

# The iRules Tool Kit

(Common Use Cases of F5  
iRules to NSX Advanced-Load  
Balancer (Avi)  
Policies/DataScripts, and  
Profiles)

VMware Network & Security Business Unit

## Table of Contents

<b>About this Document</b>	4
Helpful Links for Migrating to the NSX Advanced Load Balancer	4
<b>Summary</b>	5
<b>Document Purpose</b>	6
<b>Network Security Policy:</b>	6
Deny Source IP iRule	7
Creating an IP Group to match the IP's in the Deny Source IP Rule	8
Deny Based On Country Codes—block_country_group	11
Match HTTP Path, Deny Connection:	16
Match Path and respond with HTTP 404	18
Deny IP Group and respond with HTTP 200 message	20
Match HTTP path, Client IP Group and Close Connection	23
<b>HTTP Request Policy</b>	25
Add HTTP Header	25
Insert SSL Information in the HTTP Header	28
<b>Content Switching</b>	30
HTTP Redirection by matching the HTTP Path	33
<b>HTTP Response Policy</b>	35
Add HTTP Header during HTTP Response	35
Insert Origin Information in HTTP Header during Response	37
<b>DataScripts</b>	39
DS Library Link	39
True Client IP Replacement	39
Insert Pool Member IP in HTTP Header	40
Add Cookie Domain Attribute	41
Cookie Rewrite	42
TLS Version and IP Group Match	44
JSession ID Persistence	46
TLS1.2_Version Check and Respond with HTML Page	47
<b>Virtual Service Settings</b>	48
X-Forwarded-For	48
HTTP To HTTPS Redirection	51
Maintenance Page	53
Logging Settings	55
<b>Appendix</b>	57
<b>Glossary</b>	59



## About this Document

This document has been created by VMware NSBU NSX Advanced Load Balancer (Avi) Field Engineering to facilitate migrations from legacy load balancing appliances such as the F5 Local Traffic Manager (LTM) to the software defined NSX Advanced Load Balancer (Avi) platform. The primary motivation behind this document is the fact that during migrations from F5 LTM to Avi, the standard migration toolkit does not automatically migrate or convert F5 iRules. This document is intended as an aid for conversion of commonly encountered F5 LTM iRule use cases into equivalent Avi policies and/or datascripts and provides several illustrative examples and approaches on how to utilize templates and other system properties to replace iRule actions. It also provides details on applying the converted iRules to Avi configuration using the migration tool kit as well as references to a public Avi datascript library as appropriate.

Due to the broad variability in logic and implementation of the TCL scripting engine that F5 iRules are based on, it is not possible to provide an exhaustive reference covering all possible use cases. However, the focus of this document is the conversion of the most common use cases that have been identified after analyzing several LTM configuration files. For a majority of the identified iRules, the Appendix provides the required workflow for implementing the equivalent policies/datascripts via the user interface as well as the required YAML code for application via the command line using the `f5_converter.py` script, which is part of the standard migration toolkit.

For more details on Avi datascripts and the migration toolkit please refer the following links

### Helpful Links for Migrating to the NSX Advanced Load Balancer

<https://avinetworks.com/docs/latest/migrate-from-f5-big-ip-to-avi-vantage/>

#### DataScripts

<https://avinetworks.com/docs/latest/overview-of-datascript/>

<https://avinetworks.com/docs/latest/datascript-guide/datascript-samples/>

#### Policies

<https://avinetworks.com/docs/latest/configuration-guide/applications/vs-policies/>

## Summary

The NSX Advanced Load Balancer uses the following policies (<https://avinetworks.com/docs/latest/configuration-guide/applications/vs-policies/>) to manage and direct application delivery. The policies and DataScripts execute in the order and priority shown below <https://avinetworks.com/docs/latest/datascript-execution-priority/>

- **Network Security Policy** (<https://avinetworks.com/docs/latest/configuration-guide/applications/vs-policies/#network-security>)
  - This policy matches on IP Addresses, TCP/UDP ports, and IP reputation. This policy can then respond by allowing, closing or rate limiting the connection.
- **HTTP Security Policy** (<https://avinetworks.com/docs/latest/configuration-guide/applications/vs-policies/#http-security>)
  - This policy matches on IP Addresses, TCP/UDP ports, Protocol Types, HTTP Method, Version, and Header and Host Header, as well as matching on Path, Query, Cookie, and IP Reputation.
  - This policy can respond by Allowing or closing the connection, redirecting to HTTPS, Rate Limiting, sending an HTTP Response, or enabling ICAP.
- **HTTP Request Policy** (<https://avinetworks.com/docs/latest/http-request-policy/>)
  - This policy matches on the same criteria as HTTP Security; however, it does not close connections.
  - Its actions include Content Switching, redirection, rewrite, and modify (add, remove, replace) headers.
- **HTTP Response Policy** (<https://avinetworks.com/docs/latest/http-response-policy/>)
  - This policy matches on the same criteria as HTTP Security; however, it does not close connections.
  - Its actions include Content Switching, redirection, rewrite, and modify (add, remove, replace) headers.
- **DataScripts** (<https://avinetworks.com/docs/latest/overview-of-datascript/>)
  - The NSX Advanced Load Balancer uses DataScripts and the Lua language to provide custom functionality where required by the use case.

- **Virtual Service Application Profiles** (<https://avinetworks.com/docs/latest/application-profile/>)
  - This category includes actions and parameters relating to Virtual Services, including application templates, pool actions, and default behaviors.

## Document Purpose

The intent of this document is to provide a resource for the user to create the policies described here or provide guidance to adapt to a particular use case. These have been tested. The iRule text boxes are searchable and the YAML code can be copied into a text editor such as Visual Studio Code and used to create.

## Network Security Policy:

The *Network Security Policy* matches on IP Addresses, country codes, TCP/UDP ports, and IP reputation. This policy can then respond by allowing, closing or rate limiting the connection.

- Network Security Policy executes first in policy priority; in the UI it is accessible by selecting (or creating) a virtual service.
- As noted above, NSX Advanced Load Balancer policies operate with match and action. Network security matches on three criteria, IP addresses, service ports, or IP reputation database.
- Network policy then takes the following actions—allow, close, or rate limit the connection.

## Deny Source IP iRule

**The phrase:** when CLIENT\_ACCEPTED means when the client connects, if { not ([matchclass [IP::client\_addr] equals deny\_source\_ip] meaning if the client IP address matches the list of denied IP addresses then, reject, meaning close the connection.

```
ltm rule /Common/deny_source_ip {  
    when CLIENT_ACCEPTED {  
        if {not [class match [IP::client_addr] equals deny_source_ip]  
    } } {  
        reject  
    }  
}
```

## Creating an IP Group to match the IP's in the Deny Source IP Rule

Navigate to Templates | IP Groups | Create IP Group

The screenshot shows the VMware NSX-ALB interface. The top navigation bar includes 'Applications', 'Operations', 'Templates' (selected), 'Infrastructure', and 'Administration'. The left sidebar shows 'Profiles', 'Policies', and 'Groups' (selected), with sub-items 'IP Groups' and 'String Groups'. The main area displays a table of IP Groups with columns for Name, IP Addresses or Ranges, and Country Codes or EPG. A 'CREATE IP GROUP' button is in the top right.

<input type="checkbox"/>	Name	IP Addresses or Ranges	Country Codes or EPG	
<input type="checkbox"/>	block_country_group	Country Code	CU, KP, SD, SY, IR	
<input type="checkbox"/>	denylist_group	1		
<input type="checkbox"/>	denylist_ips	7		

The screenshot shows the 'New IP Group: deny\_source\_ip' configuration page. The 'General' tab is selected. The 'Name' field is 'deny\_source\_ip'. The 'Type' is 'IP Address'. The 'Import IP Address From File' section has a 'Select a file' button and an 'IMPORT FILE' button. The 'IP Addresses (1)' section shows a list of IP addresses: '/32, 192.168.110.12/32, 192.168.110.13/32, 192.168.110.14/32, 192.168.110.15/32, 192.168.111.0/24'. A note below the list states 'Accepts Single, Comma Separated and Ranges'. The 'Items per page' is set to 10.

