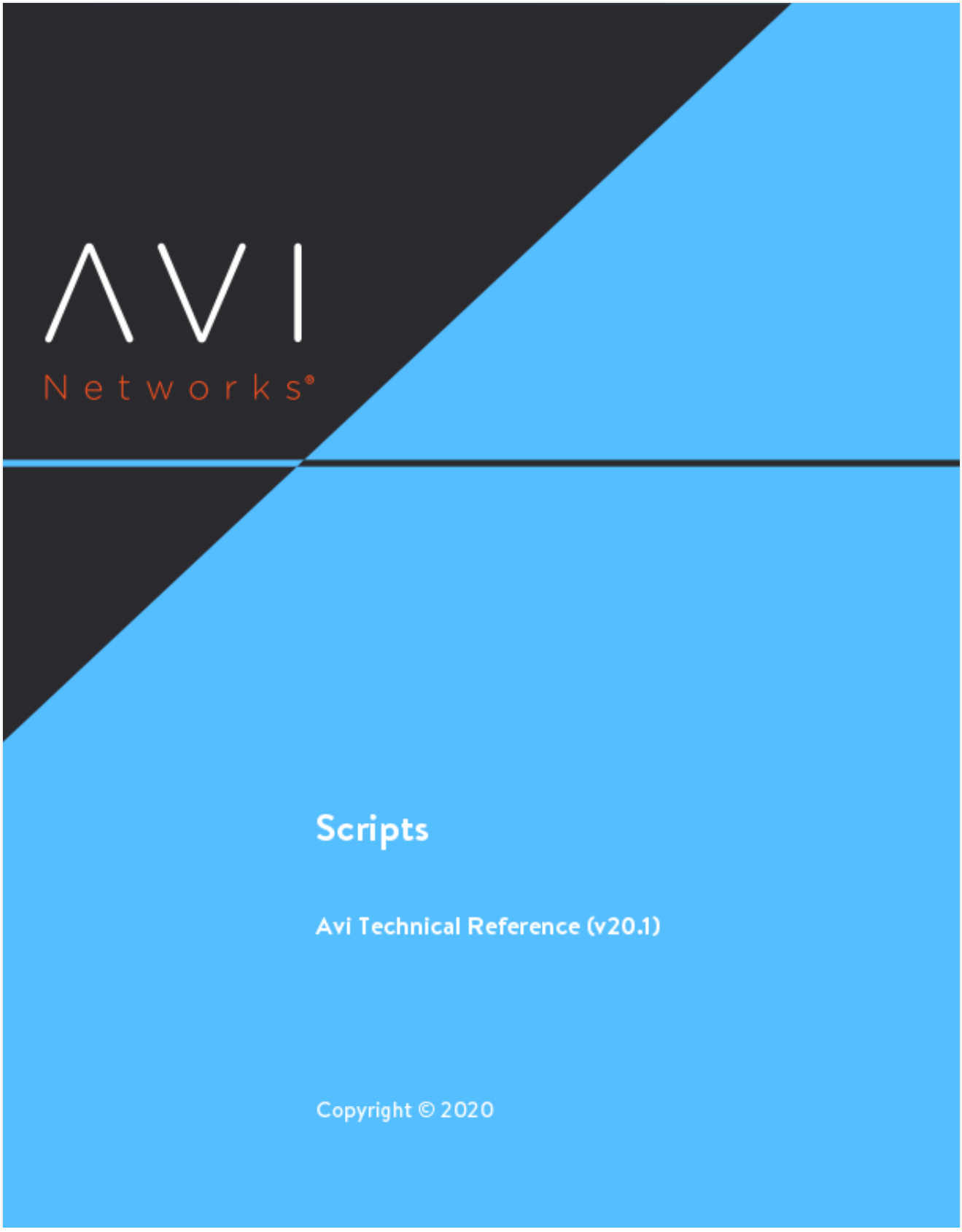

Scripts

Avi Technical Reference (v20.1)

Scripts

view online

Extensive behavior customization and automation can be done by utilizing scripts for both the control plane with the Python-based ControlScripts, and the data plane via Lua-based DataScripts.

DataScript

DataScripts are a powerful mechanism for customizing the behavior of Avi Vantage on a per virtual service, or even per client, basis. DataScripts are lightweight scripts coded in Lua. These scripts can be executed against each client making a TCP connection, an HTTP request or response, or other events within the dataplane.

One or more DataScripts may be attached to the rules section of a virtual service.

Scripts may be uploaded or copy/pasted into either the Request Event Script or Response Event Script section. For instance, to restrict access to the secure directory, the following text would be pasted into the Request Event Script section.

```
if avi.http.uri == "/secure/" then
   avi.http.send(403)
end
```

See the Avi DataScript Guide for complete documentation of commands and example DataScripts.

