Table of Contents

[Lab 1 (AS3 – BIG-IQ) 1](#_Toc7959365)

[Lab 2 (Facts and AS3 with ansible – BIG-IP) 2](#_Toc7959366)

[Lab 3 (GIT) 3](#_Toc7959367)

[Lab 4 (POSTMAN) 6](#_Toc7959368)

[Lab 5 (Charon – BIG-IP config 🡪 AS3) 6](#_Toc7959369)

[Lab 6 (Validating declarations) 6](#_Toc7959370)

[Lab 7 (UI) 8](#_Toc7959371)

[Extra labs 11](#_Toc7959372)

[Upcoming labs (in future versions) 11](#_Toc7959373)

## Lab 1 (AS3 – BIG-IQ)

Goal

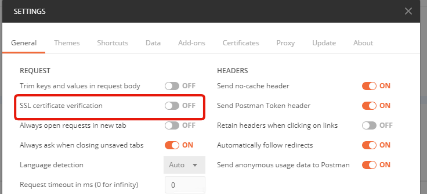
* Understand AS3 schema
* Create AS3 applications
* Read, update and delete AS3 applications
* Use the schema to create any type of application

Study the schema reference - [ <https://clouddocs.f5.com/products/extensions/f5-appsvcs-extension/latest/refguide/schema-reference.html> ] – (take some time going over it). This will be your guide when creating a new declaration.

Also look at some basic declaration examples (note that they are meant for BIG-IP direct, so if using pointing to the BIG-IQ, target needs to be specified) –

<https://clouddocs.f5.com/products/extensions/f5-appsvcs-extension/latest/userguide/examples.html>

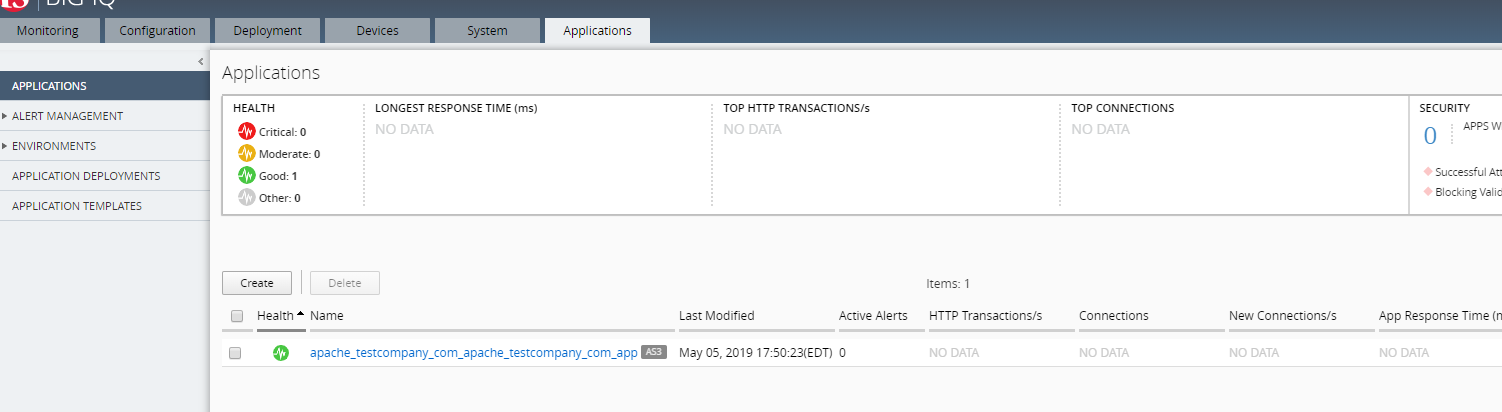
<https://clouddocs.f5.com/products/extensions/f5-appsvcs-extension/latest/declarations/>

* RDP to your windows machine in UDF (login as user, user)
* Open postman (you can login to your account or not login at all), make sure to turn ssl verification off.
* 
* Import the provided collection and environment (desktop 🡪 lab files)

