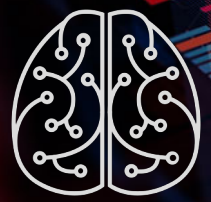
Automating BIG-IP with Ansible

WEBINAR

February 28, 2019

Network Automation Challenges

46%



Lack of talent

40%

Policy creation & governance

38%

Budget for tools

36%

Integration of toolsets across vendors/devices

D510 What do you find most frustrating, challenging about automating the network?

n=850

Agenda

Automate:

Deployment of Virtual BIG-IP appliances

Application deployments to your F5 BIG-IP

Application Lifecycle tasks (Day 2 Operations)

Integrate:

CI Pipeline

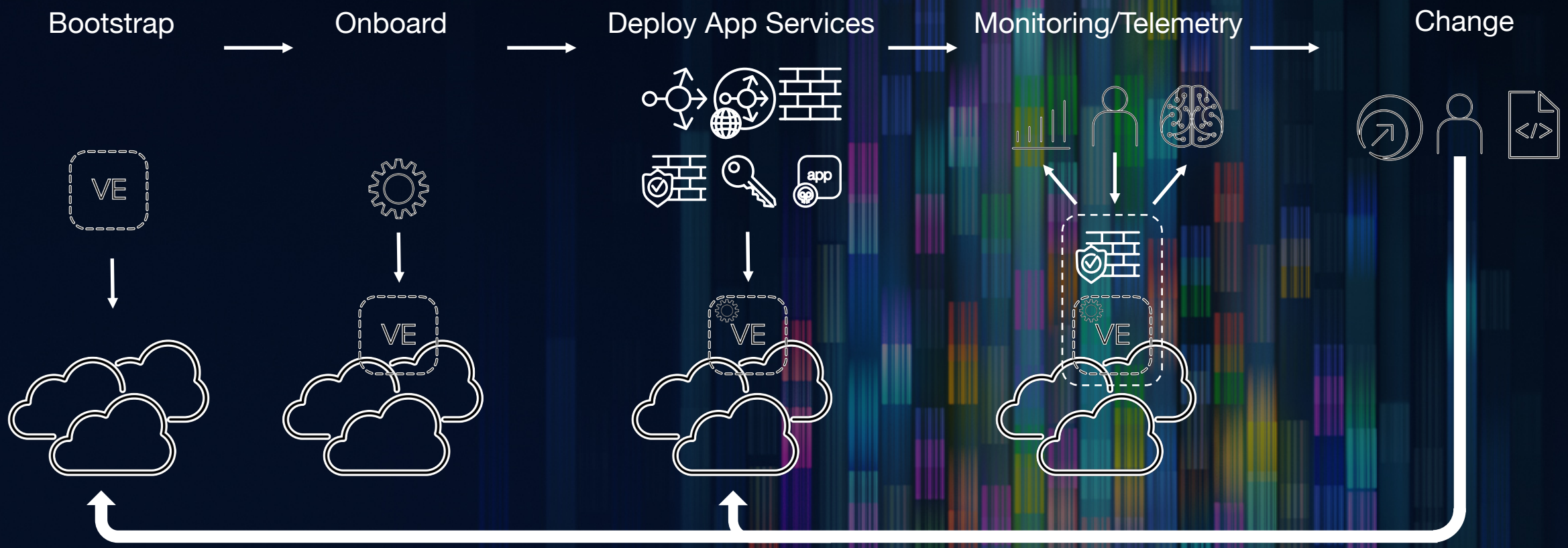
ServiceNow

Automation Series

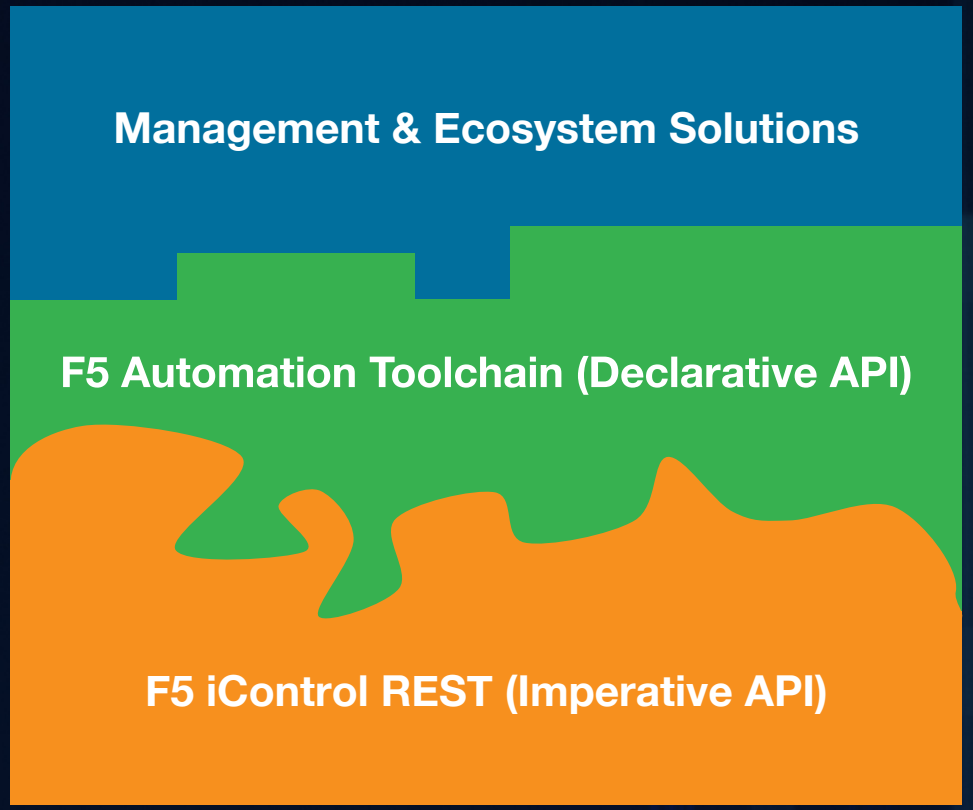
Webinar 1	Automating F5 BIG-IP with Ansible
Webinar 2	Infrastructure-as-Code
Webinar 3	Build a full CI Pipeline

Automation Lifecycle

DEPLOYING JUST AN APP IS NOT ENOUGH



Imperative vs Declarative



Declarative – Tell the system **WHAT** you would like to happen, and let it figure out **HOW** to do it

Imperative – Tell the system **HOW** to do something, and as a result **WHAT** you want to happen

