s = requests.session()
s.auth = ("admin", "admin")

resp = s.post(base_url, data=json.dumps(pool_attributes))

if resp.status_code != requests.codes.ok:
    print "Error creating pool"

sys.exit(1)

```

When using the Python SDK the equivalent code is:

```

from f5.bigip import ManagementRoot

mgmt = ManagementRoot("10.1.1.4", "admin", "admin")

pool = mgmt.tm.ltm.pools.pool.create(partition="Common", name="test_pool")

```

```
pool.loadBalancingMode = "least-connections-member"
pool.minUpMembers = 1

pool.update()
```

As you can see, the code utilizing the SDK is much more condensed and far easier to read. This is a result of the SDK exposing abstracted methods to build the URL. Additionally the SDK creates standard CURDLE (create, update, refresh, delete, load, exists) methods that behave correctly depending on REST object type (Organizing Collection, Resource, etc.) you are interacting with (e.g., you cannot DELETE an Organizing Collection, therefore a delete() method is not available).

Full documentation for the API exists at [here](#)


For the purpose of this lab, your Windows Jumphost has everything pre-installed, however, since the SDK is a standard python package the process is trivial on any system (Windows, Linux, Mac, etc.) that has Python installed.

It's important to keep in mind, while going through this module, that we are only demonstrating what is possible with the SDK from a high level. For example, the same scripts used in this module are designed to run from the command line with arguments, however, they could easily be modified to use JSON files as the input mechanism.

## 4.5.1 Lab 5.1 – create\_pool.py

In this lab we will review, line-by-line an example script that has been created to allow creation of a BIG-IP Pool with Pool Members directly from the command line.

### Task 1 – Review create\_pool.py

1. Open Notepad++ using the  located in the Windows Taskbar.
2. Double click the file `create_pool.py` in the menu on the left side of the Notepad++ screen
3. We will now review the code line-by-line:

```
from f5.bigip import ManagementRoot
import pprint
import argparse

pp = pprint.PrettyPrinter(indent=3)
```

These lines import in various Python libraries. The first line imports the F5 Python SDK. The pprint and argparse libraries are standard Python libraries that aid in print data to the console and parsing command line arguments.

```
parser = argparse.ArgumentParser(description='Script to create a pool on a BIG-IP_
↳device')
parser.add_argument("host", help="The IP/Hostname of the BIG-IP device")
parser.add_argument("pool_name", help="The name of the pool")
parser.add_argument("pool_members", help="A comma seperated string in the format <IP>:
↳<port>[,<IP>:<port>]")
parser.add_argument("-P", "--partition", help="The partition name", default="Common")
parser.add_argument("-u", "--username", help="The BIG-IP username", default="admin")
parser.add_argument("-p", "--password", help="The BIG-IP password", default="admin")
args = parser.parse_args()
```

These lines setup the command line arguments for the script and store those arguments in a python dictionary names 'args'. The argparse library automatically generates help text, checks for required arguments, sets defaults, etc.

```
mgmt = ManagementRoot(args.host, args.username, args.password)
```

This line creates a new Python object that refers to the BIG-IP device. We are calling the ManagementRoot method with 3 arguments:

- The value of the `host` argument
- The value of the `username` argument
- The value of the `password` argument

This method automatically performs a test to ensure that we are able to reach the device and authenticate successfully.

```
pool_path = "/%s/%s" % (args.partition, args.pool_name)
```

This line just stores the human-readable path to the pool name for later use

```
if mgmt.tm.ltm.pools.pool.exists(partition=args.partition, name=args.pool_name):  
    raise Exception("Pool '%s' already exists" % args.pool_name)
```

This if statement checks to see if a pool with the same name already exists on the specified partition on the device. The return value of the `exists()` method is a Boolean value of True or False. In this case we want the Exception to execute if a pool DOES exist and stop execution of the script.