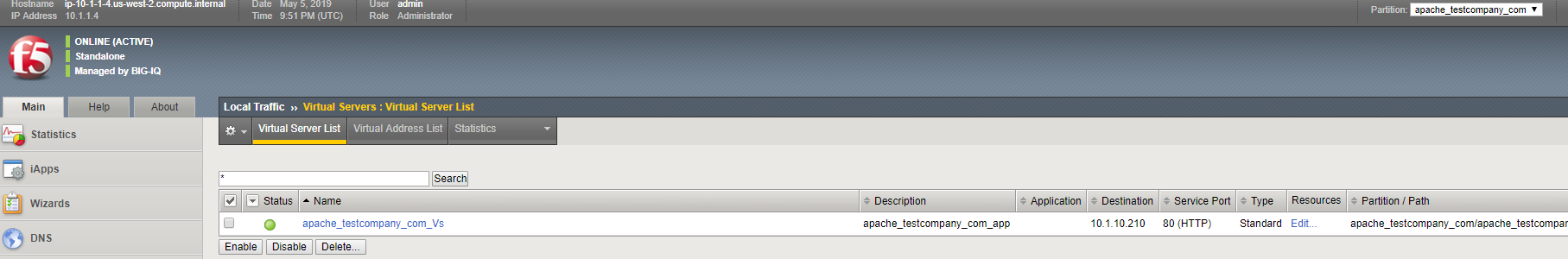
**[NOTE – postman should already be setup]**

* Use environment variables to create the provided apps
  + <https://hive.f5.com/docs/DOC-50438> (instructions on using the collection)
* Create the following applications using the collection provided (IP ranges for VS – 10.1.10.201 – 210)
  + HTTP
  + HTTP with LTM policy
  + HTTPs Offload
  + HTTPs Bridging
  + TCP
  + UDP
* Observe the response,
* Use the ID in the response to check the status of the deployment task
* Check the rest logs on BIG-IQ (/var/log/restjavad.0.log)

Once created, check the applications section of the BIG-IQ to see if you see new applications.



Also look at the BIG-IP to see how the applications are created, are they in partitions?



* Read the above applications and observe the output using the collection provided
  + List all apps
  + List one app
  + If you were designing a page which supports CRUD, think optimization while listing multiple apps
* Delete an app using the tenant
* Study the jinja2 templating used (in postman)
* Create the following applications (not provided) [ use the schema]
  + HTTP with SNAT pool
  + Based on SSL SNI match conditions, steer traffic to separate pools
  + A declaration using the metadata – how can you leverage metadata?
  + Compare declarations created using HTTP template and generic template
  + Can you disable a pool member when using HTTP template Vs Generic template?
* Study and play with the age parameter (<https://clouddocs.f5.com/products/extensions/f5-appsvcs-extension/latest/refguide/as3-api.html?highlight=history#method-get>)
* Study the “AS3 will all properties” page (have it as a ref) - <https://clouddocs.f5.com/products/extensions/f5-appsvcs-extension/latest/declarations/all-properties.html>

## Lab 2 (Facts and AS3 with ansible – BIG-IP)

Goal – Use ansible for BIG-IP fact gathering and creating applications using AS3

1. bigip\_device\_facts – Collect facts from F5 BIG-IP devices

<https://docs.ansible.com/ansible/latest/modules/bigip_device_facts_module.html#bigip-device-facts-module>

Device facts outputs data about existing BIG-IP devices in JSON format. For example, it can provide data about virtual servers, pools, ssl-certs, profiles etc. This data can be leveraged to analyze the existing environment to understand the existing applications (specially in large environments). Also, since its JSON it can be programmatically leveraged.

Lab steps

* Go to the directory /home/ubuntu/ansible/techdayansible/lab-bigip-facts
* Run the playbook (ansible-playbook -i inventory/hosts playbooks/site.yaml)
* Watch the output (on the screen as well as in the host\_vars), what is it obtaining?
* Change the playbook to get facts for what you want (see docs above)

1. Uri module – ansible has a “uri” module to interact with web services. You can send GET/POST to the AS3 URI using this module.

