

August 2018



# Introduction to Automation & Orchestration

PRESENTED BY:

**Shain Singh**

**APCJ Security Architect**

# Background



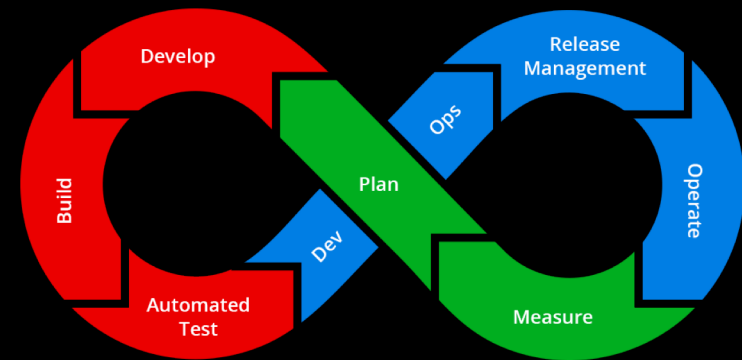
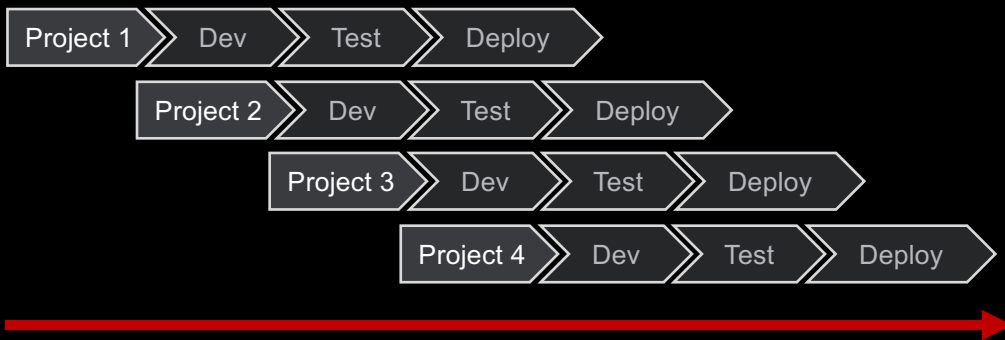
# Super-NetOps

The Super-NetOps training program from F5 is **free, on-demand training** (complete with labs) designed for Network Operations professionals that want to learn more about the **automation and orchestration features available in F5 products** and how to apply them in accordance with DevOps practices. It is intended to help Network Operations and DevOps professionals work more closely together to leverage the unique skillsets of each group.



# Traditional vs Modern Architecture

- Traditional
- Project Oriented
- Monolithic (risky) rollouts
- DevOps (CI/CD)
- System Oriented
- Continuous Improvement