```
pool = mgmt.tm.ltm.pools.pool.create(partition=args.partition, name=args.pool_name)  
print "Created pool %s" % pool_path
```

The first line in this block actually creates the new pool. The partition and name of the pool are specified as arguments to the `create()` method and the 'pool' variable represents an object that holds the created pool's properties. The second line simply prints a message that the pool has been created.

```
member_list = args.pool_members.split(',')
```


This line uses a built-in python method called `split()` to separate the value of the command line argument into discrete strings using a ',' as a separator. The return type of the `split()` is a python list (lists = arrays)

```
for member in member_list:  
    pool_member = pool.members_s.members.create(partition=args.partition, name=member)  
    print " Added member %s" % member
```

This for loop iterates over the elements in the list generated above and creates a new member in the pool.

## Task 2 – Run create\_pool.py



1. Open Console2 using the  icon on the Windows Taskbar
2. The console window automatically opens in the Desktop\Module 5 – Python SDK directory
3. Type `set PYTHONWARNINGS=ignore` to disable the printing of SSL/TLS warnings about self-signed certificates.

4. Type `python create_pool.py` and examine the help output:

```
C:\Users\user\Desktop\Module 3 - Python SDK>python create_pool.py
usage: create_pool.py [-h] [-P PARTITION] [-u USERNAME] [-p PASSWORD]
                    host pool_name pool_members
create_pool.py: error: too few arguments
```

5. Type `python create_pool.py 10.1.1.4 test_pool 10.1.10.10:80,10.1.10.11:80` to create a new pool:

```
C:\Users\user\Desktop\Module 3 - Python SDK>python create_pool.py 10.1.1.4 test_pool 10.1.10.10:80,10.1.10.11:80
Created pool /Common/test_pool
Added member 10.1.10.10:80
Added member 10.1.10.11:80
```

6. Using Chrome open a tab to BIGIP-A (<https://10.1.1.4>). Examine the pool that was created.

## 4.5.2 Lab 5.2 – read\_pool.py

In this lab we will review, line-by-line an example script that has been created to view the attributes of a BIG-IP Pool directly from the command line.

### Task 1 – Review read\_pool.py

1. Open `read_pool.py` in Notepad++
2. We will review the code. For brevity we have removed lines that are common with previous examples:

```
if not mgmt.tm.ltm.pools.pool.exists(partition=args.partition, name=args.pool_name):
    raise Exception("Pool '%s' does not exist" % args.pool_name)
```

This if statement checks to see if a pool with the same name exists in the specified partition on the device. The key difference between this and the example in the previous lab is the inclusion of the 'not' keyword. This inverts the logic of the statement so that the Exception is raised when the pool DOES NOT exist

```
pool = mgmt.tm.ltm.pools.pool.load(partition=args.partition, name=args.pool_name)
```

This line loads the configuration of the pool into a variable

```
print "Pool %s:" % pool_path
pp.pprint(pool.raw)
```

These lines print the human-readable pool path and then uses the PrettyPrint library to dump all the attributes associated with the pool

### Task 2 – Run read\_pool.py

1. In the command prompt type `python read_pool.py 10.1.1.4 test_pool` and examine the output:

```
C:\Users\user\Desktop\Module 3 - Python SDK>python read_pool.py 10.1.1.4 test_pool
Pool /Common/test_pool:
{ '_meta_data': { 'allowed_commands': [],
'allowed_lazy_attributes': [ <class 'f5.bigip.tm.ltm.pool.Members_s'>],
'attribute_registry': ( 'tm:ltm:pool:memberscollectionstate': <class 'f5.bigip.tm.ltm.pool.Members_s'>),
'bigip': <f5.bigip.ManagementRoot object at 0x02FDCB90>,
'container': <f5.bigip.tm.ltm.pool.Pools object at 0x02FF5EB0>,
'creation_uri_frag': '',
'creation_uri_qargs': { 'u'ver': [u'12.0.0']},
'exclusive_attributes': [],
'icontrol_version': '',
'icr_session': <icontrol.session.iControlRESTSession object at 0x02FDCA50>,
'minimum version': '11.6.0',
'read_only_attributes': [],
'required_command_parameters': set([]),
'required_creation_parameters': set(['name']),
'required_json_kind': 'tm:ltm:pool:poolstate',
'required_load_parameters': set(['name']),
'uri': u'https://10.1.1.4:443/mgmt/tm/ltm/pool/~Common-test_pool/'),
u'allowNat': u'yes',
u'allowSnat': u'yes',
u'fullPath': u'/Common/test_pool',
u'generation': 5191,
u'ignorePersistedWeight': u'disabled',
u'ipToClient': u'pass-through',
u'ipToServer': u'pass-through',
u'kind': u'tm:ltm:pool:poolstate',
u'linkQosToClient': u'pass-through',
u'linkQosToServer': u'pass-through',
u'loadBalancingMode': u'round-robin',
u'membersReference': ( u'isSubcollection': True,
u'link': u'https://localhost/mgmt/tm/ltm/pool/~Common-test_pool/members?ver=12.0.0'),
u'minActiveMembers': 0,
u'minUpMembers': 0,
u'minUpMembersAction': u'failover',
u'minUpMembersChecking': u'disabled',
u'name': u'test_pool',
u'partition': u'Common',
u'queueDepthLimit': 0,
u'queueOnConnectionLimit': u'disabled',
u'queueTimeLimit': 0,
u'reselectTries': 0,
u'selfLink': u'https://localhost/mgmt/tm/ltm/pool/~Common-test_pool?ver=12.0.0',
u'serviceDownAction': u'none',
u'slowRampTime': 10}
```

2. Notice the various attributes that are associated with the pool. Take note of the value of the `loadBalancingMode` attribute for the next lab

### 4.5.3 Lab 5.3 – update\_pool.py

In this lab we will review, line-by-line an example script that has been created to allow updating any attribute of a pool using the command-line. This script is a good example of creating generic tools that enable many use cases. Rather than creating a script that just updates a specific attribute we created one that updates ANY pool attribute, greatly expanding its potential use cases.

#### Task 1 – Review update\_pool.py

1. Open `update_pool.py` in Notepad++
2. We will review the code. For brevity we have removed lines that are common with previous examples:

```
pool = mgmt.tm.ltm.pools.pool.load(partition=args.partition, name=args.pool_name)
pp.pprint("Current: %s=%s" % (args.attribute, getattr(pool, args.attribute)))
```

These lines load the pool from the device and print the current value of the attribute specified on the the command line. The second line is a little bit tricky because the SDK dynamically populates the objects attributes based on the type of object (pool, virtual server, etc.). Normally we could just use something like `pool.loadBalancingMode` to get the current lb-method for the pool, however, since this script implements a way to change ANY attribute in the object we have to dynamically substitute the attribute name at run-time.

To do this we use the `getattr()` python built-in function to resolve the mapping at runtime and return the value of the attribute specified on the command line.

```
kwargs = {args.attribute: args.value}
```

This line creates a new python dictionary with one entry specifying a key-value pair using the command line arguments. For example if you were updated the `loadBalancingMode` attribute to 'least-connections-member' the dictionary would look like `{"loadBalancingMode": "least-connections-member"}`

```
pool.update(**kwargs)
```

The first line updates the pool we loaded previously with the new value for the attribute. The `**kwargs` argument to the `update()` method triggers a special mechanism in python called 'keyword unpacking' which allows us to pass the attribute to be updated to the `update()` method.