Lab steps

* Go to the directory /home/ubuntu/ansible/techdayansible/lab-uri
* Run the “site.yaml” playbook, notice the response. Notice the BIG-IQ/BIG-IP, do you see an application created? Is there a way we can display a more meaningful response?
* Run the delete-site.yaml playbook. Do you see the application deleted from the BIG-IP/BIG-IQ?
* What do you think is the downside of using this module (from an ansible perspective) ?

[Note – if necessary – these playbooks can be cloned using “ git clone [git@bitbucket.org:arunhotra/techdayansible.git](mailto:git@bitbucket.org:arunhotra/techdayansible.git) “ ]

## Lab 3 (GIT)

Goal – Understand basic GIT concepts by experimenting with GIT bash and GIT GUI (you can use any GUI of your choice)

Basic git bash steps - <https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html>

Note – you can right click and open ‘git bash’ and ‘git gui’ in the ‘gittest’ folder.

Lab steps

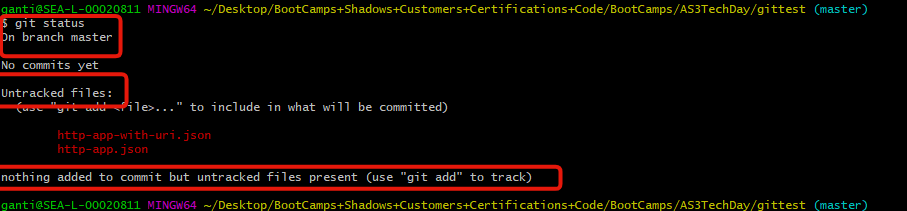
[basic lab] – refer to commands in the link above

* Create a bitbucket/github account (if you do not have already)
* Add your public key to your account (~/.ssh/id\_rsa.pub). If it is not present, create it by using ssh-keygen

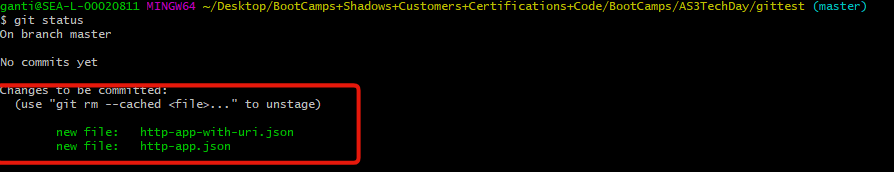
Bitbucket - <https://confluence.atlassian.com/bitbucket/set-up-an-ssh-key-728138079.html#SetupanSSHkey-ssh1SetupSSHforGitonWindows>

Github – <https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account>

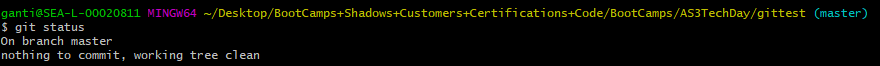
* Initialize a repo in the ‘gittest’ folder(desktop) using git bash. Notice what happens in the folder after you initialize the repository.
* Set name and email globally using git config. This will be used to just as an identifier when committing to the repository.
* Check status



* Add all files to the staging area
* Check status



* Commit with a message
* Check status



* Create a repo in bitbucket/github
* Add the repo created as a remote in your local repository
* Check the remote got added

https://i.gyazo.com/ce9a9873ba205ddebb46b9eae6ec1ac8.png

* Push to remote, verify the remote repository that the files got added.
* Update the http file (change any value you want)
* Open GIT GUI
* Check the un staged, staged and modified areas
* Stage the changed file, notice the change in the windows
* Commit with a new message (e.g. – modified IP address of VIP”)
* Push to remote, notice the commits section of the remote repository has the new commit now.



* Click on the hash value and notice your change
* Create a file named ‘.gitignore’ with the body ‘\*.txt’
* Create two files with random content, name them test.json and testtext.txt
* Add all files in tracking again, notice which files get added. What does adding ‘\*.txt’ to ‘.gitignore’ do? What other type of files can you think of that fit the similar use case?

Go through the following links

[GIT and GITHUB - core concepts] - (<https://www.youtube.com/watch?v=uR6G2v_WsRA>)

[GIT and GITHUB - branching and merging] - (<https://www.youtube.com/watch?v=FyAAIHHClqI&t=209s>)

[GIT and GITHUB - Remotes] - (<https://www.youtube.com/watch?v=Gg4bLk8cGNo&t=71s>)

Try doing the following after going through these videos.