# Agenda

## Introduction & Concepts

- Automation and Orchestration Concepts
- Introduction to REST

## Imperative Automation via the BIG-IP REST API

- Lab Module 1

## Abstracting Services using the App Services 3 Extension

- Lab Module 2

## Creating Declarative Service Interfaces

- Lab Module 3

# Automation & Orchestration Concepts



# Cost effective automation requires appropriate Abstraction

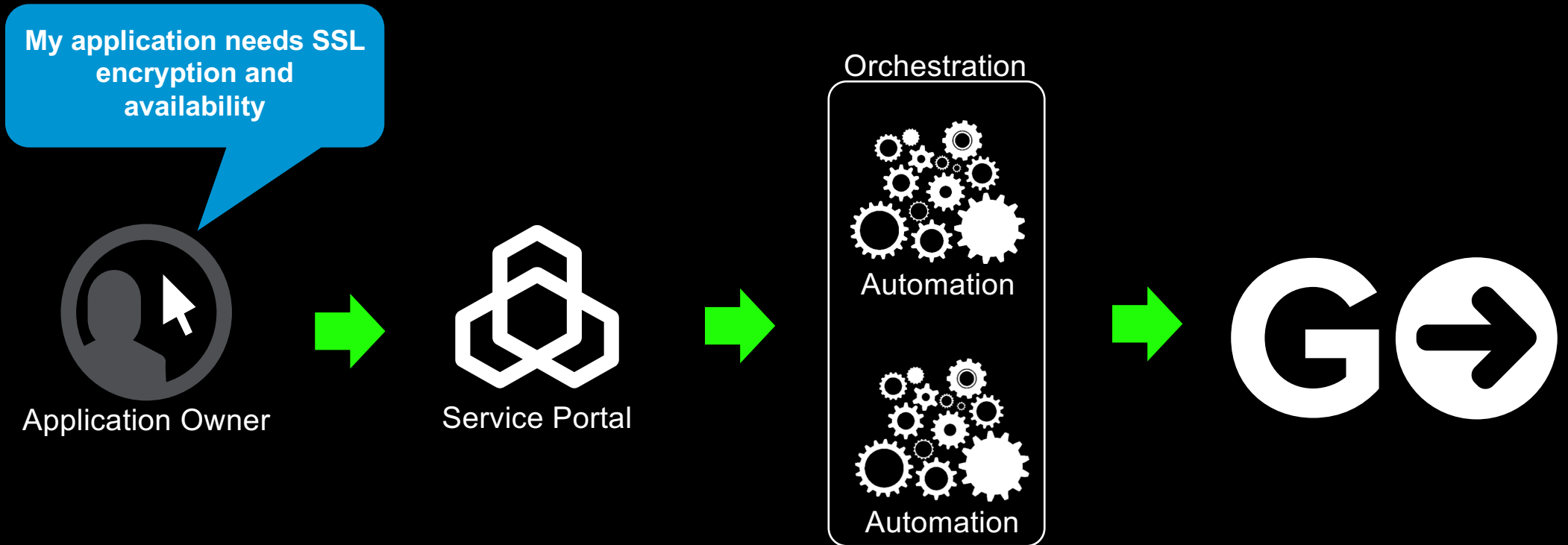


# Cost effective automation requires appropriate Abstraction

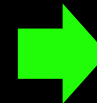
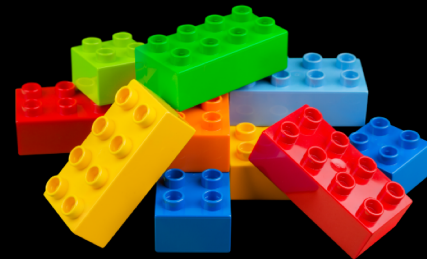
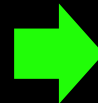
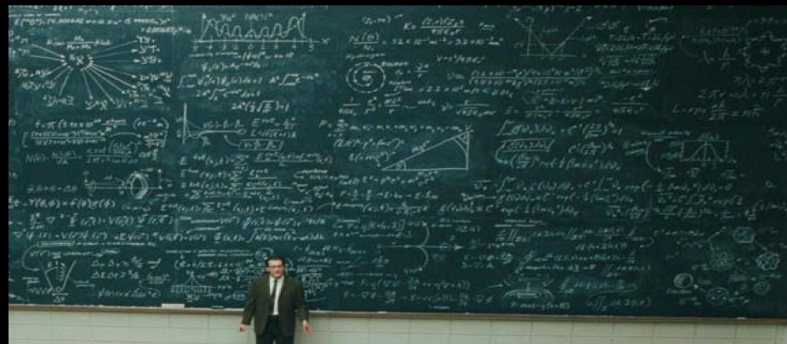
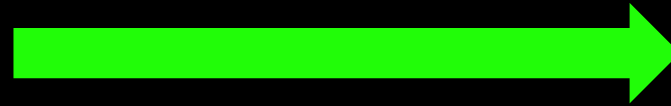
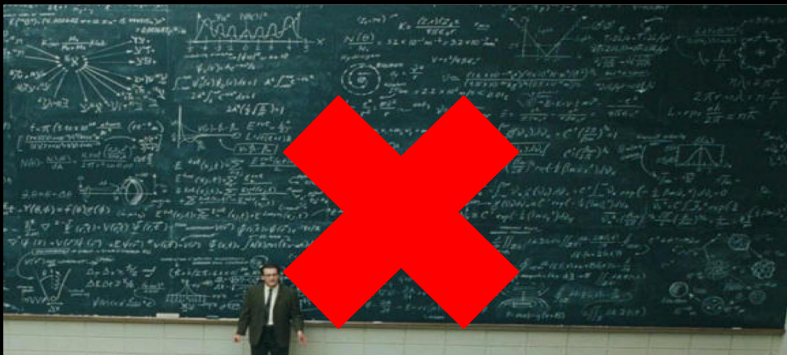




# Appropriate abstraction enables Declarative Interfaces

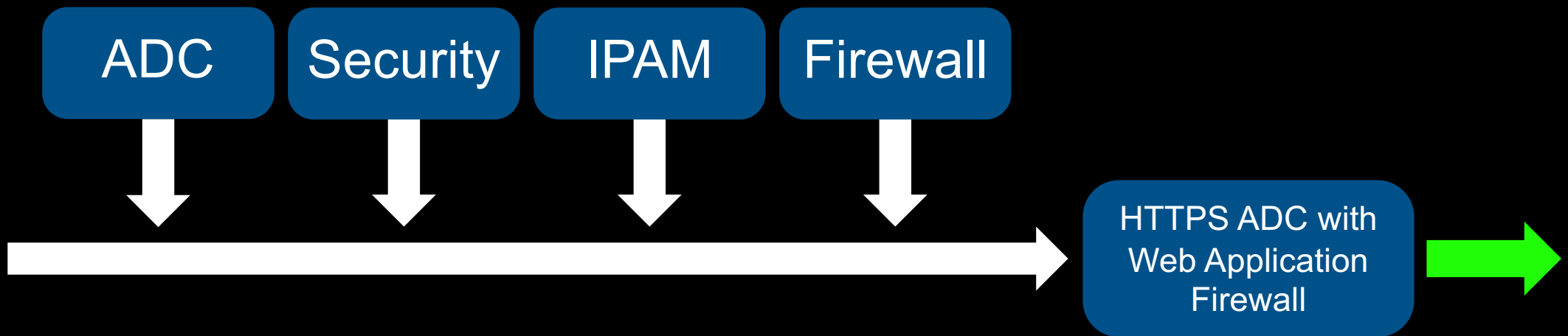


# Declarative Interfaces reduce or eliminate Domain Specific Knowledge



**Reduced requirement for Domain Specific  
Knowledge enables effective collaboration  
between Super-NetOps and DevOps**

# Lower domain specific knowledge enables simplified DevOps Orchestration

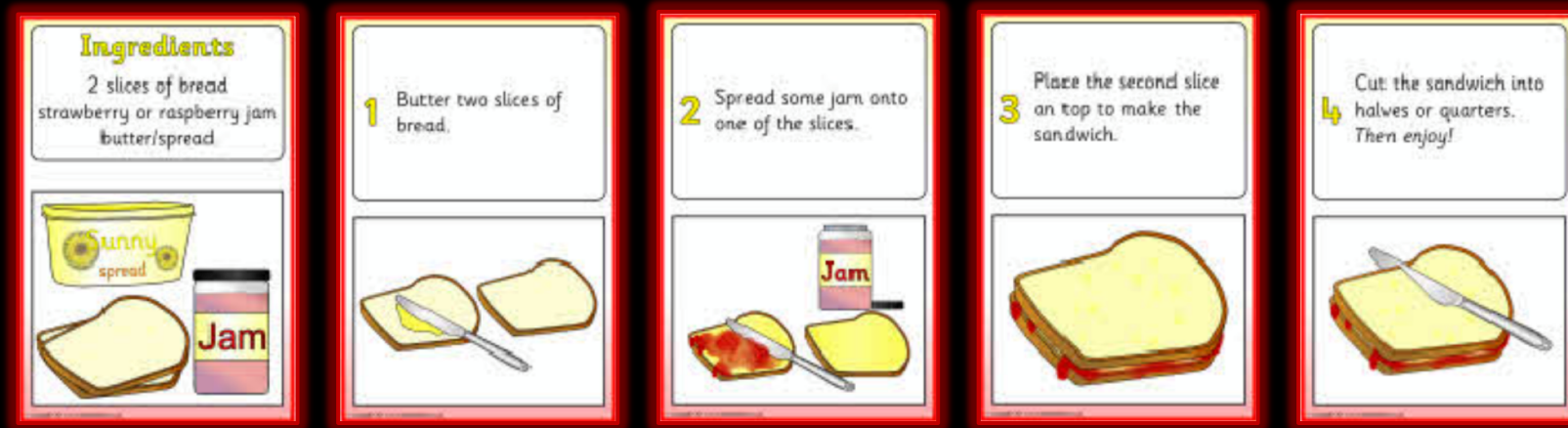


**Orchestration enables an  
effective DevOps culture**



# Imperative Model

Imperative – What we've done for years (scripting, iRules, etc.) Imperative methodology implies that you define the flow of an operation implicitly. It also implies that domain-specific knowledge is required to interact with the system.



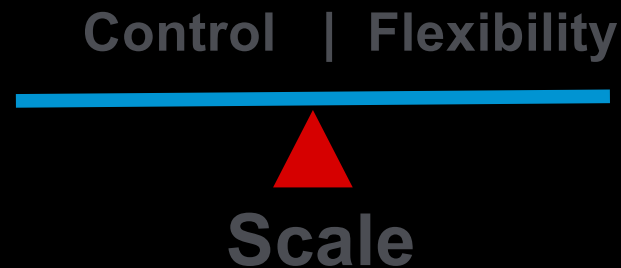
What domain-specific knowledge is required to make this sandwich?