**General**

Name\*

Type

Import IP Address From File  
Select a file

IP Addresses (1)

<input type="checkbox"/>	IP Address	
<input type="checkbox"/>	/32, 192.168.110.12/32, 192.168.110.13/32, 192.168.110.14/32, 192.168.110.15/32, 192.168.111.0/24	

Accepts Single, Comma Separated and Ranges

Items per page 10



The IP Addresses can be imported from a CSV file:



192.168.110.10/32, 192.168.110.11/32, 192.168.110.12/32, 192.168.110.13/32,  
192.168.110.14/32, 192.168.110.15/32, 192.168.111.0/24, 172.16.0.0/16, 10.1.1.0/8

This IP Group can then be used in creating the Network Security policy next. Navigate to **Applications | Virtual Services** | click on the **Edit** (the pencil icon above) and select the **Network Security** tab:

## UI Configuration

The screenshot shows the VMware NSX UI configuration for a Network Security rule. The rule is named 'deny\_source\_ip' and is configured with the following settings:

- Rule Name:** deny\_source\_ip
- Logging:** Enabled (checked)
- Matching Rules:**
  - Client IP Address:**
    - Is not** (selected)
    - IP Address:** deny\_source\_ip
- Action:**
  - Deny** (selected)

The interface includes tabs for Settings, Policies, Analytics, and Advanced. The Policies tab is active, and the Network Security sub-tab is selected. The rule is currently in the 'Add Network Security Rule' state.

## YAML Code for Migration Utility

```

irule_custom_config:

- rule_name: deny_source_ip
  type: NetworkSecurityPolicy
  avi_config:
    name: nw_policy_rule
    rules:
      - index: 1
        enable: true
        name: "deny_source_ip"
        action: "NETWORK_SECURITY_POLICY_ACTION_TYPE_DENY"
        match:
          client_ip:
            match_criteria: IS_NOT_IN
            group_refs:
              - "/api/ipaddrgroup?name=deny_source_ip"
        log: true

```

As noted above, throughout this document there will be examples provided to assist in migrating from F5 Big IP to Avi from both the UI and from YAML code to use with conversion tool as shown in the Appendix. This syntax shows how to include the custom iRule replacement configuration with the conversion utility. Each example will include the iRule, the UI configuration workflow and many will include the YAML code to use with the conversion utility.

## Deny Based On Country Codes—block\_country\_group

There is a similar process when allowing or denying based on [ISO Country Codes](#). In the example below security policy may require blocking access to sensitive information from certain countries:

### iRule Syntax

```
ltm rule /Common/block_country_group {  
  when CLIENT_ACCEPTED {  
    switch [whereis [IP::client_addr] country] {  
      "CU" -  
      "KP" -  
      "SD" -  
      "SY" -  
      "IR"  
      {  
        reject  
      }  
    }  
  }  
}
```

## UI Workflow

Navigate to Templates | Groups | IP Group, click Create

IP GROUP  
block\_country\_group

New IP Group: block\_country\_group

Tenant  
admin

General

General

Name\* ⓘ  
block\_country\_group

Type  
Country Code

Country Code\* ⓘ  
Select Country Code

Search

- Afghanistan (AF)
- Albania (AL)
- Algeria (DZ)
- American Samoa (AS)
- Andorra (AD)
- Angola (AO)
- Anguilla (AI)

Search for the country codes to save the IP Group.

**IP GROUP**  
block\_country\_group

## New IP Group: block\_country\_group

General

### General

**Name\*** ⓘ  
block\_country\_group

**Type**  
Country Code

**Country Code\*** ⓘ  
Cuba (CU) X North Korea (KP) X Sudan (SD) X Syria (SY) X Iran (IR) X

Tenant admin

Add the group to a network security policy to replace the iRule above; edit the Virtual Service, click on the Policy tab, then Network Security, and the **green** + to add a rule.

Add the name for the rule and select the match, choose IP Address, the country code group will be in the dropdown menu (see below):

**Edit Virtual Service: iRule-Test-FE-VS**

Settings Policies Analytics Advanced

SELECT IP Group/Network Security Policy

Select Geo DB

• Add Network Security Rule •

Rule Name  
block\_country\_group

Logging

**Matching Rules**

Client IP Address

Is Is not

IP Address \*

1.1.1.1, 1.1.1.1-1.1.1.2, 1.1.1.0/24, 200e::1, 200e::1/6

Set Custom Value

Custom Value

Or pick one of the following String/IP Groups

access\_denylist\_group

block\_country\_group

denylist\_ips

Internal

New GB

Search

Create

Action

Allow Deny

Once the match is selected, choose the action, in this case deny. In the upper righthand corner, check the logging box to log the results of this policy. Due to resource consumption, the recommended way is enabling logging for troubleshooting and testing purposes only.

## UI Workflow

Edit Virtual Service: iRule-Test-FE-VS

Settings

Policies

Analytics

Advanced

IP Reputation DB ?

Select IP Reputation DB

Geo DB ?

Select Geo DB

• Add Network Security Rule •

Rule Name

block\_country\_group

☐ Logging

Matching Rules

Client IP Address

☒ Is ☐ Is not

IP Address \*

block\_country\_group

+ Add Item

Add New Match

Action

Action

☐ Allow ☒ Deny ☐ Rate Limit

Cancel

Save Rule

## YAML Code for Migration Utility

```

! irule_custom_config:
- rule_name: block_country_group
  type: NetworkSecurityPolicy
  avi_config:
    name: nw_policy_rule
    rules:
      - index: 1
        enable: true
        name: "block_country_group"
        age: 0
        action: "NETWORK_SECURITY_POLICY_ACTION_TYPE_DENY"
        match:
          client_ip:
            match_criteria: IS_IN
            group_refs:
              - "/api/ipaddrgroup?name=block_country_group"
        log: false

```

## HTTP Security Policy

The *HTTP Security Policy* executes second in priority; it matches on IP Addresses, TCP/UDP ports, Protocol Types, HTTP Method, Version, and Header and Host Header, as well as matching on Path, Query, Cookie, and IP Reputation.

The actions this policy can use are allowing or closing the connection, redirecting to HTTPS, Rate Limiting, sending an HTTP Response, or enabling *ICAP* (Internet Content Adaptation Protocol).

One use case that occurs with enterprise customers is:

### Match HTTP Path, Deny Connection:

iRule Syntax:

```
ltm rule /Common/path_reject {  
  when HTTP_REQUEST {  
    switch -glob [string tolower [HTTP::uri]] {  
      "/admin*" { reject }  
      default { return }  
    }  
  }  
}
```



In the UI, edit the VS and select the Policies | HTTP Security and add a rule:

The screenshot shows the VMware NSX Manager interface for configuring an HTTP Security rule. The top navigation bar includes 'Settings', 'Policies', 'Analytics', and 'Advanced'. Under 'Policies', 'HTTP Security' is selected, with sub-tabs for 'Network Security', 'HTTP Request', 'HTTP Response', 'DataScripts', and 'Access'. The 'IP Reputation DB' and 'Geo DB' dropdowns are set to 'Select IP Reputation DB' and 'Select Geo DB' respectively. A green banner at the top of the configuration area says '+ Add HTTP Security Rule +'. The 'Rule Name' field is set to 'path\_reject', with 'None' and 'Log' buttons next to it. The 'Matching Rules' section is expanded, showing a 'Path' match rule. The 'Criteria' dropdown is set to 'Contains', and the 'String group or custom string' dropdown is set to '/admin\*'. The 'Match Case' checkbox is checked. Below the matching rules, there is an 'Add New Match' button. The 'Action' section is set to 'Close Connection'. At the bottom, there are 'Cancel' and 'Save Rule' buttons.