```
pool.refresh()
pp.pprint("New: %s=%s" % (args.attribute, getattr(pool, args.attribute)))
```

The first line refreshes the data in the object from the BIG-IP device. The second line prints this refreshed information to the console so the user can verify the update completed successfully.

## Task 2 – Run `update_pool.py`

1. In the command prompt type `python update_pool.py 10.1.1.4 test_pool loadBalancingMode least-connections-member` and examine the output:

```
C:\Users\user\Desktop\Module 3 - Python SDK>python update_pool.py 10.1.1.4 test_pool loadBalancingMode least-connections-member
u'Current: loadBalancingMode=round-robin'
Updating pool /Common/test_pool
u'New: loadBalancingMode=least-connections-member'
```

2. You can manually verify the load balancing method was changed via TMUI or by re-running `read_pool.py` (it's not required since the line that prints the new value forces a refresh() )
3. Experiment with changing other pool attributes

## 4.5.4 Lab 5.4 – `update_pool_member_state.py`

One of the most common tasks asked for by customers is the ability to set a pool member's state via a script. We have included an example of such a script in the lab that can be used to see how easy it is to automate specific operational tasks.

## Task 1 – Run `update_pool_member_state.py`

1. In the command prompt type `python update_pool_member_state.py 10.1.1.4 test_pool 10.1.10.10:80 disabled` and examine the output.
2. Verify the pool member was disabled via TMUI
3. Re-run the script with `python update_pool_member_state.py --help` to see additional options.
4. Re-enable the pool member using the script

### 4.5.5 Lab 5.5 – delete\_pool.py

In this lab we will review, line-by-line an example script that has been created to allow deletion of a pool using the command-line.

#### Task 1 – Review delete\_pool.py

1. Open delete\_pool.py in Notepad++
2. We will review the code. For brevity we have removed lines that are common with previous examples:

```
pool = mgmt.tm.ltm.pools.pool.load(partition=args.partition, name=args.pool\_name)
pool.delete()

print "Deleted pool %s" % pool\_path
```

These lines should be fairly self-explanatory at this point. First we load the pool and then we delete() it and print that we have done so.

#### Task 2 – Run delete\_pool.py

1. In the command prompt type `python delete_pool.py 10.1.1.4 test_pool` and examine the output:

```
C:\Users\user\Desktop\Module 3 - Python SDK>python delete_pool.py 10.1.1.4 test_pool
Deleted pool /Common/test_pool
```

2. If desired verify the pool was deleted using TMUI or the `read_pool.py` script (it should return an error)

### 4.5.6 Lab 5.6 – Create a Python Script

In this lab we will use the 'Generate Code' feature of Postman to create a python script from a collection of requests.

#### Task 1 – Create a simple script

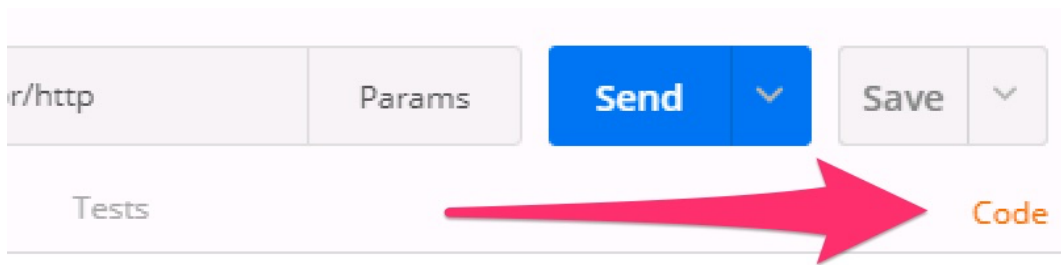
---

**Note:** Remember to have the correct environment selected in Postman

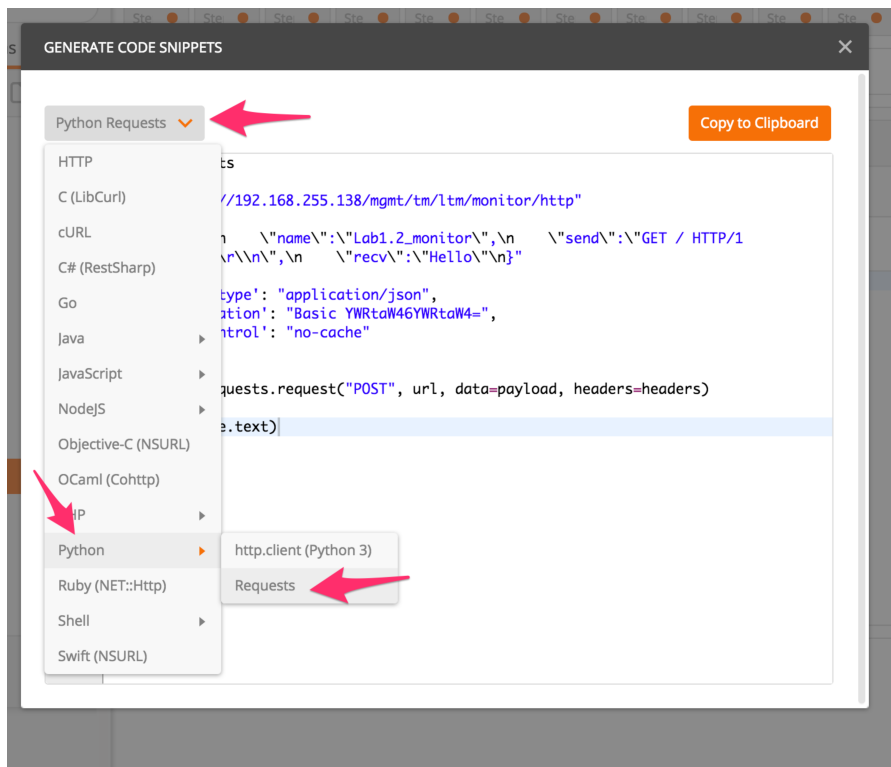
---

Perform the following steps to complete this task:

1. Expand the 'Lab 5.6 – Create a Python Script' folder in the Postman collection
2. Click the 'Step 1 – Create a HTTP Monitor' item in the collection
3. Click the 'Code' link in the Postman window:



4. Select Python -> Requests from the menu on the top right of the window:



5. Examine the Python code that was generated. Click the 'Copy to Clipboard' button
6. Open a new text file and paste the generated code. We need to modify the line that sends the request to DISABLE SSL certificate verification. Find the following line:

```
response = requests.request("POST", url, data=payload, headers=headers)
```

And add a verify=False option to it:

```
response = requests.request("POST", url, data=payload, headers=headers,
    ↪ verify=False)
```



### Save the file on your Desktop as lab5\_6.py

7. Open a command prompt and run the script by typing `python lab5_6.py`:

```
C:\Users\user\Desktop>python lab1_2.py
C:\Python27\lib\site-packages\requests\packages\urllib3\connectionpool.py:821: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.org/en/latest/ssl-security.html
  InsecureRequestWarning)
{"Kind": "tm:ltm:monitor:http:httpstate", "name": "Lab1.2_monitor", "fullPath": "Lab1.2_monitor", "generation": 0, "selfLink": "https://localhost/mgmt/tm/ltm/monitor/http/Lab1.2_monitor?ver=12.0.0", "adaptive": "disabled", "adaptiveDivergenceType": "relative", "adaptiveDivergenceValue": 25, "adaptiveLimit": 200, "adaptiveSamplingTimespan": 300, "defaultsFrom": "/Common/http", "destination": "*", "interval": 5, "ipDscp": 0, "manualResume": "disabled", "recv": "Hello", "reverse": "disabled", "send": "GET / HTTP/1.0\r\n\r\n", "timeUntilUp": 0, "timeout": 16, "transparent": "disabled", "upInterval": 0}