# Declarative Model

Declarative – What we're evolving to. Declarative methodology implies that you define the desired outcome and depend on underlying mechanisms to deliver that outcome. This methodology tries to reduce or eliminate the need for domain specific knowledge.



# Practical Example - Role Based Access Control



- **Imperative Methodology** – Implement Access Control on every device in the environment to protect the system from outages that could occur due to misconfiguration
- **Declarative Methodology** – Eliminates the need for device level policies because the Declarative Interface itself provides the Consumer of a service with a set of safe attributes they can modify; and nothing else



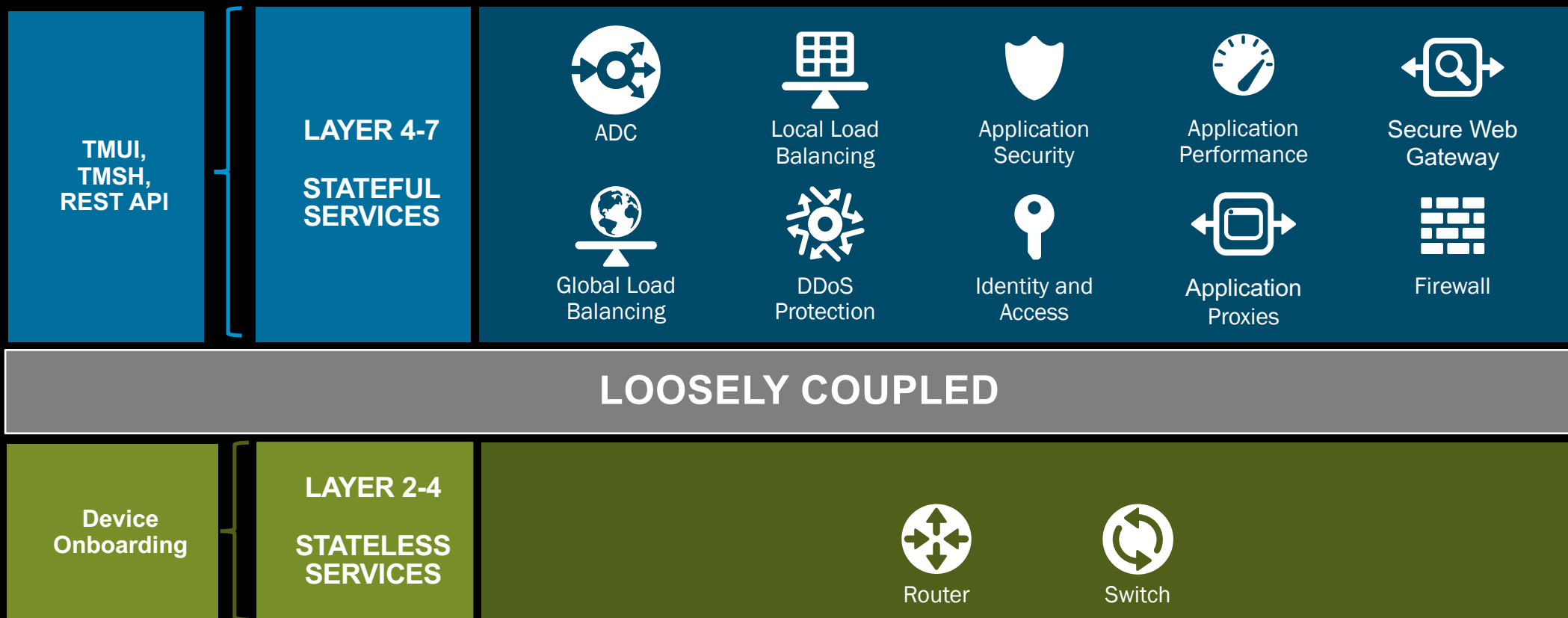
# Concepts – Multi-Tenancy

- BIG-IP is Multi-Tenant inherently.
  - Route-Domains
  - Partitions
  - What about route-domain 0?
- 
- All automation code should be multi-tenant aware by nature.
  - It's not that hard... always use a partition for object names.
  - Always use a route domain on all L3 addresses (including 0).

# Concepts – Source of Truth(iness)

- Source of Truth is defined as a system or object that contains the authoritative representation of a service.
- Changes for a service should propagate (push) from the source of truth to sub-ordinate systems.
- Out-of-band changes must be handled very carefully (or be totally avoided).
- To address real-world challenges we prefer to call this ‘Source-of-Truthiness’:
  - In an automation tool chain it’s possible to create layers of ‘truth’ as long as sub-ordinate systems ‘filter’ the service definition upstream.
  - What the northbound system doesn’t know won’t hurt it.
  - Changes should NEVER have to flow from sub-ordinate devices upstream!
- iApp strict updates are an example of strict Source-of-Truth enforcement.
- GitHub repositories are an example of a Source-of-Truthiness.

# Concepts – F5 Automation Anatomy



# Introduction to REST



# Introduction to REST

- Based on HTTP and JSON
- Uses HTTP verbs (GET, POST, PUT, PATCH, DELETE)
- Data is sent using the Javascript Object Notation format
- ```
{  
  "attribute1": "value1",  
  "attribute2": ["array", "of", "values"],  
  "attribute3": [ { "nested1": "value1", "nested2": "value2" }, { "nested3": "value3" } ]  
}
```

# REST APIs and HTTP Verbs

- The following table summarizes the interpretation of HTTP methods for REST APIs.
- HTTP methods (verbs) are used to create, read, update, and delete (CRUD) resources.
- APIs must use HTTP verbs in a manner described in the table below.
- APIs may implement a subset as required.

| URI               | POST                                          | GET                                                    | PUT                                         | DELETE                                | PATCH                                           |
|-------------------|-----------------------------------------------|--------------------------------------------------------|---------------------------------------------|---------------------------------------|-------------------------------------------------|
| <b>Collection</b> | Create resources.                             | Get representation of all resources in the collection. | Fully update all resources in a collection. | Delete all resources in a collection. | Partially update all resources in a collection. |
| <b>Resource</b>   | Used for non-idempotent controller resources. | Get a resource's representation.                       | Fully update the resource if it exists.     | Delete a resource.                    | Partially update a resource.                    |

# REST API Organization

| Type                    | Description                                                                                                  |
|-------------------------|--------------------------------------------------------------------------------------------------------------|
| Organizing Collection   | Objects are not configurable, rather they contains other Collections or Resources.                           |
| Collection              | Objects are not configurable, however, a <b>Collection</b> contains Resources of the same type.              |
| Resource                | A fully configurable object that supports create, update, refresh, delete, load, exists (CURDLE) operations. |
| Sub-Collection          | A collection that is attached to a particular Resource. Must be accessed through the 'parent' Resource.      |
| Sub-Collection Resource | Same as a Resource except it must be accessed via the sub-collection.                                        |

# Response Codes

- APIs must make use of HTTP response codes where appropriate.
- The following table describes the required success response codes.