F5 Automation Toolchain

Bootstrap → Onboard → Deploy App Services → Monitoring/Telemetry

CLOUD TEMPLATES

Start BIG-IP Instances in Public & Private Clouds

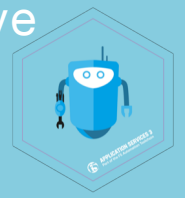
DECLARATIVE ONBOARDING EXTENSION

Initial Config of BIG-IP Instances

L1-L3

APP SERVICES 3 EXTENSION

Deploy Classic and Advanced Application Services on BIG-IP using Declarative REST APIs



L4-L7

TELEMETRY STREAMING EXTENSION

Stream Telemetry, Events & Logs from BIG-IP to various Analytics and Logging solutions

F5 AUTOMATION TOOLCHAIN

F5 Automation Toolchain

Bootstrap → Onboard → Deploy App Services → Monitoring/Telemetry

API SERVICES GATEWAY
Container / VM platform for running F5 & Custom iControl LX Extensions

CLOUD TEMPLATES

Start BIG-IP Instances in Public & Private Clouds

DECLARATIVE ONBOARDING EXTENSION

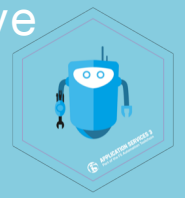
Initial Config of BIG-IP Instances

L1-L3

APP SERVICES 3 EXTENSION

Deploy Classic and Advanced Application Services on BIG-IP using Declarative REST APIs

L4-L7



TELEMETRY STREAMING EXTENSION

Stream Telemetry, Events & Logs from BIG-IP to various Analytics and Logging solutions

F5 AUTOMATION TOOLCHAIN

Ecosystem

BIG-IQ

CONTAINER INGRESS SERVICES

vmware

aws

Azure

Google Cloud

ANSIBLE

Terraform

GitHub

kubernetes

cisco

openstack.

servicenow

GitLab

OPENSHIFT

API SERVICES GATEWAY
Container / VM platform for running F5 & Custom iControl LX Extensions

CLOUD TEMPLATES

Start BIG-IP Instances in Public & Private Clouds

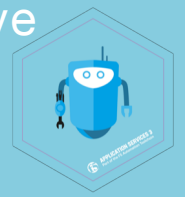
DECLARATIVE ONBOARDING EXTENSION

Initial Config of BIG-IP Instances

L1-L3

APP SERVICES 3 EXTENSION

Deploy Classic and Advanced Application Services on BIG-IP using Declarative REST APIs



L4-L7

TELEMETRY STREAMING EXTENSION

Stream Telemetry, Events & Logs from BIG-IP to various Analytics and Logging solutions

F5 AUTOMATION TOOLCHAIN

Why Ansible

More powerful than bash scripts

Agentless, does not require a master node/server

Modules make it easy to automate common tasks

Over 113 F5 modules as of 2.7 and more coming in 2.8

Requirements

System Requirements:

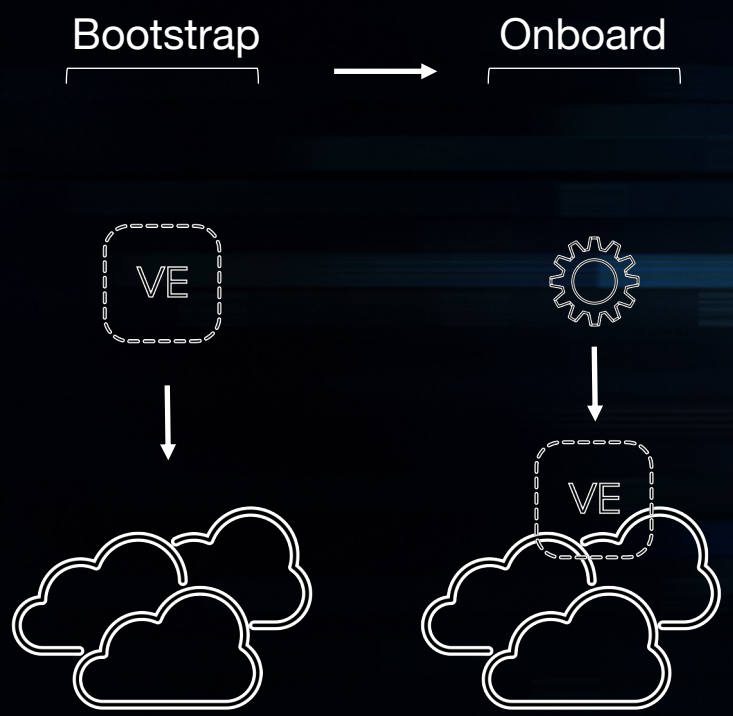
- Python 2.7+
- Ansible 2.7+
  - F5 recommends installing via virtualenv
    - pip install ansible
    - pip install f5-sdk
    - pip install boto3 (for AWS modules)
    - pip install botocore (for AWS modules)
- BIG-IP 12.1+
- Setup your AWS credentials
  - via ~/.aws/credentials or via environment variables
    - https://docs.ansible.com/ansible/latest/modules/ec2_module.html

Tips:

- If you don't have Python installed on your computer, consider running Ansible in a Docker container
- If you do have Python installed on your computer, consider running Ansible in a python virtual environment


Automate

Bootstrap and Onboarding



Bootstrap and Onboarding

PREPARE TO DEPLOY CFT

Ansible modules make it easy to deploy via CFT or ARM

- For this demo, we will use AWS

Always lockdown your management interface

- The CFT asks for a CIDR to restrict management traffic so we'll use the Ansible uri module to obtain this from AWS
- We'll use http://checkip.amazonaws.com to find our public IP
- Register allows us to set a variable that other tasks can use
  - We set our public IP address to the *myip* variable

```
--- # Deploy a BIG-IP in AWS

- name: Deploy BIG-IP
  hosts: localhost
  gather_facts: false

  tasks:
  # Get My IP Address
  - name: Get public IP address
    uri:
      url: "http://checkip.amazonaws.com"
      return_content: yes
    register: myip
```

Bootstrap and Onboarding

DEPLOY CFT

Ansible cloud formation module allows us to deploy a CFT

- AWS variables:
  - stack_name
  - region
  - vpc
  - subnet
  - sshKey
  - instanceType
- F5 variables:
  - template_url (F5's CFT template URL)
  - imageName (AMI image name)
  - restrictedSrcAddress (Management CIDR lockdown)
  - restrictedSrcAddressApp(CIRD lockdown for app traffic)

```
# Deploy BIG-IP in AWS
- name: Deploy a BIG-IP in AWS via CFT
  register: deploy_output
  cloudformation:
    stack_name: "{{ stack_name }}"
    state: present
    region: "{{ region }}"
    template_url: "{{ template_url }}"
    template_parameters:
      Vpc: "{{ vpc }}"
      subnet1Az1: "{{ subnet1Az1 }}"
      imageName: "{{ imageName }}"
      instanceType: "{{ instanceType }}"
      sshKey: "{{ sshKey }}"
      restrictedSrcAddress: "{{ myip.content | replace('\n', '') }}/32"
      restrictedSrcAddressApp: "{{ myip.content | replace('\n', '') }}/32"
```