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: path_reject
  type: HTTPPolicySet
  avi_config:
    name: path_reject
    http_security_policy:
      rules:
        - name: path_reject
          index: 1
          enable: true
          match:
            path:
              match_criteria: CONTAINS
              match_case: SENSITIVE
              match_str:
                - '/admin*'
          action:
            action: HTTP_SECURITY_ACTION_CLOSE_CONN
            log: true
```

## Match Path and respond with HTTP 404

Another path match use case is to respond with a 404 and log:

iRule Syntax:

```
ltm rule /Common/sys_404 {
  when HTTP_REQUEST {
    set my_uri [string tolower [HTTP::uri]]
    if { $my_uri starts_with "/sys" } {
      HTTP::respond 404
    }
  }
}
```

In the UI, this policy rule is configured in the same way, add a rule, then a match on the path:

The screenshot displays the VMware NSX Manager interface for configuring an HTTP Security Rule. The top navigation bar includes 'Settings', 'Policies', 'Analytics', and 'Advanced'. The 'Policies' tab is active, and the 'HTTP Security' sub-tab is selected. Below the navigation, there are dropdowns for 'IP Reputation DB' and 'Geo DB'. A green banner indicates '+ Add HTTP Security Rule +'. The rule configuration section shows the 'Rule Name' as 'sys\_404' with 'None' and 'Log' buttons. The 'Matching Rules' section is expanded, showing a 'Path' match with the criteria 'Begins with' and the string group '/sys'. The 'Match Case' checkbox is checked. Below this, there is an 'Add New Match' button. The 'Action' section is expanded, showing 'Send Response Settings' with radio buttons for status codes: 200, 204, 403, 404 (selected), 429, and 501. There is a 'Send Response' dropdown and a 'choose file' button with an 'Upload File' button. At the bottom, there are 'Cancel' and 'Save Rule' buttons.

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: sys_404
  type: HTTPPolicySet
  avi_config:
    name: sys_404
    http_security_policy:
      rules:
        - name: sys_404
          index: 1
          enable: true
          match:
            path:
              match_criteria: BEGINS_WITH
              match_case: SENSITIVE
              match_str:
                - "/sys"
          action:
            action: HTTP_SECURITY_ACTION_SEND_RESPONSE
            status_code: HTTP_LOCAL_RESPONSE_STATUS_CODE_404
          log: true
```

## Deny IP Group and respond with HTTP 200 message

Another use case that occurs frequently is denying access based on an IP Group; this iRule explains why the user is unable to access content using custom HTML Code:

### iRule Syntax:

```
ltm rule /Common/Match_IP_Group_200_OK {
  when HTTP_REQUEST {
#    log local0. "Received connection from [IP::client_addr]"
    if {not ([class match [IP::client_addr] equals IP_Group])}{
      log local0. "irule denied connection from [IP::client_addr]"
      HTTP::respond 200 content "
        <html>
          <head>
            <title>Apology Page</title>
          </head>
          <body>
            We are sorry, but your IP of [IP::client_addr] is not permitted. <br>
            Please contact the site administrator if you feel this is an error.
          </body>
        </html>"
    }
  }
}
```

Using the UI, the rule is configured with a match on and *IP Group* (see Network Security for details on creating an IP group); the action is to send a HTTP 200 response and calls a file to inform the user the reason for denying access.

## UI Workflow

The screenshot shows the VMware NSX Manager UI for configuring an HTTP Security Rule. The interface includes tabs for Settings, Policies, Analytics, and Advanced. Under Policies, there are sub-tabs for Network Security, HTTP Security (selected), HTTP Request, HTTP Response, DataScripts, and Access. The rule is named 'Match\_IP\_Group\_200\_OK'. The matching rule is 'Client IP Address' with the condition 'Is not' and the value 'IP\_Group'. The action is 'Send Response' with status code '200' and a file named 'file attached'.

The file attached to the policy can be a HTML page or a text file. In this case we have added a text file with the message:

```
We are sorry, but your IP address is not permitted.
Please contact the site administrator if you feel this is an
error.
```

## YAML Code for Migration Utility

```

irule_custom_config:
- rule_name: Match_IP_Group_200_OK
  type: HTTPPolicySet
  avi_config:
    name: Match_IP_Group_200_OK
    http_security_policy:
      rules:
        - name: Match_IP_Group_200_OK
          index: 1
          enable: true
          match:
            client_ip:
              match_criteria: IS_NOT_IN
              group_refs:
                - "/api/ipaddrgroup?name=IP_Group"
          action:
            action: HTTP_SECURITY_ACTION_SEND_RESPONSE
            status_code: HTTP_LOCAL_RESPONSE_STATUS_CODE_200
            file:
              content_type: text/rtf
              file_content: |-
                {We are sorry, but your IP address is not permitted. Please contact
your administrator if you feel this is an error.}
            log: true

```

## Match HTTP path, Client IP Group and Close Connection

iRule Syntax:

```
ltm rule /Common/Drop_IP_Path {
  when HTTP_REQUEST {
    set my_uri [string tolower [HTTP::uri]]
    # block external access to http://www.test.com/test-internal
    # and http://www.test.com/test-internal
    if { ($my_uri contains "/test-internal") and (not ([class match
[IP::client_addr] equals Internal])) } {
      #log local0. "Received connection from [IP::client_addr]"
      drop
      #log local0. "Dropped connection from [IP::client_addr]"
    }
  }
}
```

## UI Configuration

From the UI, add the new rule with the path match and the IP address not matching the approved IP Group ( Internal ) ; add the action to close the connection.

The screenshot displays the VMware NSX Manager interface for configuring an HTTP Security Rule. The 'Policies' tab is active, and the 'HTTP Security' sub-tab is selected. The rule is named 'Drop\_IP\_Path' and is currently in 'None' state. The configuration is as follows:

- Client IP Address:** The rule is configured to match IP addresses that are **Is not** **Internal**.
- Path:** The rule is configured to match paths that **Contains** the string **/test-internal**. The 'Match Case' checkbox is checked.
- Action:** The action is set to **Close Connection**.

Buttons for 'Cancel' and 'Save Rule' are visible at the bottom of the configuration panel.

## YAML Code for Migration Utility

```
irule_custom_config:

- rule_name: Drop_IP_Path
  type: HTTPPolicySet
  avi_config:
    name: Drop_IP_Path
    http_security_policy:
      rules:
        - name: Drop_IP_Path
          index: 1
          enable: true
          match:
            client_ip:
              match_criteria: IS_NOT_IN
              group_refs:
                - "/api/ipaddrgroup?name=Internal"
            path:
              match_criteria: CONTAINS
              match_case: INSENSITIVE
              match_str:
                - "/test-internal"
          action:
            action: HTTP_SECURITY_ACTION_CLOSE_CONN
          log: true
```



## HTTP Request Policy

*HTTP Request Policy* executes third in priority and matches on the criteria below, unlike HTTP Security; HTTP Request Policy does not close connections as part of its actions.

Match Criteria-

- Client IP (individually or in an IP group, as well as country codes and reputation)
- Service Port
- Protocol Type (HTTP/HTTPS)
- HTTP Method (CONNECT, COPY, DELETE, GET, HEAD, LOCK, MKCOL, MOVE, OPTIONS, PATCH, POST, PROPFIND, PRODPATCH, PUT, TRACE)
- HTTP Version (0.9, 1.0, 1.1, 2.0)
- Path
- Query
- Headers
- Cookie
- Host Header

Its actions include content switching, redirection, rewrite, and modify (add, remove, replace) headers.

The most frequent use case is using the HTTP Request Policy to modify headers:

Add HTTP Header

iRule Syntax:

```
ltm rule /Common/httpsterminated_add {
  when HTTP_REQUEST {
    if { ![HTTP::header exists "httpsterminated"]} {
      HTTP::header insert "httpsterminated" "true"
    }
  }
}
```

## UI Workflow

To add this rule, go to Virtual Service | Policies | HTTP Request, click the green + to add the new rule:

Add the name `httpsterminated_add`, match will be if the header does not exist "httpsterminated" and action will be to insert header `httpsterminated`, and the value "true."