| Response Code | Applicable Verbs                                                                                      | Notes                                                                                                                |
|---------------|-------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| 200 OK        | <ul style="list-style-type: none"><li>• All</li></ul>                                                 | Return on most positive responses including DELETE.                                                                  |
| 201 Created   | <ul style="list-style-type: none"><li>• POST</li></ul>                                                | HTTP Location header contains link to newly created resource.                                                        |
| 202 Accepted  | <ul style="list-style-type: none"><li>• POST</li><li>• PUT</li><li>• PATCH</li><li>• DELETE</li></ul> | Return when a request will take a long time; server should return a Location header for client to get state updates. |
| 404           | <ul style="list-style-type: none"><li>• GET</li></ul>                                                 | The resource does not exist.                                                                                         |
| 500           | <ul style="list-style-type: none"><li>• All</li></ul>                                                 | Check <code>/var/log/restjavad.0.log</code>                                                                          |



# Anatomy of a REST URI

```
tmsl list ltm pool /Common/mypool members {/Common/m1:80}
```



**NOTE: Resource names map '~' to '/' (e.g. ~Common~mypool is really /Common/mypool)**

# How the REST API is Implemented on TMOS



- REST API attributes are derived from TMSH schema. (Odata Standard)
- Generally, if the attribute/option is available in TMSH it's available in REST
- You can 'follow' the API by mapping to tmsh commands in most cases.
  - `list ltm pool pool1 members {10.1.20.1:80}`
  - GET: `https://10.1.1.10/mgmt/tm/ltm/pool/pool1/members/~Common~10.1.20.1:80`

# REST Query Parameters

- `?$filter=<partition>`
- Allows an OData search string to filter objects (e.g `?$filter eq MyPartition`)
- `?$select=attribute1,attribute2`
- Select specific attributes to return in the response
- `?expandSubcollections=true`
- Automatically expand any Sub-Collections in a resource
- `?ver=x.y.z`
- Use a specific version of the API schema
  
- See **REST API** guide for more (11.6, 12.X, and 13.X)

# REST Chicken and the Egg

- Problem: I want to create something but can't figure out how REST wants it represented.
- Solution: GET the Resource Example template to see how it's represented.
- GET [https://{{bigip\\_mgmt}}/mgmt/tm/ltm/pool/example](https://{{bigip_mgmt}}/mgmt/tm/ltm/pool/example)
- [https://{{bigip\\_mgmt}}/mgmt/toc](https://{{bigip_mgmt}}/mgmt/toc)


**Lessons from the road...**



# JSON Formatting

- JSON is a strictly defined syntax. As a result it is very unforgiving of syntactical errors.
- When defining a JSON object the last element in the object must NOT have a comma:

```
1 {  
2   "username": "",  
3   "password": "",  
4   "loginProviderName": "tmos",  
5 }
```



```
1 {  
2   "code": 400  
3   "message": "com.google.gson.stream.MalformedJsonException: Expected name at line 5 column 2 path $.loginProviderName",  
4   "originalRequestbody": {  
5     "username": "",  
6     "password": "",  
7     "loginProviderName": "tmos",  
8   }  
9   "referer": "192.168.255.1",  
10  "restOperationId": 297725,  
11  "kind": ":resterrorresponse"
```

# JSON Formatting – When you get stuck

- <http://jsonlint.com> -- Verify Syntax, Pretty Print Ugly JSON Responses
- Atom.io - jsonlint & Prettify

**JSONLint**  
The JSON Validator

[Try PRO →](#)

```
1 {  
2   "username": "",  
3   "password": "",  
4   "loginProviderName": "tmos",  
5 }
```

**Results**

```
Error: Parse error on line 4:  
...viderName": "tmos",}  
-----^  
Expecting 'STRING', got '}'
```



# Useful Logs

- `tail -f /var/log/restjavad.0.log | python -m json.tool`
- `/var/log/ltn`
- `/var/log/audit`



# The Sledge Hammer Approach

- The REST API is not complete (not all modules implement REST API endpoints)
- TMSH is not complete (loading APM policy from CLI, etc.)

• When all else fails, BASH it:

• POST to /mgmt/tm/util/bash

• JSON:

```
{  
  "command": "run",  
  "utilCmdArgs": "-c 'rm -Rf /'"  
}
```

Ex.

```
"utilCmdArgs": "-c 'mkdir -r /var/config/rest/downloads/mydir'"
```

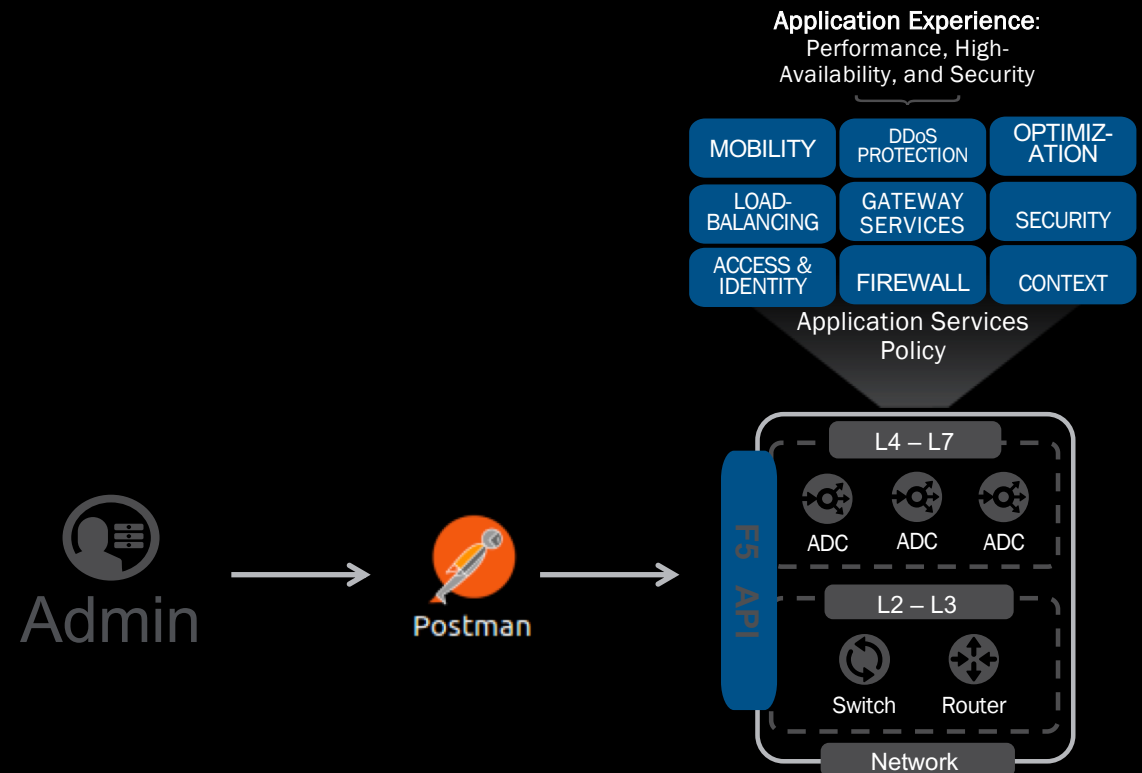
- This is a **ABSOLUTE** last resort, but useful nonetheless