Bootstrap and Onboarding

SET PASSWORD

We need to wait for the BIG-IP to finish booting

- Ansible wait_for module lets us check the management interface until it is ready
- The results from the CFT deployment provide the IP address of the management interface

BIG-IPs deployed into public clouds do not have a default password, it must be set

- Ansible bigip_command allows us to run tmsh commands against the BIG-IP
- We will run the required tmsh command to set the admin password

```
# Wait for BIG-IP to be ready
- name: Wait for BIG-IP to be ready
  wait_for:
    host: "{{ deploy_output.stack_outputs.Bigip1subnet1Az1SelfEipAddress }}"
    port: "{{ deploy_output.stack_outputs.Bigip1Url | urlsplit('port')}}"
    state: present

# Change the BIG-IP admin password
- name: Change BIG-IP admin password
  bigip_command:
    provider:
      server: "{{ deploy_output.stack_outputs.Bigip1subnet1Az1SelfEipAddress }}"
      #ssh_keyfile: "{{ playbook_dir }}/files/cody-key.pem"
      ssh_keyfile: "{{ ssh_keyfile }}"
      transport: cli
      user: "{{ f5_user }}"
    commands: modify auth user {{ f5_user }} password {{ f5_password }}
```

Bootstrap and Onboarding

INSTALL DECLARATIVE ONBOARDING

Declarative Onboarding provides an easy way to onboard BIG-IP appliances

- Set appliance specific information like:
  - Licensing
  - Name
  - DNS
  - NTP
  - VLAN
  - SIP
- Configure Device Service Groups
  - Easily cluster BIG-IP appliances together

```
# Install Declarative Onboarding RPM
# NOTE: rpm binary must be located on host running playbook
- name: Retrieve DO Install Version
  find:
    paths: "{{ playbook_dir }}/files"
    patterns: "f5-decl*.rpm"
  register: dorpm

- name: Install DO
  bigip_iapplx_package:
    package: "{{ dorpm.files[0].path }}"
    provider:
      server: "{{ deploy_output.stack_outputs.Bigip1subnet1Az1SelfEipAddress }}"
      server_port: "{{ deploy_output.stack_outputs.Bigip1Url | urlsplit('port')}}"
      transport: rest
      user: "{{ f5_user }}"
      password: "{{ f5_password }}"
      validate_certs: no

# Push Declarative Onboarding declaration to BIG-IP
- name: Push DO declaration to BIG-IP
  uri:
    url: "{{ deploy_output.stack_outputs.Bigip1Url }}/mgmt/shared/declarative-onboarding"
    method: POST
    user: "{{ f5_user }}"
    password: "{{ f5_password }}"
    #body: "{{ playbook_dir }}/files/single_nic_do.json"
    body: "{{ lookup('file', 'files/single_nic_do.json') }}"
    status_code: 202
    timeout: 300
    body_format: json
    validate_certs: no
```

Bootstrap and Onboarding

CONFIGURE DECLARATIVE ONBOARDING

Make the API call to push the DO declaration

```yaml
# Push Declarative Onboarding declaration to BIG-IP
- name: Push DO declaration to BIG-IP
  uri:
    url: "{{ deploy_output.stack_outputs.Bigip1Url }}/mgmt/shared/declarative-onboarding"
    method: POST
    user: "{{ f5_user }}"
    password: "{{ f5_password }}"
    body: "{{ lookup('file', 'files/single_nic_do.json') }}"
    status_code: 202
    timeout: 300
    body_format: json
    validate_certs: no
```

```json
{
    "schemaVersion": "1.0.0",
    "class": "Device",
    "async": true,
    "Common": {
        "class": "Tenant",
        "hostname": "bigip.codygreen.com",
        "myDns": {
            "class": "DNS",
            "nameServers": [
                "8.8.8.8"
            ],
            "search": [
                "f5.com",
                "test.com"
            ]
        },
        "myNtp": {
            "class": "NTP",
            "servers": [
                "0.pool.ntp.org",
                "1.pool.ntp.org"
            ],
            "timezone": "UTC"
        },
        "myProvisioning": {
            "class": "Provision",
            "ltm": "nominal"
        }
    }
}
```

Bootstrap and Onboarding

UPDATE ANSIBLE HOST FILE

Once the BIG-IP is deployed update your ansible host file:

- BIG-IP Management IP address
- BIG-IP Management port

```
localhost ansible_connection=local

[aws]
3.18.69.111 f5_admin_port=8443
```

Alternatively, you can use Ansible dynamic host scripts to query inventory from AWS