## UI Configuration

**Edit Virtual Service: SaaS-test**

Settings Policies Analytics Advanced

Network Security HTTP Security **HTTP Request** HTTP Response DataScripts Access

IP Reputation DB   
 Select IP Reputation DB

**• Add HTTP Request Rule •**

Rule Name\*   
 `httpsterminated_add` None Log Log with headers

**Matching Rules**

**Header** ✕

Criteria\* Does not exist ▼ Name\* httpsterminated

Add New Match ▼

**Action**

**Modify Header** ✕

Action\* Add header ▼ Name\* httpsterminated Value\* Custom Value ▼ Custom Value\* true

Add Action

Add New Action ▼

Cancel Save Rule

Click on Save Rule and click on Save to save it to the VS.

## YAML Code for Migration Utility

```
irule_custom_config:

- rule_name: httpsterminated_add
  type: HTTPPolicySet
  avi_config:
    name: httpsterminated_add
    http_request_policy:
      rules:
        - name: httpsterminated_add
          index: 1
          enable: true
          match:
            hdrs:
              - match_criteria: HDR_DOES_NOT_EXIST
                hdr: httpsterminated
          hdr_action:
            - action: HTTP_ADD_HDR
              hdr:
                name: httpsterminated
                value:
                  val: 'true'
          log: false
```

## Insert SSL Information in the HTTP Header

Another header modification use case is inserting SSL information into the header:

### iRule Syntax

```
ltm rule /Common/add_ssl_data {
  when HTTP_REQUEST {
    HTTP::header insert "SSL_PROTOCOL" [SSL::cipher version]
    HTTP::header insert "SSL_CIPHER" [SSL::cipher name]
  }
}
```

In creating iRules, F5 defines SSL Cipher Version as the current SSL cipher version using the format of the [OpenSSL SSL\\_CIPHER\\_get\\_version\(\) function](#) (e.g. "SSLv2", "SSLv3", "TLSv1", "TLSv1.1", "TLSv1.2").

F5 also defines SSL Cipher Name as the current SSL cipher name using the format of the [OpenSSL SSL\\_CIPHER\\_get\\_name\(\) function](#) (e.g. "EDH-RSA-DES-CBC3-SHA" or "RC4-MD5").

Go to the Policies tab, and select HTTP Request then add a new rule called add\_ssl\_data. The iRule above matches on any client, so we will not add a match, however, we will add two actions.

### UI Workflow

The screenshot shows the F5 iRule configuration interface. The 'Policies' tab is selected, and the 'HTTP Request' policy is chosen. A new rule named 'add\_ssl\_data' is being created. The 'Rule Name' field is filled with 'add\_ssl\_data'. The 'Matching Rules' section is empty. The 'Action' section is titled 'Modify Header' and contains two actions: 'Add header' for 'SSL\_Protocol' and 'Add header' for 'SSL\_Cipher'. The 'Value' field for each action is set to the corresponding header name. The 'Add Action' button is visible at the bottom of the 'Modify Header' section. The 'Cancel' and 'Save Rule' buttons are at the bottom of the page.

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: add_ssl_data
  type: HTTPPolicySet
  avi_config:
    name: add_ssl_data
    http_request_policy:
      rules:
        - name: add_ssl_data
          index: 1
          enable: true
          hdr_action:
            - action: HTTP_ADD_HDR
              hdr:
                name: SSL_Protocol
                value:
                  var: HTTP_POLICY_VAR_SSL_PROTOCOL
            - action: HTTP_ADD_HDR
              hdr:
                name: SSL_Cipher
                value:
                  var: HTTP_POLICY_VAR_SSL_CIPHER
          log: true
          all_headers: true
```

## Content Switching

Another use case in HTTP Policy is the ability to *content switch*, which would allow users to match on various criteria and switch content accordingly.

Avi's Content Switching has the ability to match on a variety of criteria's like Client IP, Service Port, Protocol Type, Path, Query, Headers etc. For more information please go to this link:

(<https://avinetworks.com/docs/latest/http-request-policy/>)

In this use case, the iRule matches IP addresses and chooses the pool accordingly.

```
ltm rule /Common/demo_80_irule {  
  when CLIENT_ACCEPTED {  
    if { [IP::addr [IP::remote_addr] equals 10.1.138.37] } {  
      pool demo_app01_80  
    }  
    if { [IP::addr [IP::remote_addr] equals 10.1.138.38] } {  
      pool demo_app02_80  
    }  
  }  
}
```

This iRule switches to two separate pools when matching separate IP addresses so users can take advantage of the ability to add a second rule to the HTTP Request Policy. When necessary, users can continue adding rules to this policy, each with separate matches and actions.

## UI Workflow

Navigate to the Virtual Service | Policies | HTTP Request, click green + to add the new rule. Then add the first rule, matching on a specific IP Address (users can match on any of the HTTP Request match criteria), then add the action of Content Switching and assign the pool or other content switching action.

Since this use case includes two matches, the process needs to be repeated. (See <https://avinetworks.com/docs/latest/architectural-overview/applications/vs-policies/>).

Edit Virtual Service: demo\_app

Settings
Policies
Analytics
Advanced

Network Security
HTTP Security
HTTP Request
HTTP Response
DataScripts
Access

IP Reputation DB ⓘ  
Select IP Reputation DB

+

Q

INDEX	ENABLE	NAME	MATCHING RULES	ACTION	
1	<input checked="" type="checkbox"/>	demo_80_irule_match_1	Client IP is in (10.1.138.37)	Content Switch Pool: demo_app01_80	<div> <div></div> <div>...</div> <div></div> </div>
2	<input checked="" type="checkbox"/>	demo_80_irule_match_2	Client IP is in (10.1.138.38)	Content Switch Pool: demo_app02_80	<div> <div></div> <div>...</div> <div></div> </div>

Cancel
Save

## YAML Code for Migration Utility

```

irule_custom_config:
- rule_name: demo_80_irule
  type: HTTPPolicySet
  avi_config:
    name: demo_80_irule
    http_request_policy:
      rules:
        - name: demo_80_irule _match_1
          index: 1
          enable: true
          match:
            client_ip:
              match_criteria: IS_IN
              addrs:
                - addr: 10.1.138.37
                  type: V4
          switching_action:
            action: HTTP_SWITCHING_SELECT_POOL
            status_code: HTTP_LOCAL_RESPONSE_STATUS_CODE_200
            pool_ref: /api/pool?name=demo_app01_80
        - name: demo_80_irule _match_2
          index: 2
          enable: true
          match:
            client_ip:
              match_criteria: IS_IN
              addrs:
                - addr: 10.1.138.38
                  type: V4
          switching_action:
            action: HTTP_SWITCHING_SELECT_POOL
            status_code: HTTP_LOCAL_RESPONSE_STATUS_CODE_200
            pool_ref: /api/pool?name=demo_app02_80
      log: false

```



## HTTP Redirection by matching the HTTP Path

Another use case in HTTP Policy is the ability to match the HTTP path and redirect.

### iRule Syntax :

```
ltm rule /Common/Redirection {
    when HTTP_REQUEST { if { [HTTP::path] equals "/" } {
        HTTP::redirect "https://test.email.com/owa" } else { HTTP::redirect
        "https://test.email.com[HTTP::uri]" } }
}
```

## UI Workflow

Navigate to the Virtual Service | Policies | HTTP Request, click **green +** to add the new rule. This rule will match the path in the HTTP Request and redirect to the specific path defined in the action. Then add a second rule, matching on any path and redirecting to a specific path (see <https://avinetworks.com/docs/latest/architectural-overview/applications/vs-policies/>).

Settings Policies Analytics Advanced				
Network Security HTTP Security <b>HTTP Request</b> HTTP Response DataScripts Access				
IP Reputation DB ⓘ Select IP Reputation DB ▼				
<div> <div>+</div> <div>Q</div> </div>				
INDEX	ENABLE	NAME	MATCHING RULES	ACTION
1	✓	Redirection	Path equals (/)	Redirect https://test.email.com:443/owa?<KeepQueryString>
2	✓	Redirection_else	Any	Redirect https://test.email.com:443/<ExistingPath>?<KeepQueryString>

## YAML Code for Migration Utility