# Putting it all together



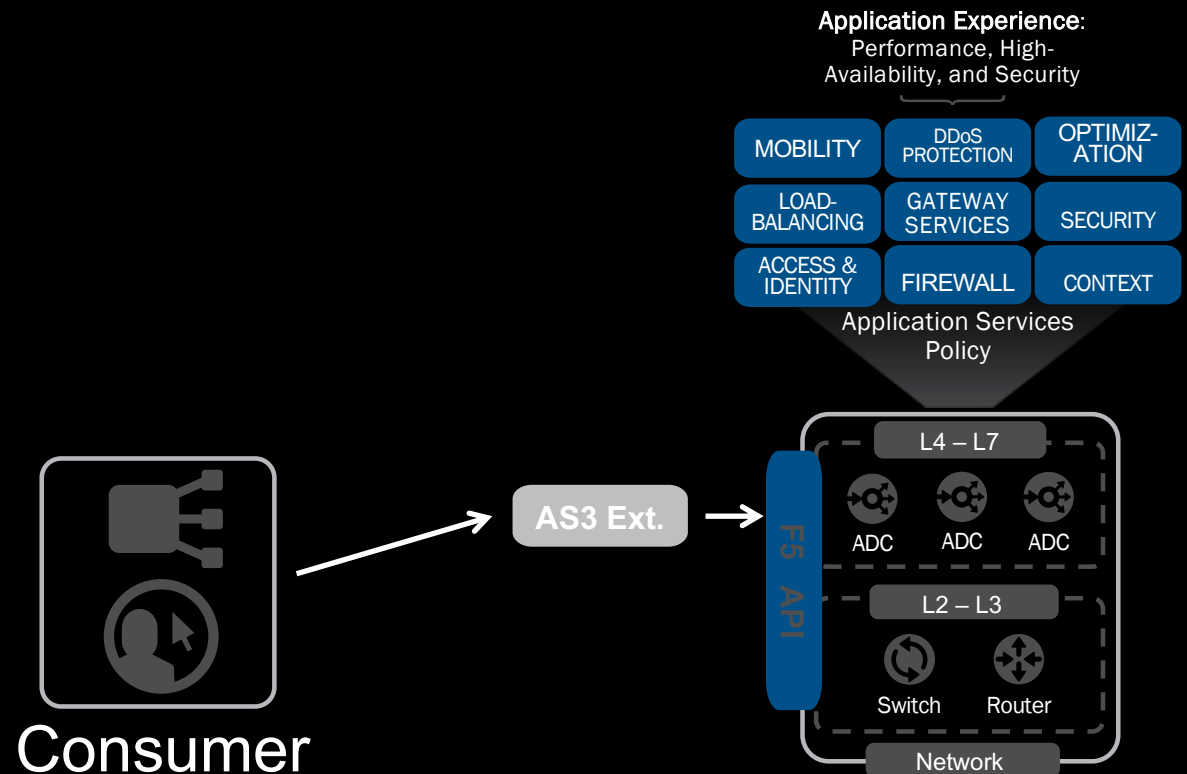
# REST API – Direct Automation

- Device Onboarding
- Operational Workflow
- **Imperative** Interface

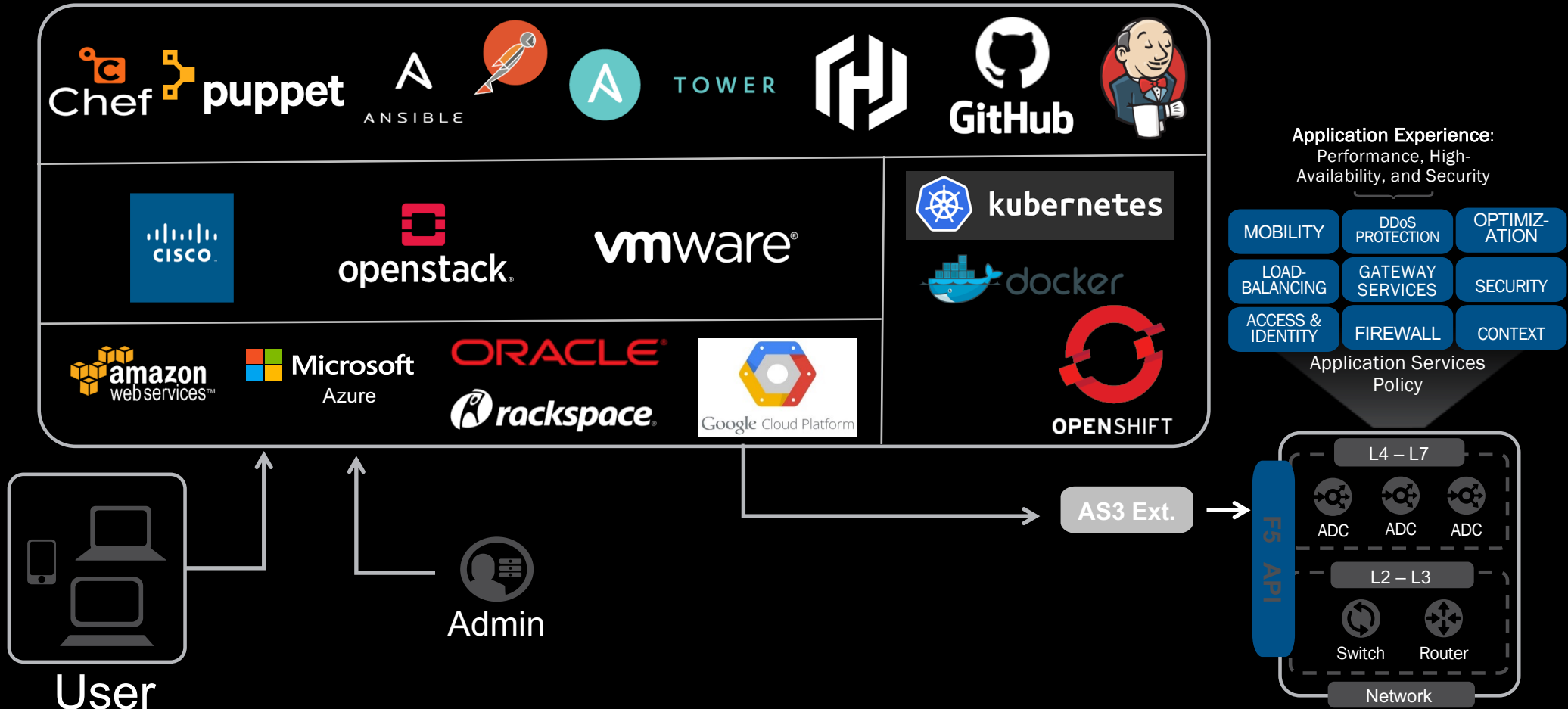


# App Services 3 (AS3) Extension – Implement Abstraction

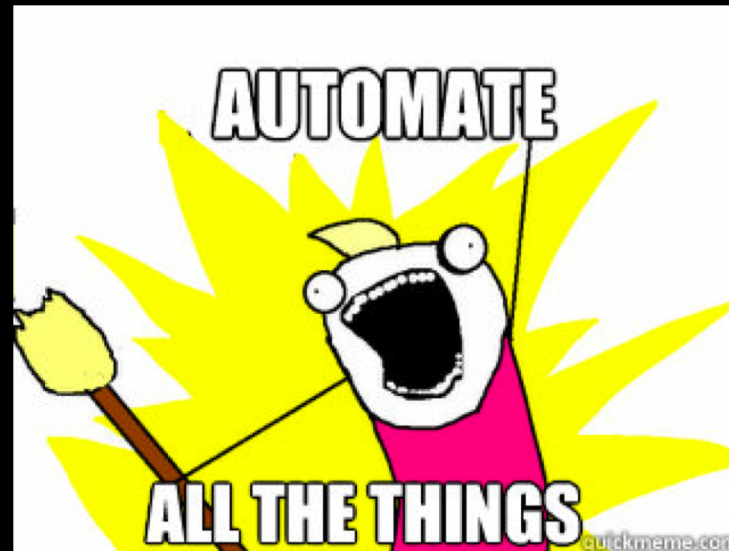
- L4-7 **Service** Abstraction



# Declarative Orchestration/DevOps



Now lets...



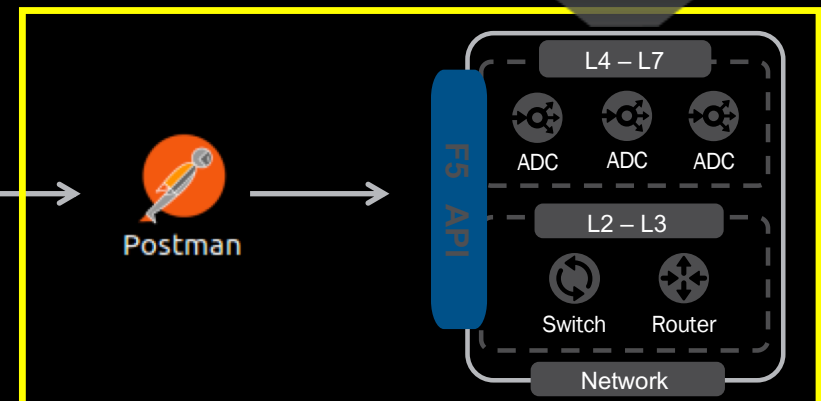
# Class 1 – Module 1:

## Imperative Automation via the **BIG-IP REST API**

Application Experience:  
Performance, High-  
Availability, and Security



Application Services Policy



# Start Module 1

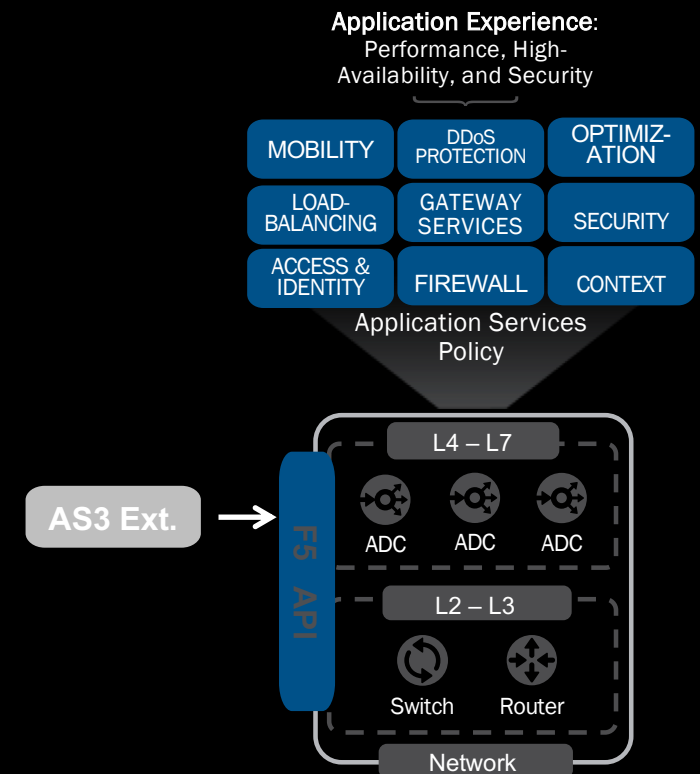
[Click to start timer](#)





# App Services 3 Extension

- Provide the capability to **Appropriately Abstract** services
- Bridge between the **Imperative** and **Declarative** Models
- Single API call for deployment
- Enable granular access to configuration (as needed)
- Host iControl LX Extensions



# Deployment Models

- L4-7 Service Deployments can involve multiple steps
- Deployment is a **Synchronous** or **Asynchronous** operation
- **Synchronous**
  - *Blocking* in nature/**Immediately Consistent**
  - API Response sent AFTER operation completes
  - Can be 'wrapped' to enable Asynchronous operation
- **Asynchronous**
  - *Non-blocking* in nature/**Eventually Consistent**
  - User queries the status of the deployment to determine success or failure
  - Allows other processes in the automation toolchain to continue

# App Services 3 Extension

- Designed specifically to enable Abstraction & Automation
- Fully supported
- Implements an **Synchronous** deployment model
- Can be wrapped to achieve **Asynchronous** operation
  
- Compatible with F5 TMOS versions 12.1 and newer
- Covers many L4-7 use cases including Security Services
- Available on GitHub
- <https://github.com/F5Networks/f5-appsvcs-extension>

# AS3 API Endpoint



# Module Objectives

- Deploy the App Services 3 Extension on a BIG-IP device
- Learn how to interact with deployments using the REST API
- Create/Update deployments
- Read deployments
- Delete deployments