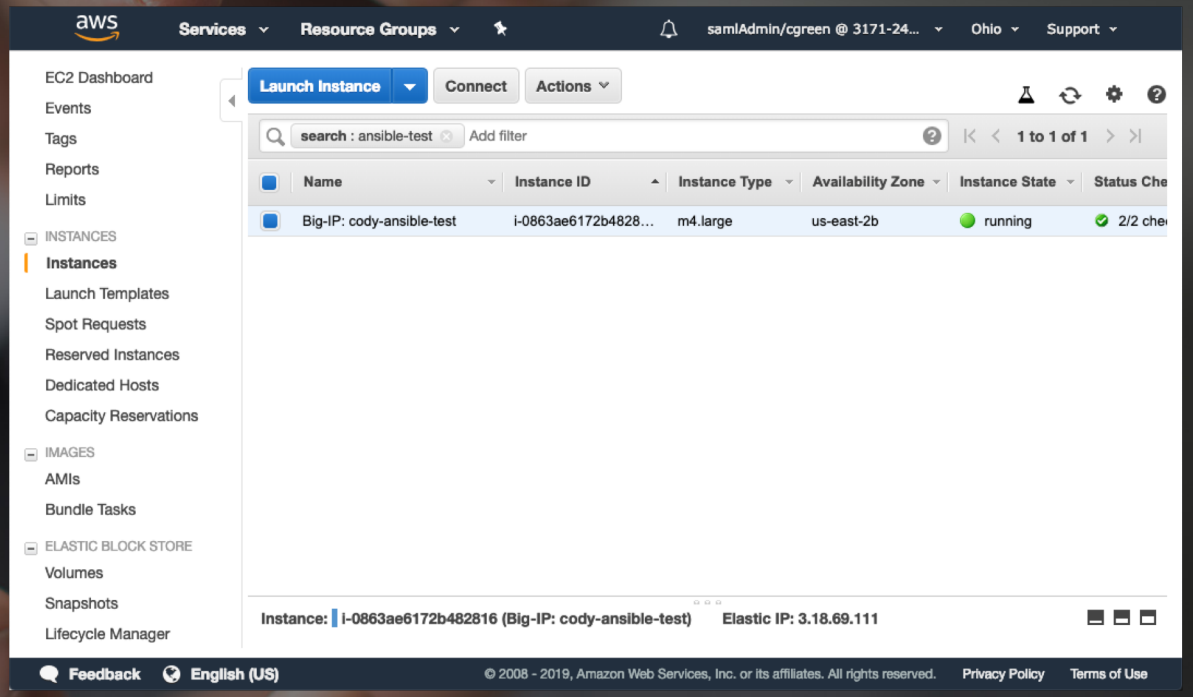
Bootstrap and Onboarding

RECAP

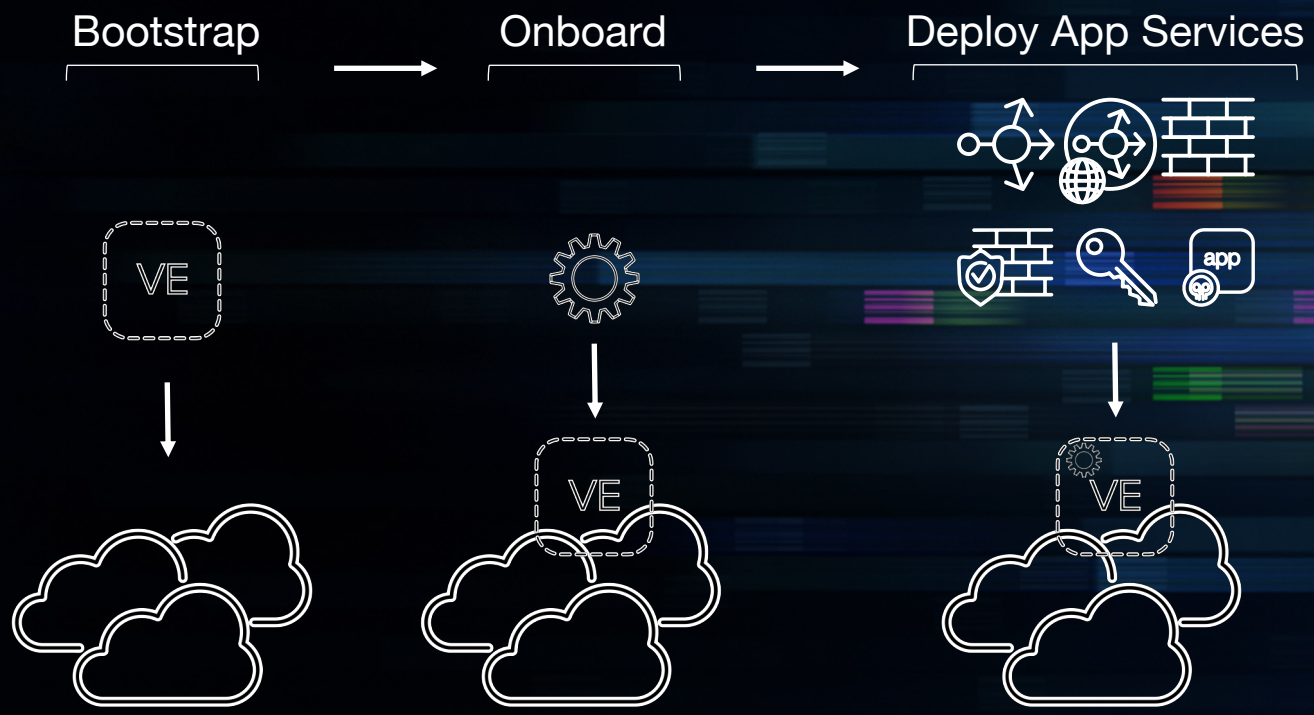
Ansible Playbook combined multiple processes for us:

- Get Public IP
- Deploy CFT
- Wait for BIG-IP management interface to be online
- Change the BIG-IP admin password
- Install Declarative Onboarding extension
- Onboard the BIG-IP using Declarative Onboarding

Provides a fast and repeatable process



Deploy App Services



Deploy App Services

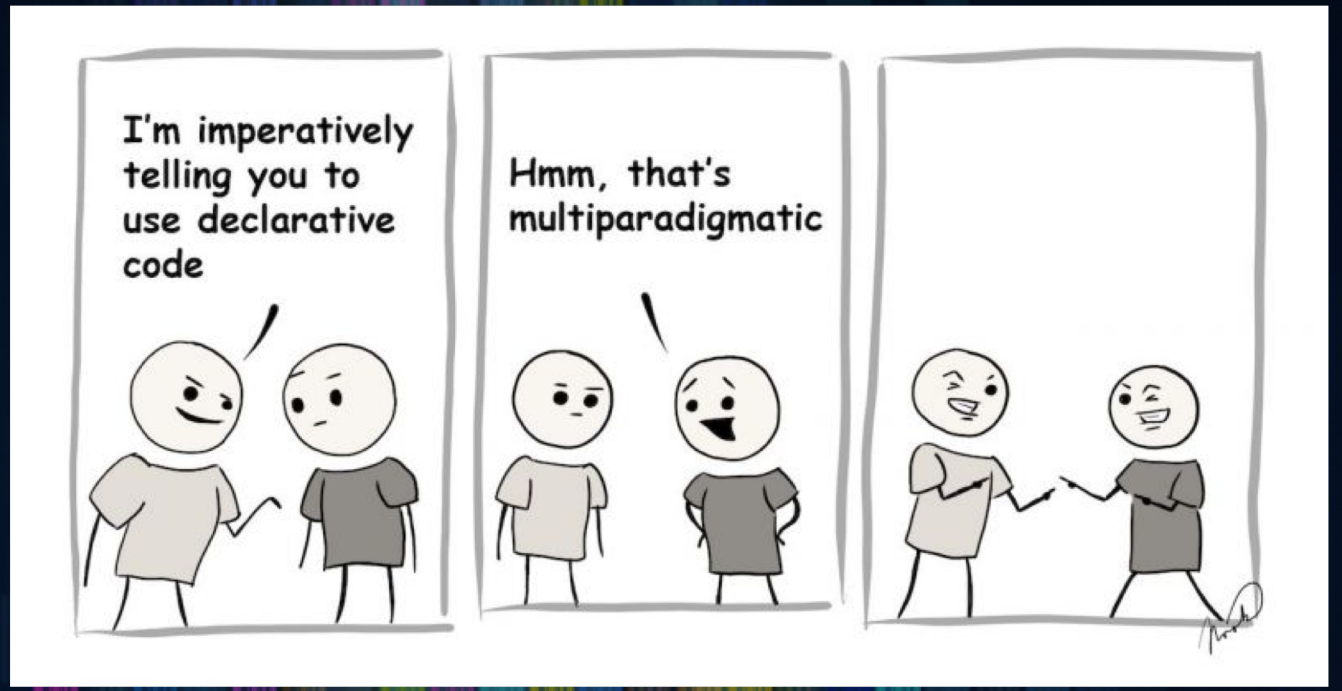
TWO METHODS

Imperative:

- Ansible Modules
- Easy for beginners
- Great for brownfield environments

Declarative:

- Application Services Extension 3 (AS3)
- Greater control and flexibility
- Sets the groundwork for
  - CI/CD integration
  - Infrastructure-as-Code



Deploy App Services

ANSIBLE MODULES

Ansible modules are mapped against F5's imperative APIs

- Very similar to the process F5 admins are use to:
  - Create node
  - Create Pool
  - Add pool member
  - Create Virtual Server
- Pros:
  - Easy to get running, there are good docs and examples
  - Can simplify Day2 operations
- Cons:
  - Can change between TMOS versions
  - Requires domain specific knowledge to create and maintain

```
- name: Add virtual server
  bigip_virtual_server:
    server: lb.mydomain.net
    user: admin
    password: secret
    state: present
    partition: Common
    name: my-virtual-server
    destination: 10.10.10.10
    port: 443
    pool: my-pool
    snat: Automap
    description: Test Virtual Server
    profiles:
      - http
      - name: clientssl
        context: server-side
      - name: ilx
        context: client-side
  delegate_to: localhost
```

Deploy App Services

APPLICATION SERVICE EXTENSION

AS3 provides declarative interfaces for common L4-L7 App Services use cases

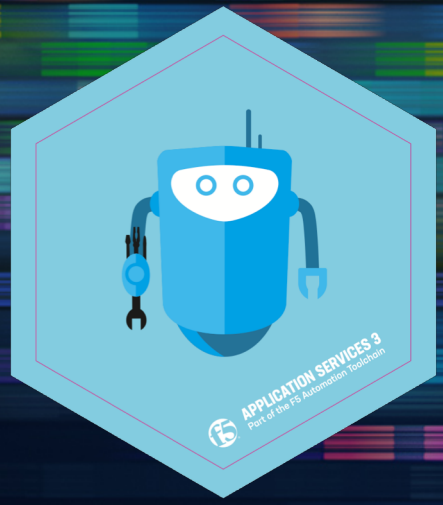
Deployment via a single REST API endpoint

JSON-based documentation with defaults included

Supports multi-tenancy

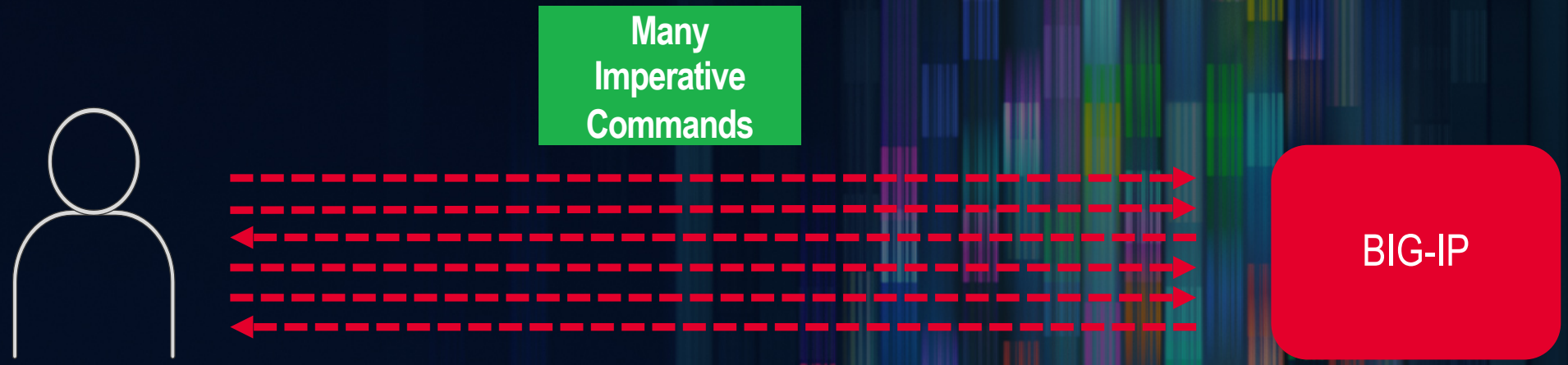
Atomic and Idempotent

Supports pool member service discovery



Deploy App Services

WITHOUT AS3



- Requires BIG-IP domain expertise
- Dozens of REST API calls
- Costly to automate and integrate with orchestration systems
- Time-consuming
- Error-prone

Deploy App Services

WITH AS3

- AS3 abstracts away all BIG-IP configuration complexity
- User only needs to define the desired configuration end-state
- Requires no BIG-IP domain expertise
- Single REST API call simplifies automation and orchestrator integrations
- Accelerates app service deployment
- Declaration is reusable, ensures consistency, reduces errors

Deploy App Services

AS3 SAMPLE DECLARATION

Create a simple HTTP application service:

- HTTP Virtual Server
- Pool named web_pool with 2 members
- Monitored by default HTTP health monitor
- Partition (tenant) name Sample_01

Declaration not ordered, nor sequenced

Post declaration to /mgmt./shared/appsvcs/declare

```
{
   "class": "AS3",
   "action": "deploy",
   "persist": true,
   "declaration": {
      "class": "ADC",
      "schemaVersion": "3.0.0",
      "id": "urn:uuid:33045210-3ab8-4636-9b2a-c98d22ab915d",
      "label": "Sample 1",
      "remark": "Simple HTTP application with RR pool",
      "Sample_01": {
         "class": "Tenant",
         "A1": {
            "class": "Application",
            "template": "http",
            "serviceMain": {
               "class": "Service_HTTP",
               "virtualAddresses": [
                  "10.0.1.10"
               ],
               "pool": "web_pool"
            },
            "web_pool": {
               "class": "Pool",
               "monitors": [
                  "http"
               ],
               "members": [{
                  "servicePort": 80,
                  "serverAddresses": [
                     "192.0.1.10",
                     "192.0.1.11"
                  ]
               }]
            }
         }
      }
   }
}
```