```

- rule_name: Redirection
  type: HTTPPolicySet
  avi_config:
    name: Redirection
    http_request_policy:
      rules:
        - name: Redirection
          index: 1
          enable: true
          match:
            path:
              match_criteria: EQUALS
              match_case: INSENSITIVE
              match_str:
                - "/"
          redirect_action:
            protocol: HTTPS
            host:
              type: URI_PARAM_TYPE_TOKENIZED
              tokens:
                - type: URI_TOKEN_TYPE_STRING
                  str_value: test.email.com
            port: 443
            path:
              type: URI_PARAM_TYPE_TOKENIZED
              tokens:
                - type: URI_TOKEN_TYPE_STRING
                  str_value: owa
            keep_query: true
            status_code: HTTP_REDIRECT_STATUS_CODE_302
        - name: Redirection_else
          index: 2
          enable: true
          redirect_action:
            protocol: HTTPS
            host:
              type: URI_PARAM_TYPE_TOKENIZED
              tokens:
                - type: URI_TOKEN_TYPE_STRING
                  str_value: test.email.com
            port: 443
            keep_query: true
            status_code: HTTP_REDIRECT_STATUS_CODE_302
          is_internal_policy: false

```

## HTTP Response Policy

The match criteria for the HTTP Response Policy are the same as the Request policy, including Client IP, Service Port, Protocol Type, HTTP Method and Version, Path, Query, Headers, and Host Header.

However, the Response Policy is limited to the following actions, Redirect, Modify Header, Rewrite URL, or Content Switch.

The first requirement is to add an HTTP Header:

### Add HTTP Header during HTTP Response

iRule Syntax:

```
ltm rule /Common/Response_Header_Add {
  when HTTP_RESPONSE {
    HTTP::header insert "X-Test-Tag" "noindex"
  }
}
```

In the UI, leave the match blank (Match Any) and add a custom header, **X-Test-Tag** with a value of **noindex**.

The screenshot displays the VMware NSX Manager interface for configuring an HTTP Response Rule. The top navigation bar includes tabs for Settings, Policies, Analytics, and Advanced. Under the Policies tab, the sub-tabs are Network Security, HTTP Security, HTTP Request, HTTP Response (selected), DataScripts, and Access.

The main configuration area is titled "Add HTTP Response Rule". It includes a "Rule Name" field with the value "Response\_Header\_Add" and buttons for "None", "Log", and "Log with headers".

Below the rule name, there are two sections: "Matching Rules" and "Action".

The "Matching Rules" section has a dropdown menu labeled "Add New Match".

The "Action" section is titled "Modify Header" and contains the following fields:

- Action:** A dropdown menu with "Add header" selected.
- Name:** A text field with "X-Test-Tag" entered.
- Value:** A dropdown menu with "Custom Value" selected.
- Custom Value:** A text field with "noindex" entered.

At the bottom of the "Action" section, there is a button labeled "Add New Action".

The bottom of the interface features a "Cancel" button on the left and a "Save Rule" button on the right.

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: Response_Header_Add
  type: HTTPPolicySet
  avi_config:
    name: Response_Header_Add
    http_response_policy:
      rules:
        - name: Response_Header_Add
          index: 1
          enable: true
          hdr_action:
            - action: HTTP_ADD_HDR
              hdr:
                name: X-Test-Tag
                value:
                  val: noindex
          log: true
          all_headers: true
```

## Insert Origin Information in HTTP Header during Response

```
ltm rule /Common/Header_Insert_Response {
  when HTTP_RESPONSE {
    HTTP::header insert origin test.origin.com
  }
}
```

### UI Workflow

Again, navigate to the Virtual Service | Policies | HTTP Response, click **green +** to add the new rule. HTTP Response matches on the same criteria as HTTP Request; however, the actions differ. Here the matching condition is any, so the match rule is not included.

Action will be to add a HTTP Header, available options are Add, Remove, or Replace. Provide a name and custom value, there are drop-down lists of other values

(See <https://avinetworks.com/docs/latest/architectural-overview/applications/vs-policies/>).

The screenshot displays the Avinetworks configuration interface for adding an HTTP Response rule. The top navigation bar includes 'Settings', 'Policies', 'Analytics', and 'Advanced'. The 'Policies' section is active, showing 'Network Security', 'HTTP Security', 'HTTP Request', 'HTTP Response' (selected), 'DataScripts', and 'Access'. A green '+ Add HTTP Response Rule +' button is visible. Below this, the 'Rule Name' field is set to 'Header\_Insert\_Response'. There are buttons for 'None', 'Log', and 'Log with headers'. The 'Matching Rules' section is empty, with an 'Add New Match' button. The 'Action' section is titled 'Modify Header' and contains a table with the following configuration:

Action *	Name *	Value *	Custom Value *
Add header	origin	Custom Value	test.origin.com

Below the table is an 'Add Action' button and an 'Add New Action' button. At the bottom, there are 'Cancel' and 'Save Rule' buttons.

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: Header_Insert_Response
  type: HTTPPolicySet
  avi_config:
    name: Header_Insert_Response
    http_response_policy:
      rules:
        - name: Header_Insert_Response
          index: 1
          enable: true
          hdr_action:
            - action: HTTP_ADD_HDR
              hdr:
                name: origin
                value:
                  val: test.origin.com
          log: true
          all_headers: true
  is_internal_policy: false
```

## DataScripts

DataScripts are comprised of any number of function or method calls which can be used to inspect and act on traffic flowing through a virtual service. DataScript's functions are exposed via Lua libraries and grouped into modules: *string*, *vs*, *http*, *pool*, *ssl* and *crypto*.

(See <https://avinetworks.com/docs/latest/datascript-functions/>)

### DS Library Link

<https://github.com/vmware/datascript-library>

### True Client IP Replacement

#### iRule Syntax

```
ltm rule /Common/True_Client_IP_Replacement {
when HTTP_REQUEST {
    if { [HTTP::header exists "True-Client-IP"] } {
        HTTP::header replace "X-Forwarded-For" "[HTTP::header
"True-Client-IP"]"
    }
    elseif { [HTTP::header exists "X-Forwarded-For"] } {
        HTTP::header replace "X-Forwarded-For"
"[IP::client_addr]"
    }
    else {
        HTTP::header insert "X-Forwarded-For"
"[IP::client_addr]"
    }
}
}
```

### DataScript Library Link

[https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/true\\_client\\_ip\\_replacement.md](https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/true_client_ip_replacement.md)

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: True_Client_IP_Replacment
  type: VSDataScriptSet
  avi_config:
    name: True_Client_IP_Replacement
    datascript:
      - evt: VS_DATASCRIPPT_EVT_HTTP_REQ
        script: |-
          if avi.http.get_header("True-Client-IP") then
            avi.http.replace_header("X-Forwarded-For", "True-Client-IP")
          else avi.http.replace_header("X-Forwarded-For", avi.vs.client_ip())
          end
```

## Insert Pool Member IP in HTTP Header

### iRule Syntax

```
ltm rule /Common/Add_Pool_Member_IP {
  when HTTP_RESPONSE {
    # Insert pool member IP in a header named Pool-Info
    HTTP::header insert Pool-Info [LB::server addr]
  }
}
```

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: Add_Pool_Member_IP
  type: VSDataScriptSet
  avi_config:
    name: Add_Pool_Member_IP
    datascript:
      - evt: VS_DATASCRIPPT_EVT_HTTP_RESP
        script: avi.http.add_header("Pool-Info", avi.pool.server_ip())
```



## DataScript Library Link

[https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/add\\_pool\\_server\\_ip.md](https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/add_pool_server_ip.md)

## Add Cookie Domain Attribute

## iRule Syntax

```
ltm rule /Common/cookie_domain {
  when HTTP_RESPONSE {
    # Check if the persistence cookie exists in the response
    if {[HTTP::cookie exists "persist_cookie"]} {
      # set the domain attribute on the persistence cookie
      HTTP::cookie domain "persist_cookie" "test.com"
    }
  }
}
```

## DataScript Library Link

[https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/update\\_cookie\\_domain.md](https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/update_cookie_domain.md)

## YAML Code for Migration Utility