* + [simulating multiple users] - Clone the repository from another location (your laptop), change the git user and email, make a change and push to the remote repository. Observe the new username listed for the new commit
  + [fetch and merge/pull] - In your original gittest folder, make a change and push to the repository. Does git let you push the changes to the ‘master’. What would you have to do to push changes to master?
  + [branching] –
    - create 2 new branches, Dev and UAT.
    - View the branches, how do we know which branch we are working on?
    - Work on the Dev branch. Change the virtual server IP and commit. Switch to the UAT branch. Change the node IP and commit. Merge the UAT branch into master branch (what message does the merge display?). Check that the UAT branch is merged into the master. Delete the UAT branch. Delete the Dev branch. Does it let you delete the Dev branch? Merge the Dev branch to master, notice the merge messages. Delete the Dev branch.
  + [merge conflicts] –
    - Create a branch called BugFixes. Work on it and change the virtual server port to 443 and commit. Work on the master, change the virtual port to 8443 and commit. Merge the BugFixes branch to master. Do you notice any different while checking git status? Fix the conflict.

Video on branching examples from Atlassian <https://www.youtube.com/watch?v=bCU_D7EHqLg>

## Lab 4 (POSTMAN)

Goal – Get comfortable with postman

Lab steps

* Observe the collection used for the AS3 lab
* Examine the templating used
* Check out the pre\_request\_scripts tab, what is it doing?
* Can you write a test which will check for status code 200?
* Can you modify the response of a request so that it displays the actual response instead of “declaration successfully submitted” ?

Reference – [Syed’s presentation on postman](https://f5networks.zoom.us/recording/play/gD-yVBxfyA3KOqdeRdiacVSl9AqI-xybeZMngjS0lOnu3PqSmBMU5KBrljdxMfy4?continueMode=true)

## Lab 5 (Charon – BIG-IP config 🡪 AS3)

Goal – Get comfortable with AS3 conversion tool

* Look at the documentation at <https://hive.f5.com/docs/DOC-50725>
* Login to the linux jump host
* Load the docker image – “ sudo docker load -i dist\_f5-appsvcs-charon-1.0.0.tar.gz “
* Run the container and do the conversion “ sudo docker run --rm -v "$PWD":/app/data f5-appsvcs-charon:1.0.0 -o data/output.json -u data/as3-charon-test.ucs “
* Examine the “output.json” file