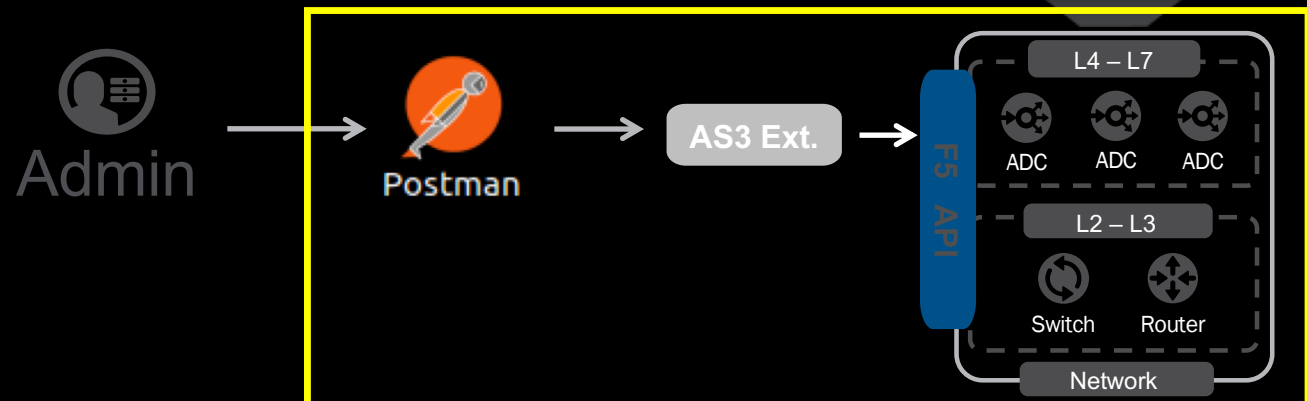
# Class 1 – Module 2:

## Abstracting Services using the App Services 3 Extension

Application Experience:  
Performance, High-Availability, and Security



Application Services Policy



# Start Module 2

[Click to start timer](#)



# Creating Declarative Service Interfaces

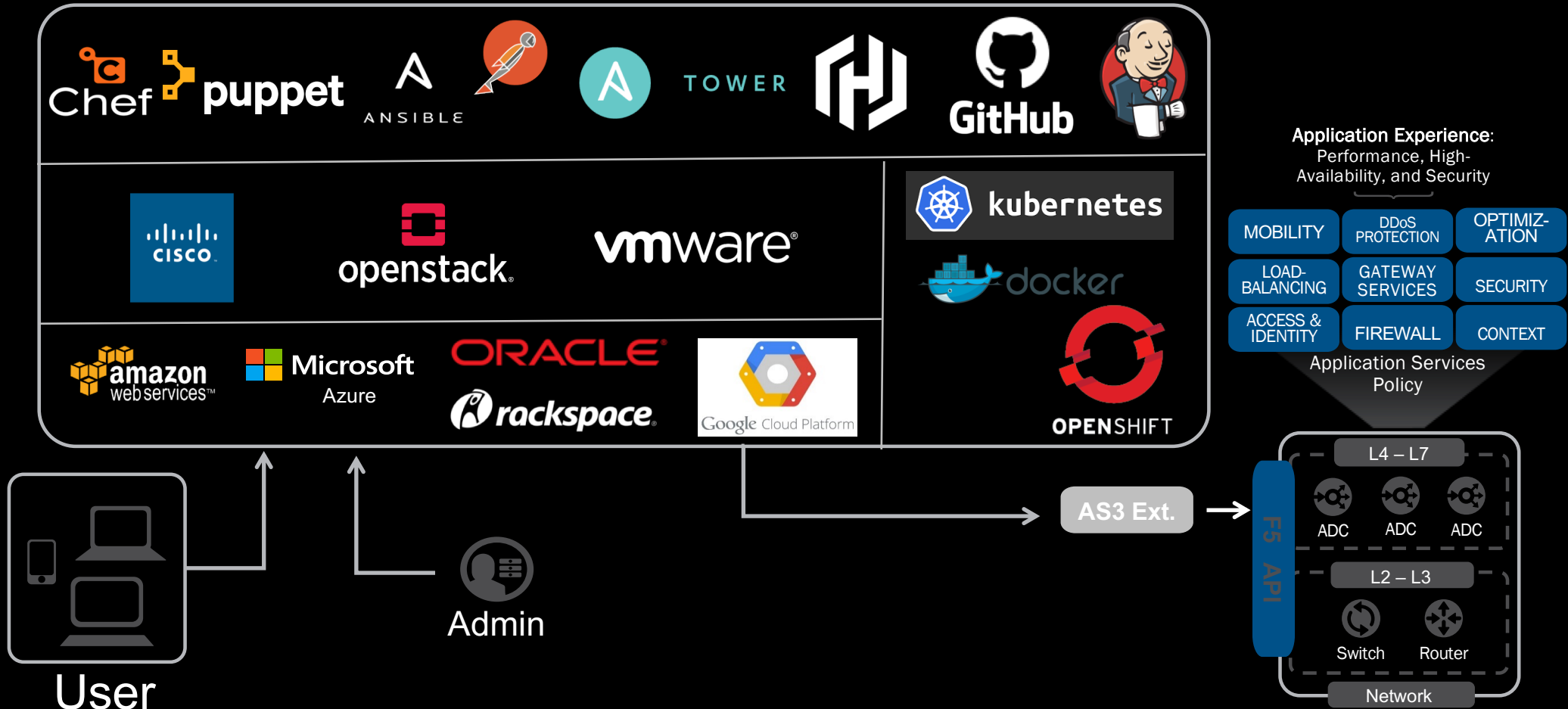




# Toolchain Components

- Source Control Manager
  - Github for SCM
- Orchestrator
  - Ansible Tower
- App Services 3 Extension
  - Abstract BIG-IP configuration into Services