```
irule_custom_config:
- rule_name: cookie_domain
  type: VSDataScriptSet
  avi_config:
    name: cookie_domain
    datascript:
      - evt: VS_DATASCRIPPT_EVT_HTTP_RESP
        script: |-
          cookie_name = "persist_cookie"
          if avi.http.cookie_exists(cookie_name) then
            avi.http.update_cookie(cookie_name, "domain", "test.com")
          end
```

## Cookie Rewrite

### iRule Syntax

```
ltm rule /Common/Cookie_Rewrite {  
  when HTTP_REQUEST {  
    if { [HTTP::cookie exists domain] } {  
      set cookie_value_sanitized [string map {"example.com"  
"example.int"} [HTTP::cookie value domain]]  
  
      # Change cookie domain's value  
      HTTP::cookie value domain $cookie_value_sanitized  
    }  
  }  
  
  when HTTP_RESPONSE {  
    if { [HTTP::cookie exists domain] } {  
      set cookie_value_sanitized [string map {"example.int"  
"example.com"} [HTTP::cookie value domain]]  
  
      # Change cookie domain's value  
      HTTP::cookie value domain $cookie_value_sanitized  
    }  
  }  
}
```

**DataScript Library Link:**

[https://github.com/vmware/datascript-library/blob/master/security/cookie\\_rewrite.md](https://github.com/vmware/datascript-library/blob/master/security/cookie_rewrite.md)

**YAML Code for Migration Utility**

```

irule_custom_config:

- rule_name: Cookie_Rewrite
  type: VSDataScriptSet
  avi_config:
    name: Cookie_Rewrite
    datascript:
      - evt: VS_DATASCRIPRT_EVT_HTTP_REQ
        script: |-
          if avi.http.cookie_exists("domain") then
            cookie_value = avi.http.get_cookie("domain")
            if cookie_value:contains("example.com") then
              cookie_value_sanitized = cookie_value:gsub("example.com", "example.int")
              cookie_table = {domain=cookie_value_sanitized, sanitized=true}
              avi.http.replace_cookie(cookie_table)
            end
          end
      - evt: VS_DATASCRIPRT_EVT_HTTP_RESP
        script: |-
          if avi.http.cookie_exists("domain") then
            cookie_value = avi.http.get_cookie("domain")
            if cookie_value:contains("example.com") then
              cookie_value_sanitized = cookie_value:gsub("example.int", "example.com")
              cookie_table = {domain=cookie_value_sanitized, sanitized=true}
              avi.http.replace_cookie(cookie_table)
            end
          end

```

## TLS Version and IP Group Match

### iRule Syntax

```
ltm rule /Common/tls_version_ip_group {
when HTTP_REQUEST {
#log local0. "[SSL::cipher version] and client [client_addr]"
  if { ([SSL::cipher version] equals "TLSv1") && (not([class match
[IP::client_addr] equals IP_Group])) }{
    log local0. "dropped [SSL::cipher version] for [http_host]
[http_uri] source-ip [client_addr] header [HTTP::header "User-
Agent"]"
    drop
  }
  elseif { ([SSL::cipher version] equals "TLSv1.1") && (not([class
match [IP::client_addr] equals IP_Group])) }{
    log local0. "dropped [SSL::cipher version] for [http_host]
[http_uri] source-ip [client_addr] header [HTTP::header "User-
Agent"]"
    drop
  }
}
```

### DataScript Library Link

[https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/tls\\_version\\_ip\\_group.md](https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/security/tls_version_ip_group.md)

## YAML Code for Migration Utility

```

irule_custom_config:

- rule_name: tls_version_ip_group
  type: VSDataScriptSet
  avi_config:
    name: tls_version_ip_group
    datascript:
      - evt: VS_DATASCRIPTEVT_HTTP_REQ
        script: |-
          var=avi.vs.client_ip()
          ua = avi.http.get_header("user-agent")
          ip_group=avi.ipgroup.contains("IP_Group", var)
          if avi.ssl.protocol() == "TLSv1.0" or "TLSv1.1" and
          avi.ipgroup.contains("IP_Group", var) == false then
            avi.http.close_conn()
          end
    ipgroup_refs:
      - /api/ipaddrgroup?name=IP_Group

```

## JSession ID Persistence

### iRule Syntax

```
ltm rule /Common/JsessionID_Persistence {
when HTTP_RESPONSE {
if { [HTTP::cookie exists "JSESSIONID"] } {
persist add uie [HTTP::cookie "JSESSIONID"]
}
}
}
when HTTP_REQUEST {
if { [HTTP::cookie exists "JSESSIONID"] } {
persist uie [HTTP::cookie "JSESSIONID"]
} else {
set jsess [findstr [HTTP::uri] "jsessionid" 11 ";"]
if { $jsess != "" } {
persist uie $jsess
}
}
}
```

### DataScript Library Link

[https://github.com/vmware/datascript-library/blob/master/availability/jsessionid\\_persistence.md](https://github.com/vmware/datascript-library/blob/master/availability/jsessionid_persistence.md)

### YAML Code for Migration Utility

```
irule_custom_config:

- rule_name: JsessionID_Persistence
  type: VSDataScriptSet
  avi_config:
    name: JsessionID_Persistence
    datascript:
      - evt: VS_DATASCRIPPT_EVT_HTTP_REQ
        script: |-
          if avi.http.get_cookie("JSESSIONID") then

avi.vs.table_insert(avi.http.get_cookie("JSESSIONID"),avi.vs.pool.server_ip())
          end
      - evt: VS_DATASCRIPPT_EVT_HTTP_RESP
        script: |-
          if avi.http.get_cookie("JSESSIONID") then

avi.vs.table_insert(avi.http.get_cookie("JSESSIONID"),avi.vs.pool.server_ip())
          end
```

## TLS1.2\_Version Check and Respond with HTML Page

### iRule Syntax

```
ltm rule /Common/TLS1.2_version_check_with_response {
when HTTP_REQUEST {
  if { [SSL::cipher version] ne "TLSv1.2" } {
    switch [string tolower [HTTP::uri]] { "/" } {HTTP::respond 200
content [ifile get browser.html] "Content-Type" "text/html"}
  }
}
```

### DataScript Library Link

[https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/availability/tls\\_version\\_1.2\\_with\\_html\\_response.md](https://github.com/vmware/nsx-alb-datascript-samples-library/blob/master/availability/tls_version_1.2_with_html_response.md)

### YAML Code for Migration Utility

```
irule_custom_config:

- rule_name: TLS1.2_version_check_with_response
  type: VSDataScriptSet
  avi_config:
    name: TLS1.2_verion_check_with_response
    datascript:
      - evt: VS_DATASCRIPPT_EVT_HTTP_REQ
        script: |-
          page = "<html><body><p><center><b>Your request cannot be completed since
you are not using TLSv1.2. Please contact your administrator.</b></p>"
          if avi.ssl.protocol() ~= "TLSv1.2" and string.lower(avi.http.get_path())
== "/" then
            avi.http.response(200, {content_type="text/html"}, page)
          end
```

## Virtual Service Settings

### X-Forwarded-For

The most common and recommended way to add X-Forwarded-For to a Virtual Service is within the Application Template. While an iRule can be based on HTTP Request and DataScripts, an application profile can accomplish the same result.

While these settings and templates can replace the need for iRules, you can't implement these settings using the conversion tool and a custom\_configuration.yml file, but it can still be configured using “avi\_application” module in the playbook. These modifications will be added after the migration is complete.

Below are some of the iRules for x-forwarded-for which can be accomplished using the application profile template.