## Lab 6 (Validating declarations)

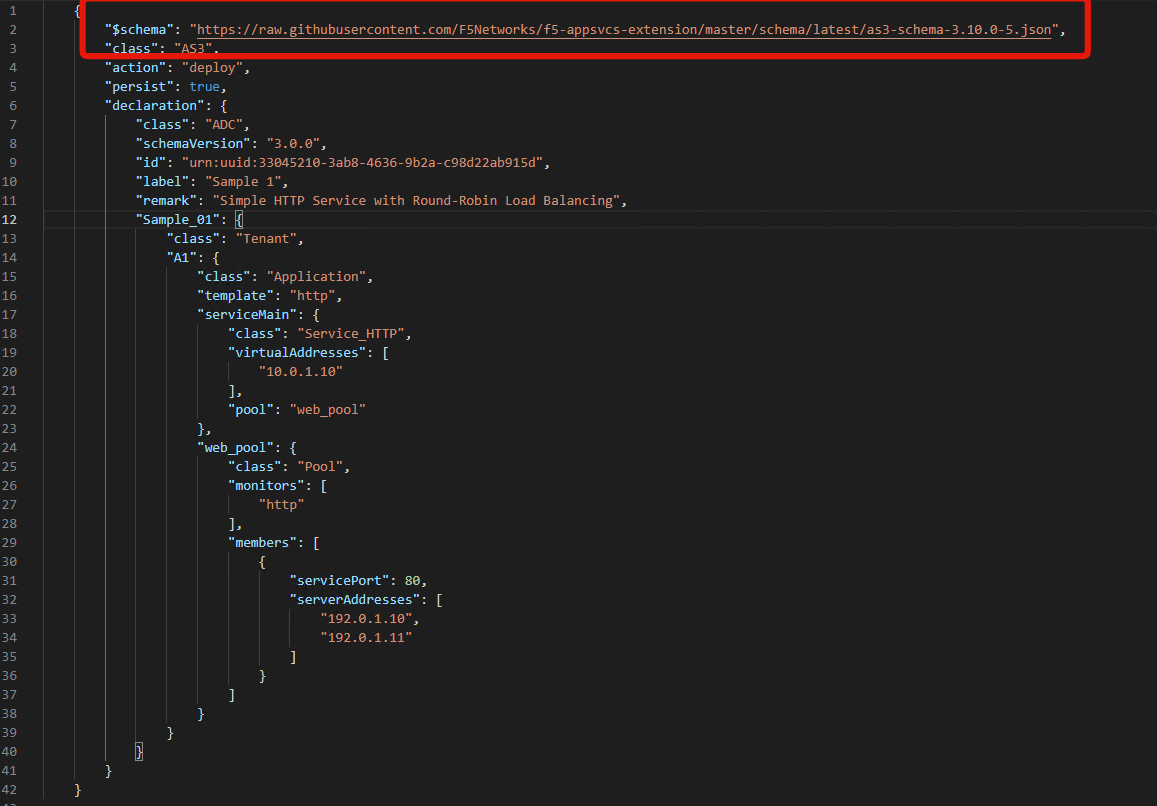
Goal – Explore validating declarations

1. VS code for validating declarations – This is used for manual validation of declarations. This comes with F5 official docs.

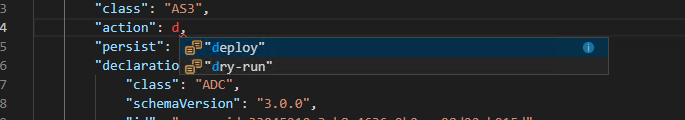
<https://clouddocs.f5.com/products/extensions/f5-appsvcs-extension/latest/userguide/validate.html>

Lab steps

* Study the link above
* Open VS code on the desktop
* Open the file VScode-validation.json (lab-files)
* Look at the $schema line up top (pointing to the latest schema)



* Notice that as you write the declaration, the editor will give you suggestions



* Notice that the editor will also do syntax highlighting if the declaration syntax is incorrect



1. Validation declarations programmatically – Most of the time, the declarations will be handled by a piece of code, and there should be a way of programmatically verifying if the declaration is valid or not. There are some 3rd party validators available for that. Example of one of them is - <https://github.com/Julian/jsonschema>

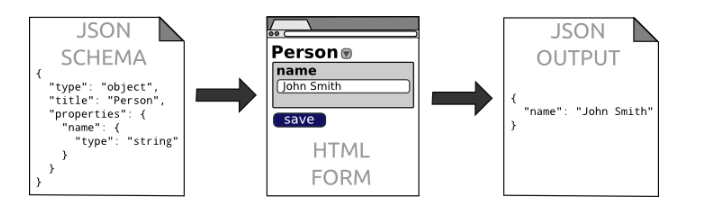
## Lab 7 (UI)

Goal – To explore some of the possible ways of UI generation

1. Generating forms using JSON schema – “JSON-editor”

<https://github.com/json-editor/json-editor>

JSON Editor takes a JSON Schema and uses it to generate an HTML form. Then it uses the data entered in the form to create a JSON document. So basically,