Deploy App Services

AS3 TEMPLATES WITH JINJA 2

Jinja allows us to templatize the AS3 declaration

- Create standardized deployment templates
  - http
  - https
  - TCP
- Jinja 2 offers:
  - variable replacement
  - filters
  - If/else statements
  - for loops

```
"{{app_name}}": {
    "class": "Application",
    "template": "http",
    "serviceMain": {
        "class": "Service_HTTP",
        "virtualAddresses": [
            "{{item.vip_address}}"
        ],
        "pool": "{{app_name}}_pool"
    },
    "{{app_name}}_pool": {
        "class": "Pool",
        "monitors": [
            "http"
        ],
        "members": [
            {
                "servicePort": {{server_port}},
                "serverAddresses": [
                    "{{ item.pool_members| join('", "') }}"
                ]
            }
        ]
    }
}
```

Deploy App Services

CREATE AS3 DECLARATION

Create_app playbook take YAML input and produces a portion of the AS3 declaration

- Stores declaration in apps folder
- Breaks deployments down to
  - Data center
  - tenant

```yaml
app_name: App1
tenant: Tenant1
app_template: http
server_port: 80
snat: self
persistance: cookie
app_locations:
  - datacenter_name: aws
    vip_address: 10.1.1.14
    pool_members:
      - 10.1.1.229
      - 10.1.1.149
      - 10.1.1.150
    pool_monitor: http
    pool_servicePort: 80
```

```yaml
---
- name: Create AS3 App
  hosts: localhost
  gather_facts: false
  connection: local

  tasks:

  - name: Create Directory Structure for Site/Tenant
    file:
      path: apps/{{item[0]}}/{{item[1]}}
      state: directory
    with_nested:
      - "{{datacenters}}"
      - "{{tenants}}"

  - name: Create App File
    template:
      src: templates/{{app_template}}.j2
      dest: apps/{{item.datacenter_name}}/{{tenant}}/{{app_name}}.json
      mode: 'preserve'
    with_items: "{{ app_locations }}"
```

Deploy App Services

PUSH AS3 DECLARATION

push_config playbook deployed AS3 declaration for specified Ansible group

- Builds final AS3 declaration based upon:
  - host data center
  - tenant mappings
- Makes API call to BIG-IP appliances in defined Ansible group
  - Easy way to deploy configuration to multiple BIG-IP appliances

```
- name: Group all Application Files
  assemble:
    remote_src: False
    src: "apps/{{item[0]}}/{{item[1]}}/"
    dest: tmp/{{item[0]}}-{{item[1]}}_combined.yaml
    delimiter: ","
  with_nested:
    - "{{datacenters}}"
    - "{{tenants}}"

- name: Retrive AS3 JSON Body
  set_fact: tenant_body="{{ lookup('file', 'tmp/{{item[0]}}-{{ite
  with_nested:
    - "{{datacenters}}"
    - "{{tenants}}"
  register: tenant_json

- name: Retrive AS3 files to push
  command: "ls tmp/"
  register: dir_out

- name: URI POST Tenant
  uri:
    url: "https://{{ groups[item.split('-')[0]][0] }}:{{hostvars|
    method: POST
    user: "admin"
    password: "{{ f5_password }}"
    validate_certs: no
    body: "{{ lookup('template', 'templates/tenant_base.j2') }}"
    body_format: json
  with_items:
    - "{{dir_out.stdout_lines}}"
```

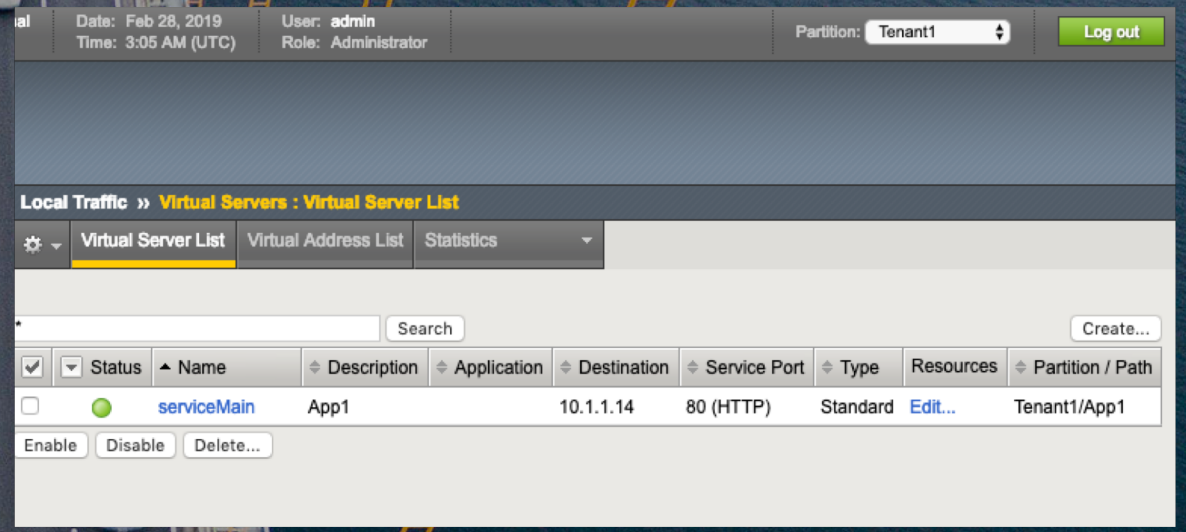
Deploy App Services

RECAP

App Services can be deployed via:

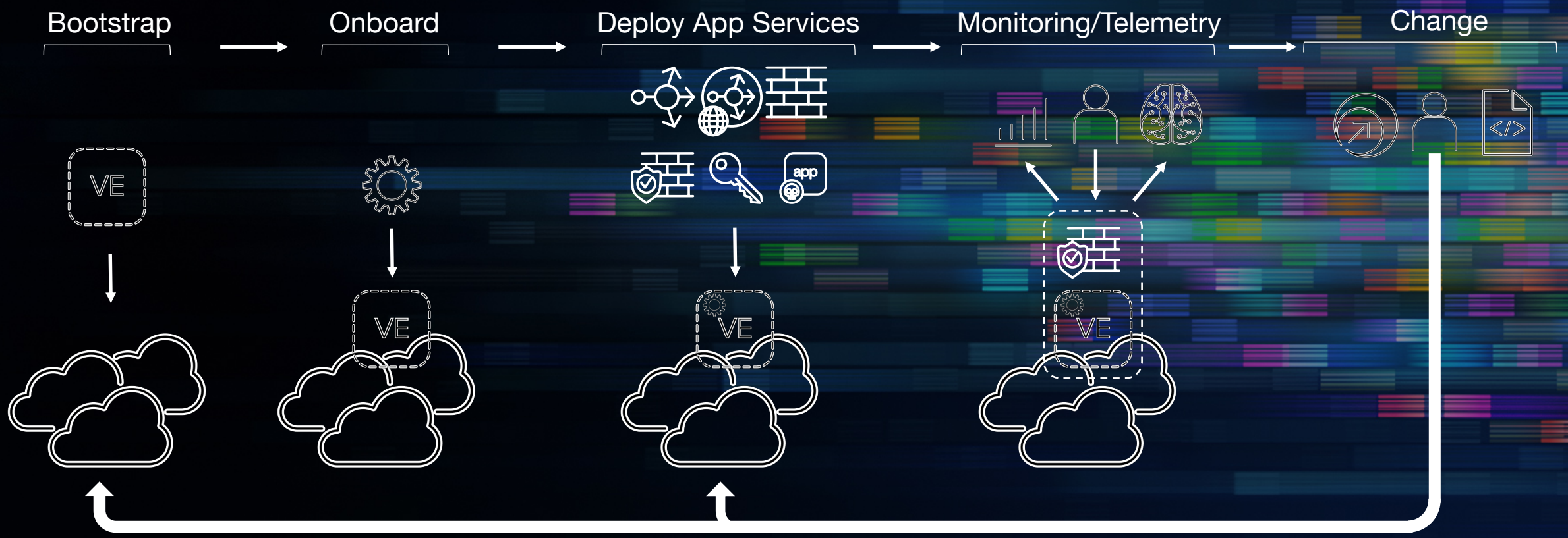
- Ansible Modules
- AS3 declaration using Jinja2 templates

No wrong way, use the method that is best for you!



Change

LIFECYCLE MANAGEMENT



Change

HANDLING LIFECYCLE MANAGEMENT

Common Day 2 BIG-IP tasks:

- add/remove pool members
- enable/disable pool members
- add/remove nodes
- enable/disable nodes
- update SSL certificate

In this demo we will focus on node enabling and disabling nodes

- Easy way to take servers offline based on IP address

Change

ENABLE/DISABLE NODES

We want to ensure the nodes we are trying to modify reside on the targeted BIG-IP

- use the bigip_device_facts to get list of nodes
  - save output as nodes_output variable using the register method
- modify node state using bigip_node module
  - loop through nodes on the targeted BIG-IP
  - when statement ensure the current node in the loop is in our list of nodes to modify
  - state can be:
    - enabled
    - disabled
    - offline

```
tasks:
  - name: Get nodes
    register: nodes_output
    bigip_device_facts:
      gather_subset:
        - nodes
      provider: "{{ provider }}"
    delegate_to: localhost

  - name: Enable/Disable a node
    bigip_node:
      state: "{{ state }}"
      partition: Common
      name: "{{ item.name }}"
      address: "{{ item.address }}"
      provider: "{{ provider }}"
    loop: "{{ query('items', '{{ nodes_output.nodes }}') }}"
    when: item.address in nodes
    delegate_to: localhost
```


Integrate

CI Pipelines

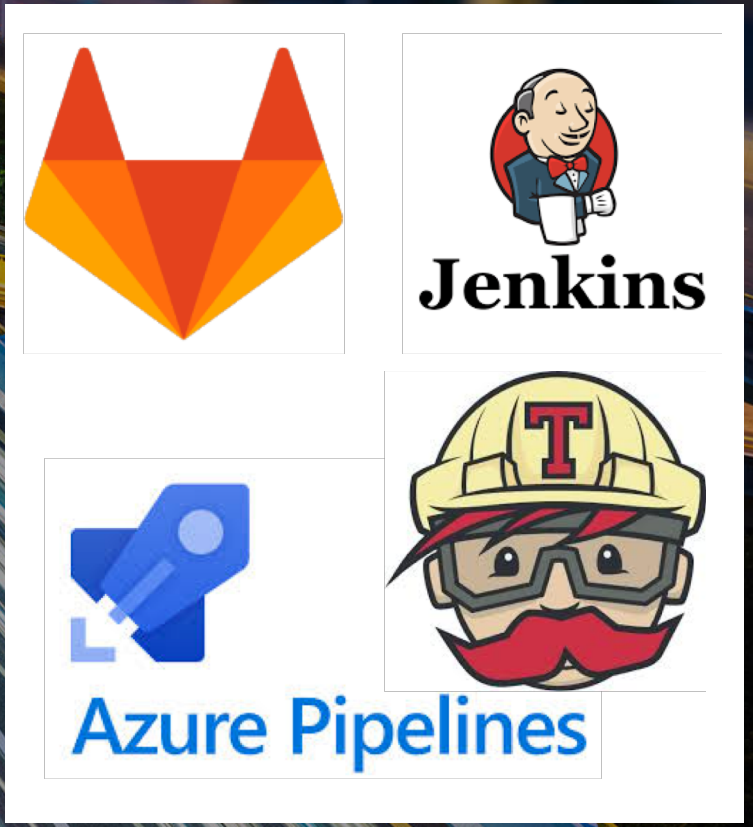
ANSIBLE MAKES IT EASY

Many popular CI tools offer support for Ansible

- Makes it easy to kickoff Ansible playbooks based upon a defined trigger
- Opens the door for Infrastructure-as-Code