```
ltm rule /Common/Forward_Client_IP {
  when HTTP_REQUEST {
    HTTP::header insert X-Forwarded-For [IP::remote_addr]
  }
}
```

```
ltm rule /Common/x-forwarded-for {
  priority 200
  when HTTP_REQUEST {
    if { HTTP::header exists "X-Forwarded-For" } {
      #log local0. "X-Forwarded-For exists"
      HTTP::header replace "X-Forwarded-For"
      "[IP::client_addr]"
      #log local0. "Replacing X-Forwarded to [IP::client_addr]"
    } else {
      #log local0. "Inserting X-forwarded into the HTTP Header"
      HTTP::header insert "X-Forwarded-For"
      "[IP::client_addr]"
    }
  }
}
```



This is accomplished through editing an existing application profile or creating a new application profile.

The screenshot shows the configuration page for an application profile named "System-HTTP". The "General" tab is active, and the "Type" is set to "HTTP". The "Description" field is empty. Under the "HTTP Settings" section, several options are checked: "Connection Multiplex", "Detect NTLM App", "X-Forwarded-For" (highlighted with a red box), "WebSockets Proxy", "Proxy Expect Header", and "Enable Chunk Merge". The "X-Forwarded-For" section shows "XFF Alternate Name" set to "X-Forwarded-For". The "Role-Based Access Control (RBAC)" section is empty, showing a message "We couldn't find any objects!". At the bottom, there are "Cancel" and "Save" buttons.

As it is visible from the image of the UI, users can enable True Client IP and can Preserve Client IP Address. To quote NSX Advanced Load Balancer documentation: “Using the true client IP feature enables fetching the actual client IP address from “X-Forwarded-For” or from a user-defined header and track the actual client IP address into logs or configure policies such as HTTP Security, HTTP Request etc. based on the true client IP address.”

(see <https://avinetworks.com/docs/latest/true-client-ip-in-L7-security-features/>)

## YAML Code for Post Migration Runbook

```

---
- connection: local
  hosts: localhost
  gather_facts: false
  vars:
    controller: 10.10.10.10
    username: admin
    password: xxxx
    api_version: 21.1.1
  collections:
    - vmware.alb
  tasks:
    - avi_applicationprofile:
        description: Add x-forwarded-for
        controller: '{{ controller }}'
        http_profile:
          client_max_header_size: 48.0
          connection_multiplexing_enabled: true
          secure_cookie_enabled: false
          x_forwarded_proto_enabled: false
          xff_alternate_name: X-Forwarded-For
          xff_enabled: true
        name: System-HTTP
        password: '{{ password }}'
        state: '{{ state | default(omit) }}'
        tenant: admin
        tenant_ref: /api/tenant/?name=admin
        type: APPLICATION_PROFILE_TYPE_HTTP
        username: '{{ username }}'
        name: 'Create or Update ApplicationProfile: System-HTTP'

```

## HTTP To HTTPS Redirection

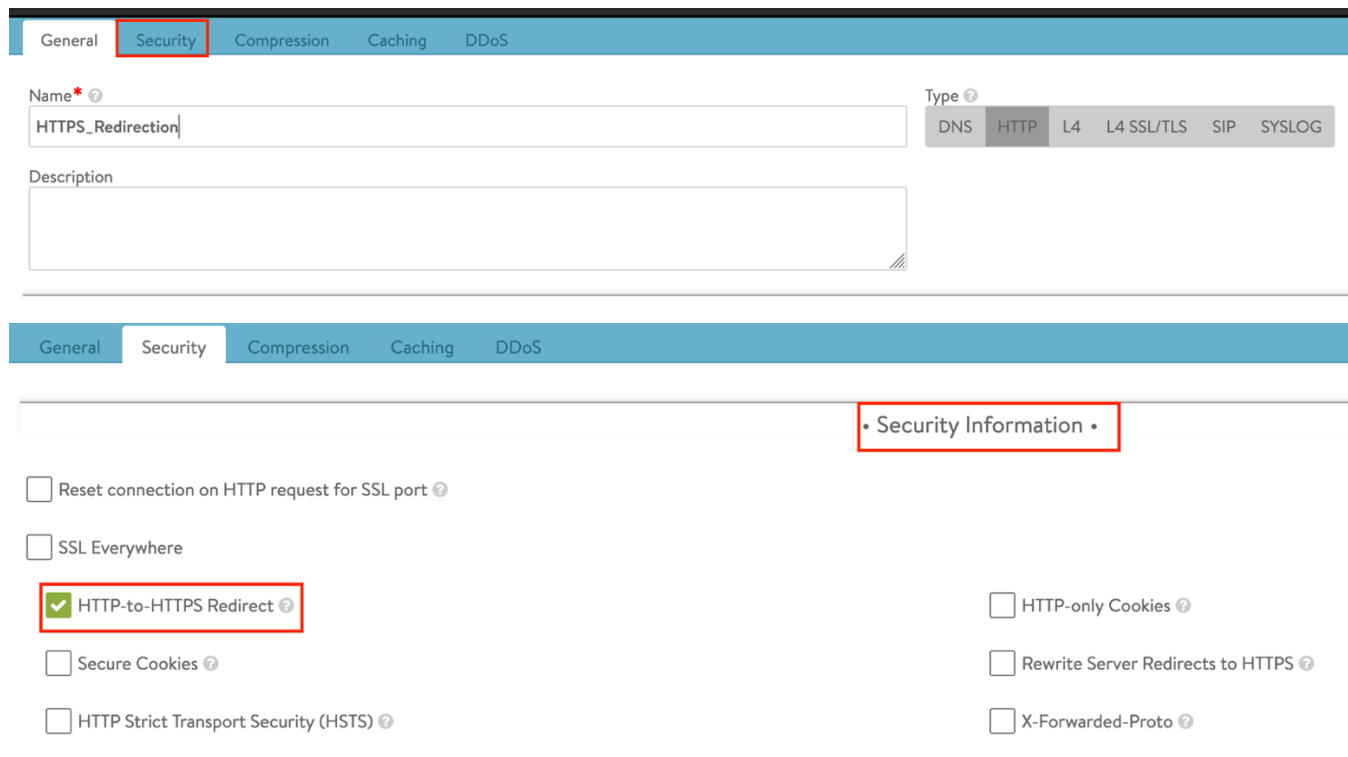
The most common iRule is http to https redirection. Although it can be achieved via HTTP Policies and DataScripts, recommended way is via Application Profiles Security Settings.

While these settings and templates can replace the need for iRules, these settings can't be implemented using the conversion tool and a custom\_configuration.yml file, but it can still be configured using “avi\_application” module in the playbook. These modifications will be added after the migration is complete.

iRule Syntax:

```
}
ltm rule /Common/HTTPS_Redirection {
when HTTP_REQUEST {HTTP::redirect https://[HTTP::host][HTTP::uri]}
}
```

## UI Configuration:



The screenshot shows the VMware vSphere configuration interface for an iRule. The 'Security' tab is selected, and the 'HTTP-to-HTTPS Redirect' checkbox is checked. The 'Security Information' section is highlighted with a red box.

**General** **Security** Compression Caching DDoS

Name\*  Type ☐ DNS ☒ HTTP ☐ L4 ☐ L4 SSL/TLS ☐ SIP ☐ SYSLOG

Description

---

**General** **Security** Compression Caching DDoS

**• Security Information •**

☐ Reset connection on HTTP request for SSL port ?

☐ SSL Everywhere

☒ HTTP-to-HTTPS Redirect ?

☐ Secure Cookies ?

☐ HTTP Strict Transport Security (HSTS) ?

☐ HTTP-only Cookies ?

☐ Rewrite Server Redirects to HTTPS ?

☐ X-Forwarded-Proto ?

## YAML Code for Post Migration Runbook

```

---
- connection: local
  hosts: localhost
  gather_facts: false
  vars:
    controller: 10.10.10.10
    username: admin
    password: xxxx
    api_version: 21.1.1
  collections:
    - vmware.alb
  tasks:
    - avi_applicationprofile:
        controller: '{{ controller }}'
        description: null
        http_profile:
          client_max_header_size: 48.0
          connection_multiplexing_enabled: true
          http_to_https: true
        name: HTTPS_Redirection
        password: '{{ password }}'
        state: '{{ state | default(omit) }}'
        tenant: admin
        tenant_ref: /api/tenant/?name=admin
        type: APPLICATION_PROFILE_TYPE_HTTP
        username: '{{ username }}'
        name: 'Create or Update ApplicationProfile: HTTPS_Redirection'

```

## Maintenance Page

Another frequent use case is the ability to notify users if the pool servers fail:

```
ltm rule /Common/Maintenance_Page_Rule
{
when HTTP_REQUEST {
if { [active_members [LB::server pool]] == 0 } {
HTTP::redirect "https://maintenance.avinetworks.com/" }
}
}
```

On the NSX Advanced Load Balancer, this is a pool setting; edit the pool and go to the Advanced tab. Pool Failure settings provide three options—Close Connection, HTTP Local Response (call an HTML file), or HTTP Redirect. In this use case it is being redirected to another URL. Follow up with saving the changes.

## UI Configuration

The screenshot shows the 'Edit Pool: SaaS-test-pool' interface with the 'Advanced' tab selected. The 'Pool Full Settings' section includes 'Request Queuing' (Disabled) and 'Queue Length' (128). The 'Pool Failure Settings' section is configured with 'Fail Action' set to 'HTTP Redirect', 'Status Code' set to '302', and 'URL' set to 'https://maintenance.avinetworks.com/'. The 'Placement Settings' section shows 'Server Network' with a '+ Add Server Network' link. The 'Connection Pool Settings' section is empty. At the bottom, there are 'Cancel' and 'Save' buttons.

## YAML Code for Post Migration Runbook

```

---
- connection: local
  hosts: localhost
  gather_facts: false
  vars:
    controller: 10.10.10.10
    username: admin
    password: xxxxxx
    api_version: 21.1.1
  collections:
    - vmware.alb
  tasks:
    - avi_pool:
        cloud_ref: /api/cloud?name=Default-Cloud
        controller: '{{ controller }}'
        description: null
        fail_action:
          type: FAIL_ACTION_HTTP_REDIRECT
          redirect:
            protocol: HTTPS
            host: maintenance.avinetworks.com
            path: "/"
            status_code: HTTP_REDIRECT_STATUS_CODE_302
        health_monitor_refs:
          - /api/healthmonitor/?name=System-HTTPS
        lb_algorithm: LB_ALGORITHM_LEAST_LOAD
        name: SaaS-test-pool
        password: '{{ password }}'
        servers:
          - description: ''
            enabled: true
            hostname: 192.168.142.34
            ip:
              addr: 192.168.142.34
              type: V4
              port: '443'
            ssl_key_and_certificate_ref:
              /api/sslkeyandcertificate/?name=serversssl-auto_created
            ssl_profile_ref: /api/sslprofile/?name=serversssl
            state: '{{ state | default(omit) }}'
            tenant: admin
            username: '{{ username }}'
            tenant_ref: /api/tenant/?name=admin
            vrf_ref: /api/vrfcontext/?name=global
        name: 'Create or Update Pool: SaaS-test-pool'

```

## Logging Settings

There are several iRule use cases for sending logging from the load balancer to an external logging system. The Advanced Load Balancer has the capability to export logs as well. This is an Analytics Profile setting. Either the existing profile associated with the VS can be modified or a new Analytics profile can be created to export logs to an external server.

(See <https://avinetworks.com/docs/latest/streaming-avi-logs-to-external-server/>).

## UI Configuration

New Analytics Profile:

Name \*

remote-logging-template

Gather Analytics for

Virtual Services x Servers x Service Engines x

Description

• HTTP Analytics •

Client Response Apdex ?

Satisfactory Latency Threshold \*

500

ms

Tolerated Latency Factor

4X

Server Response Apdex ?

Satisfactory Latency Threshold \*

400

ms

Tolerated Latency Factor

4X

Client PageLoad Apdex ?

Satisfactory Latency Threshold \*

5000

ms

Tolerated Latency Factor

4X

Exclude HTTP Status codes from Error Classification ?

400, 401-408, 5XX

• Network Analytics •

Client Connection Apdex - Lossy Connection Threshold

TCP Retransmit Threshold \*

50

%

Server Connection Apdex - Lossy Connection Threshold

TCP Retransmit Threshold \*

50

%

Cancel

Save

In addition, the Advanced Load Balancer can integrate with Splunk for logging and dashboards (<https://splunkbase.splunk.com/app/4155/> and ). For how to information on configuring this solution, see <https://www.youtube.com/watch?v=SEv0IUokUWU>.

New Analytics Profile: ✕

• Client Log Processing •

☒ Enable Significant Logs ⓘ

Significant Log Processing Type ⓘ

Sync And Index On Demand

Filtered Log Processing Type ⓘ

Sync And Index On Demand

Non-significant Log Processing Type ⓘ

Sync And Index On Demand

• Client Log Streaming •

☒ Stream Logs to an External Server

Default Port ⓘ

514

Types of Logs to Stream ⓘ

All logs

Max Logs ⓘ

100

/sec

Log Streaming Protocol

UDP

Server ⓘ \*

10.206.40.243

Port

514

[+ Add Server](#)

• Sensitive Log Profile •

+

Q

INDEX	ENABLE	NAME	MATCHING RULE	ACTION
No items found.				

Cancel

Save



## Appendix

### Helpful details

When migrating iRules to an Advanced Load Balancer controller users can take advantage of one of the properties of the `f5_converter.py` script. Re-run the conversion utility using the

`--custom_config` flag and refer to an `irule_custom_config.yml` file with the proper syntax to add policies and DataScripts to the Advanced Load Balancer configuration (see sample below).

NOTE : The Playbooks used in the Virtual Service settings section cannot be used with the `custom_config` flag with the migration tool. as it is only applicable for Policies and Datascripts conversion.

```
f5_convertor.py -f migration_bigip.conf --no_object_merge --
not_in_use --vrf global --tenant prod --controller_version
21.1.3 --cloud_name Default-Cloud --segroup Default-Group
ansible - o production_migration --custom_config
converted_irules.yml --reuse_http_policy --vs_level_status
```

## Glossary

**Avi Controller (NSX Advanced Load Balancer):** Acts as the brain behind the services delivered by the Data Plane, it represents a central point of management, configuration (GUI, CLI, REST API) and control for the system. Repository for licensing, logs, and metrics. In production three nodes are a best practice for high availability; the controller cluster manages analytics for the continuous stream of application data to decide on service placement (placing Virtual Services on Service Engines), autoscaling, and high availability for each application. NSX Advanced Load Balancer controllers operate in the Control Plane from where they manage each of the deployed Service Engines (SEs). It uses big data analytics to present actionable insights to assist administrators on intuitive dashboards on the Admin Console (UI).

### Clouds:

Clouds are containers for the environment that NSX ALB is installed or operating within. During initial setup, a default cloud, named “Default-Cloud”, is created. This is where the first Controller is deployed, into Default-Cloud. Additional clouds may be added, containing SEs and virtual services.

### Pulse:

Pulse is a cloud services platform that is built to deliver value add services to distributed NSX ALB Controller cluster deployments in a SaaS like model; provides services to simplify customer’s operations and enable friction-less support and services. Requires Portal account user approval.

### Service Engine:

In the context of the NSX Advanced Load Balancers, Service Engines (SEs) handle all of the data plane operations within Avi Vantage. SEs host the virtual services and require either direct or routable access to all client and server networks connecting to a virtual service. A typical Advanced Load Balancer deployment may have many SEs for various purposes, such as redundancy, scalability, and accommodating large numbers of applications being served. SEs are always grouped within the context of a SE group, which provides settings for high availability, scalability, and potentially resource isolation for tenants.

### Service Engine Group:

Service Engines are created within an SE Group, which contains the definition of how the SEs should be sized, placed, and made highly available. Each cloud will have at least one SE group, and multiple SE groups may exist within a cloud. Provides Data Plane isolation when different applications are mapped to different SE Groups. The options within an SE group may vary based on the type of cloud within which they exist and its settings, such as no access versus write access mode. A Virtual Service can be placed on a specific SE Group using the Advanced tab when editing the Virtual Service. To move an existing virtual service from one SE group to another, the VS must first be disabled, moved, and then re-enabled (this is disruptive to the service hosted).





**VMware, Inc.** 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 [vmware.com](http://vmware.com) Copyright © 2020 VMware, Inc.  
All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at [vmware.com/go/patents](http://vmware.com/go/patents). VMware is a registered trademark or trademark of VMware, Inc. and its subsidiaries in the United States and other jurisdictions.  
All other marks and names mentioned herein may be trademarks of their respective companies. Item No: vmw-pguide-temp-uslet-word-101-proof 6/20