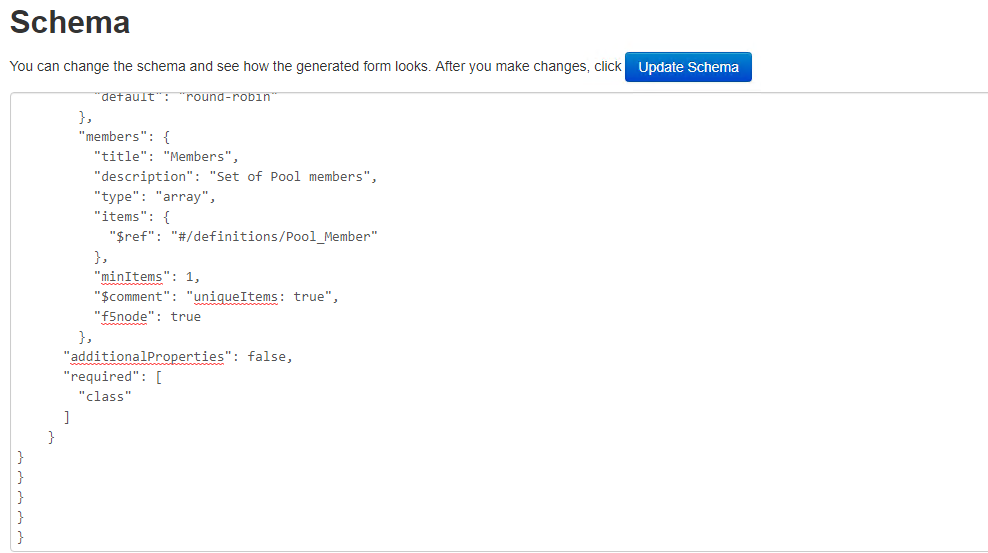
This is one of many open source tools which can be leveraged to generate UI. AS3 always publishes its JSON schema with every release.

<https://github.com/F5Networks/f5-appsvcs-extension/tree/master/schema/latest>

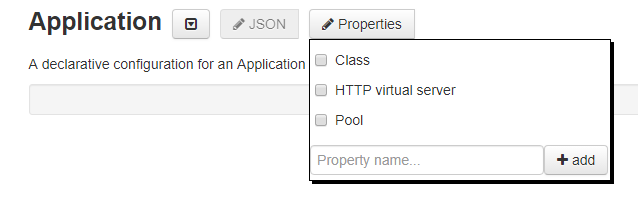
This schema can be leveraged to dynamically generate a UI (front end form) to be presented to the user who is requesting an application.

Lab steps

* Login as user to the windows machine.
* Open the labfiles folder and open the “json editor” folder inside that. Note that there are two json files, SimpleSchema and as3-schema-draft4.
* From the chrome bookmarks click on “JSON editor”. Copy the simpleSchema and paste it in the schema window below, click on “update schema”.



* You will see the following page, check “HTTP virtual server” and “Pool”



* Monitor the JSON output window
* Observe that the UI changes when the properties are selected. Fill in the details for virtual server and pool and observe the JSON output also changing.



* Try the same procedure with the “as3-schema-draft4” schema. Notice how many more options there are. This is the full schema (tweaked to draft 4 – as this tool supports only JSON draft 4).

## Extra labs

If your familiarity level with the above labs is high, then you can try these.

1. [medium difficulty] “CI/CD Demo lab” on UDF
2. [high difficulty] Write a library in the language of your choice supporting CRUD operations for applications. The library would be a class which can be instantiated by passing the credentials of BIG-IQ (IP, user, pass). Some of the supported functions to think about are the following (think of what data should be input, and how the response should be returned).

* Listing all declarations
* Listing one declaration in detail
* Creating app
* Dry run apps
* Deleting apps

## Upcoming labs (in future versions)

* As3deploy tool – brand new tool that PD is working on (depending on customer need). This tool uses mustache templates to create basic web GUI’s for different type of applications, e.g. HTTP app, HTTPS app. Currently it is in proof of concept (alpha) stage.
* Ansible appsvs module – Lab with the ansible “bigip\_appsvcs\_extension” module. Currently, I was having issues running this module (possibly due to a bug) and hence I didn’t include it. Also, currently the ansible labs are very basic, I will modify it to look more like a real production environment.
* JSON programmatic validation – Usage of the “jsonschema” tool for JSON validation, i.e. <https://github.com/Julian/jsonschema>
* DO – Declarative Onboarding, declaratively configuring the base config for BIG-IPs
* Telemetry streaming – gathering facts about Big-IP applications declaratively and sending to specific providers (as a single JSON). This can be consumed by the corresponding application performance management tool of the customer.
* BIG-IQ stats – simulating real traffic and observing BIG-IQ statistics collected by DCD’s