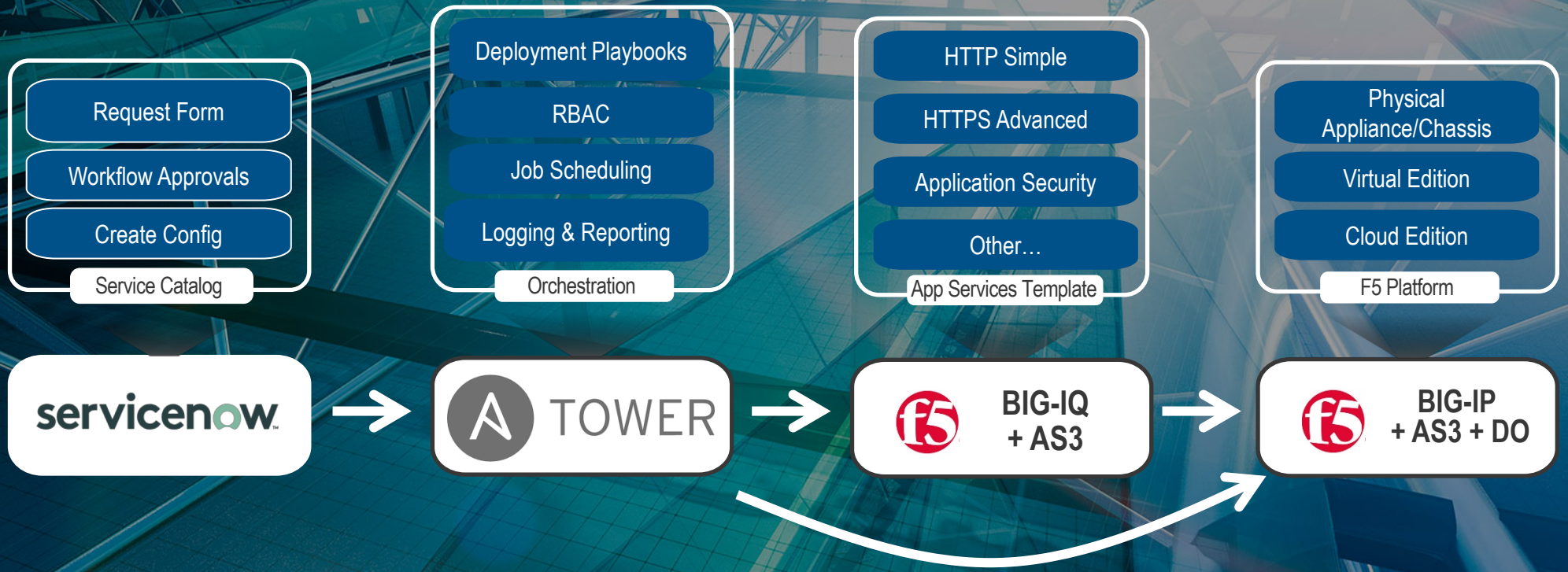
You can also have the CI tool start a custom Ansible docker container

- Easier solution when you need to load custom Python libraries



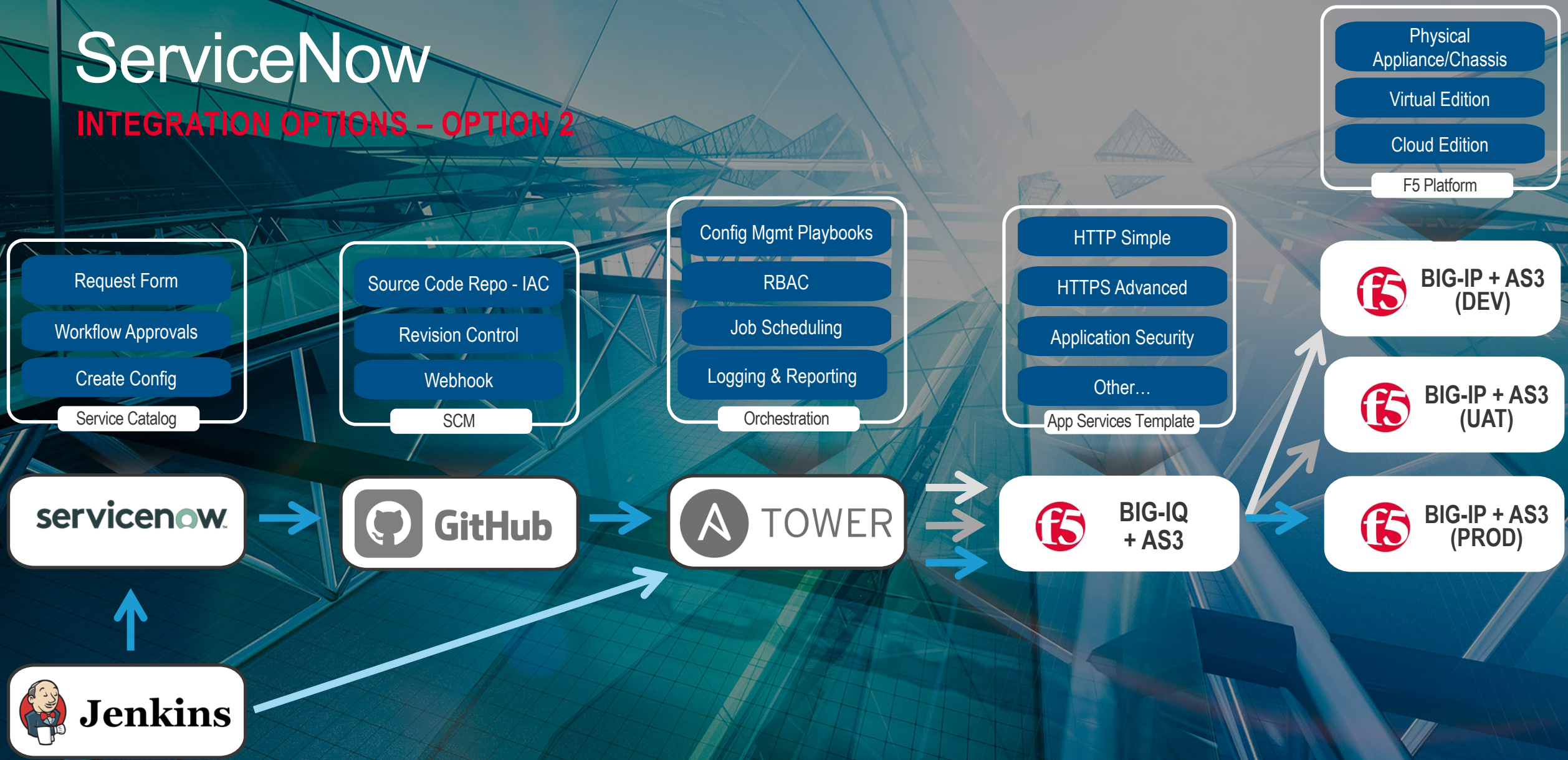
ServiceNow

INTEGRATION OPTIONS – OPTION 1



ServiceNow

INTEGRATION OPTIONS – OPTION 2




MY BRAIN IS FULL
E
F

Next Steps

Check out the webinar repository and test it in your environment:

- https://github.com/codygreen/Automation_Webinar

Register for the next webinar

- https://f5networks.zoom.us/webinar/register/WN_eN-1cZsOQ7mAJVISPIfszA

Take the survey to help provide feedback on topics you would like to see

- https://forms.office.com/Pages/ResponsePage.aspx?id=L_093Ttq0UCb4L-DJ9gcUA7GgOH8VSdJr8ne_RYXDxVUQUY1T1JLSEJESUtQUklZNldGVVpZQk1DNS4u

Join YouTube Livestream tomorrow for Q&A session