# Declarative Orchestration/DevOps

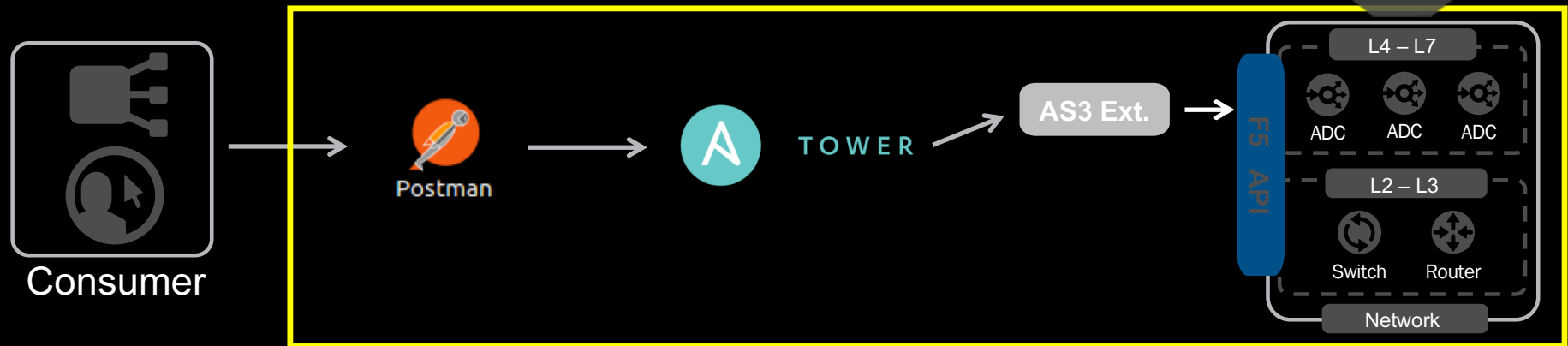


# Class 1 – Module 3: Creating Declarative Service Interfaces

Application Experience:  
Performance, High-  
Availability, and Security



Application Services  
Policy

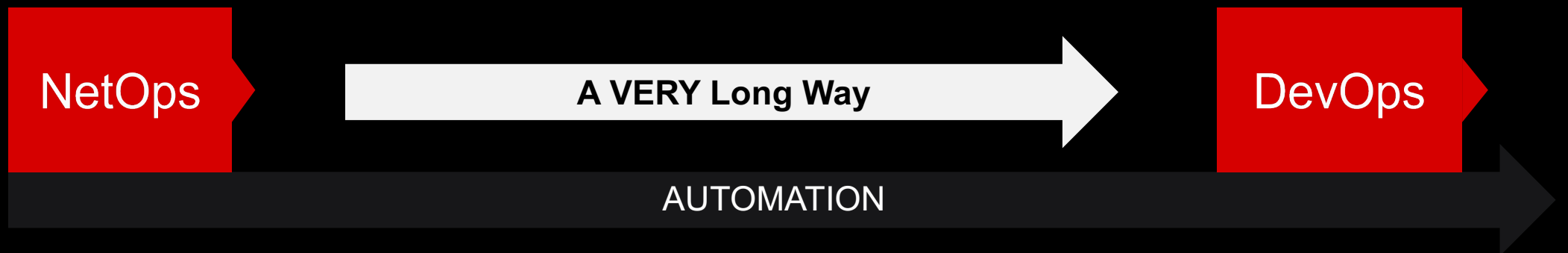


# Start Module 3

[Click to start timer](#)

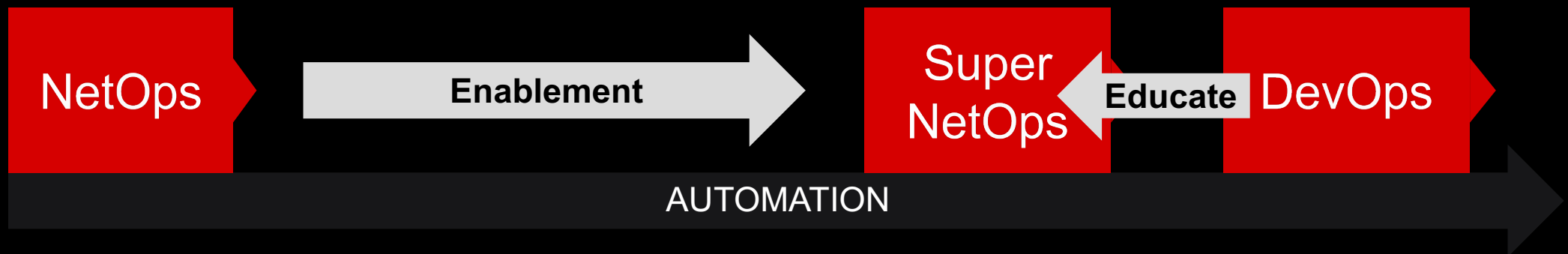


# Skills gap



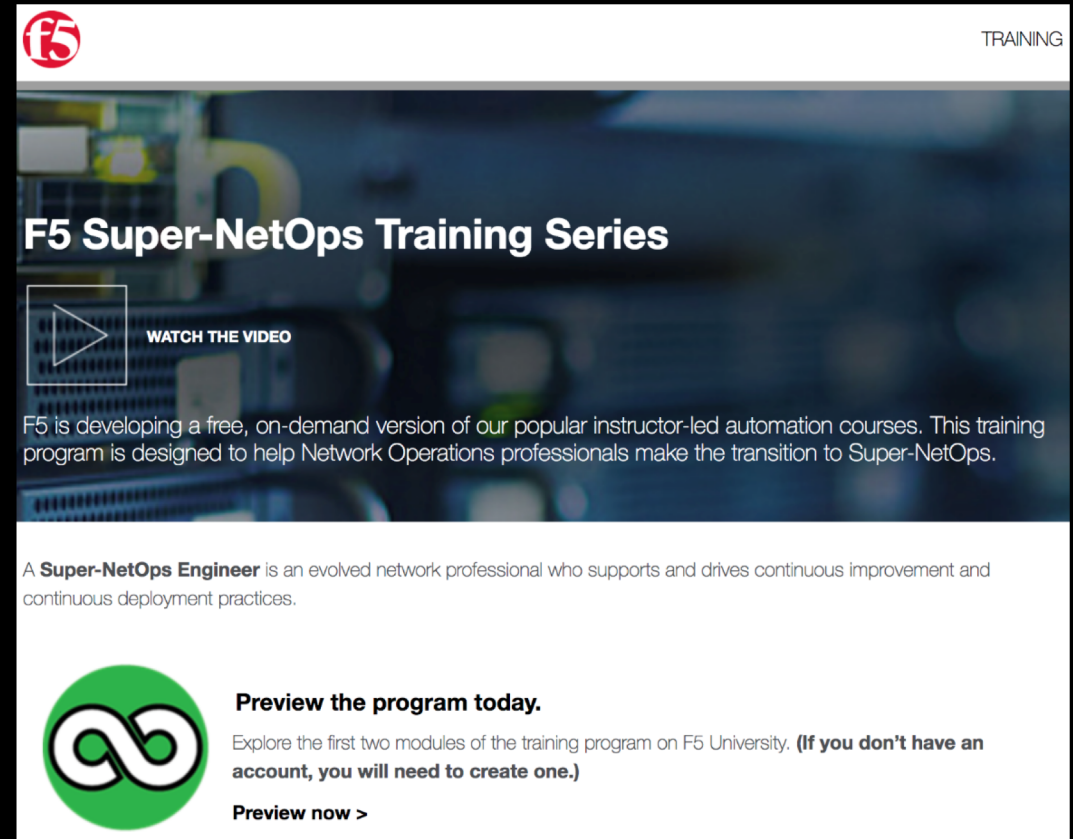


- **Bring BIG-IP expertise to the DevOps party**
- **Create F5 Super-NetOps champions**
- **Bring Awareness to DevOps about our capabilities**



# Free Super-NetOps Training

- Free Self-service training
- Go To <http://f5.com/supernetops>




**F5** TRAINING

## F5 Super-NetOps Training Series

WATCH THE VIDEO

F5 is developing a free, on-demand version of our popular instructor-led automation courses. This training program is designed to help Network Operations professionals make the transition to Super-NetOps.

A **Super-NetOps Engineer** is an evolved network professional who supports and drives continuous improvement and continuous deployment practices.

 **Preview the program today.**  
Explore the first two modules of the training program on F5 University. **(If you don't have an account, you will need to create one.)**  
[Preview now >](#)

# Thank You







**SOLUTIONS FOR AN APPLICATION WORLD**