C:\Users\user\Desktop>
```

8. Verify the monitor was created on BIG-IP
9. **Delete the monitor to prepare for the next task**

## Task 2 – Chain together multiple requests

In this task we will repeat the process from Task 1 to chain together multiple requests.

Perform the following steps:

1. Repeat the procedure from Task 1 with each of the items in the ‘Lab 5.6’ postman collection. Append each snippet of code to your existing script until you have all 5 requests in the script. **You will need to remove the duplicate ‘import requests’ lines and update each request with the ‘verify=False’ option.**
2. Save the file
3. Run the script and verify the config was created.

### 4.5.7 Lab 5.7 – EXTRA CREDIT – Modify create\_pool.py

This is an open-ended exercise. Copy `create_pool.py` to `create_vs.py` and modify it to create a Virtual Server. You could also cheat and look at `you_cheated.py`!

### 4.5.8 Lab 5.8 – EXTRA CREDIT – Review super\_pool.py

This is an open-ended exercise. Review and run the `super_pool.py` script. This script allows bulk creation/deletion of pools using CSV files.



This section contains useful HOWTOs

## 5.1 HOWTO - Update Existing iApp templates to Work with iWorkflow v2.1

This HOWTO document describes the minimal changes required to update an existing iApp template and add a version number to the template name.

Adding the version number allows the iApp template to be used by iWorkflow v2.1 and later. Versioning is required to enable iApp templates to be installed across many BIG-IP devices in a production-safe manner.

Without version information it is possible that iApp templates could be overwritten leading to deployment failures and/or outages.

### 5.1.1 Task 1 – Export the existing iApp from BIG-IP

The iApp template can be exported from a BIG-IP system where it has been installed. The file has a `.tmpl` extension and is a plaintext, readable format.

Complete the following steps:

1. Login to the BIG-IP GUI with admin credentials
2. Click iApps -> Templates
3. Find the desired template in the list and click the template name to open it
4. Scroll to the bottom of the page and click the 'Export' button
5. Click the Download: . . . button and save the file to your computer

### 5.1.2 Task 2 - Edit the Exported template

We will now edit the template name to add a version number. iWorkflow currently supports the following formats:

- `template_name_v1.0_0`

- `template_name.v.1.0.0`
- `/<partition>/template_name.v1.0.0`

Complete the following steps:

1. Open the previously saved `.tmpl` file in a text editor
2. Perform a text search for `sys application template`

Example:

```

1 cli admin-partitions {
2     update-partition Common
3 }
4
5 sys application template my_template_name {
6     actions {
7         definition {
8             implementation {

```

3. Modify the template name to include a version number using one of the formats specified at the beginning of this task.

Example:

```

1 cli admin-partitions {
2     update-partition Common
3 }
4
5 sys application template my_template_name.v1.0.0 {
6     actions {
7         definition {
8             implementation {

```

4. Save the file

### 5.1.3 Task 3 - Import the iApp template to iWorkflow

The updated iApp template is now ready to be imported to iWorkflow. Instructions on how to do this can be found at:

[https://devcentral.f5.com/wiki/iWorkflow.iWorkflowOpsGuide\\_7.ashx](https://devcentral.f5.com/wiki/iWorkflow.iWorkflowOpsGuide_7.ashx)

WE MAKE APPS  FASTER.  
SMARTER.  
SAFER.

F5 Networks, Inc. | [f5.com](https://f5.com)



US Headquarters: 401 Elliott Ave W, Seattle, WA 98119 | 888-882-4447 // Americas: [info@f5.com](mailto:info@f5.com) // Asia-Pacific: [apacinfo@f5.com](mailto:apacinfo@f5.com) // Europe/Middle East/Africa: [emeainfo@f5.com](mailto:emeainfo@f5.com) // Japan: [f5j-info@f5.com](mailto:f5j-info@f5.com)  
©2017 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at [f5.com](https://f5.com). Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. These training materials and documentation are F5 Confidential Information and are subject to the F5 Networks Reseller Agreement. You may not share these training materials and documentation with any third party without the express written permission of F5.