ControlScript

ControlScripts are Python-based scripts which execute on the Avi Vantage Controllers. They are initiated by *Alert Actions*, which themselves are triggered by events within the system. Rather than alerting an admin that a specific event has occurred, the ControlScript can take specific action, such as altering the Avi Vantage configuration or sending a custom message to an external system, such as telling VMware?s vCenter to scale out more servers if the current servers have reached resource capacity and are incurring lowered health scores.

To create a ControlScript, 1. From the Avi UI, navigate to Templates > Scripts > ControlScripts. 2. Click on Create. The New ControlScript screen appears. 3. Enter a Name for the ControlScript. Note: The *Enter Text* option is selected by default. * Enter the user defined alert action script in the text box provided. or 3. Upload a file. a) Select the Upload File option. b) Click on the Upload File button. c) Select the *.py* file required. 4. Click on Save.

Note: If the size of the payload is more than 128kb, then it will be discarded.

ControlScripts are executed with limited privileges within the Linux subsystem of the Avi Controller. See standard Python documentation for examples and definitions of Python commands. Avi Vantage configuration changes may be made by API calls from Linux to Avi Vantage via standard API mechanisms.

There are two sets of variables that are passed onto the ControlScripts. These include: * Environment Variables * Script Arguments

Environment Variables

The `654F5896F2604114B1A3FD40BEBBF809_0` in the following is the Kubernetes namespace followed by the node value. This value will change on each customer deployment. .

```
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_SERVICE_PORT_CLUSTER: '8443'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_SERVICE_PORT_SECURECHANNEL: '5098'
SERVICE: avipythoncontroller
PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION_VERSION: '2'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_22_TCP_PROTO: tcp
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_8443_TCP_ADDR: 10.43.252.87
LOGNAME: admin
USER: admin
API_TOKEN: 9d63d5bf6e655fa63c25e59090604bcc0f930551
leader_addr: node1.controller.local
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_80_TCP: 'tcp://10.43.252.87:80'
PYTHONUNBUFFERED: '1'
MY_NODE_NAME: gke-us-west1-saas-cl-node-pool-f6pi1f-85a3f803-rq4n
NUM_CPU: '8'
KUBERNETES_SERVICE_PORT: '443'
KUBERNETES_PORT: 'tcp://10.43.240.1:443'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_22_TCP_PORT: '22'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_443_TCP_ADDR: 10.43.252.87
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_5098_TCP: 'tcp://10.43.252.87:5098'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_8443_TCP: 'tcp://10.43.252.87:8443'
DISK_GB: '100'
DATABASE_IP: node1.controller.local
SHLVL: '1'
NUM_MEMG: '30'
KUBERNETES_SERVICE_HOST: 10.43.240.1
WORKER_UUID: '409CCBF6-A5A0-E37B-18DB-498994020D4D-pythoncontroller:0'
KUBERNETES_PORT_443_TCP_PORT: '443'
MANAGEMENT_IP: >-
  node-0-0.service-654f5896f2604114b1a3fd40bebbf809.654f5896f2604114b1a3fd40bebbf809.svc.cluster.local
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_SERVICE_PORT: '22'
role: leader
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_8443_TCP_PROTO: tcp
CNTRL_SSH_PORT: '5098'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_80_TCP_PORT: '80'
HOME: /home/admin
TENANT_UUID: admin
GRPC_ENABLE_FORK_SUPPORT: '0'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT: 'tcp://10.43.252.87:22'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_80_TCP_ADDR: 10.43.252.87
GCE_METADATA_ROOT: 169.254.169.254
PYTHONPATH: >-
  /opt/avi/python/lib:/opt/avi/python/bin/portal:/usr/local/lib/python2.7/dist-packages
MY_POD_NAME: node-0-0
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_SERVICE_PORT_HTTPS: '443'
PATH: '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_SERVICE_HOST: 10.43.252.87
PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION: cpp
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_22_TCP: 'tcp://10.43.252.87:22'
KUBERNETES_PORT_443_TCP_ADDR: 10.43.240.1
INSTALL_PYTHON_PATH: /opt/avi/python
container: docker
```

```
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_SERVICE_PORT_HTTP: '80'
KUBERNETES_SERVICE_PORT_HTTPS: '443'
TENANT: admin
LANG: en_US.UTF-8
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_443_TCP_PROTO: tcp
KUBERNETES_PORT_443_TCP_PROTO: tcp
node_uuid: 409CCBF6-A5A0-E37B-18DB-498994020D4D
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_SERVICE_PORT_SSH: '22'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_5098_TCP_ADDR: 10.43.252.87
KUBERNETES_PORT_443_TCP: 'tcp://10.43.240.1:443'
quiet: '0'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_8443_TCP_PORT: '8443'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_22_TCP_ADDR: 10.43.252.87
INSTANCE: '0'
PWD: /home/admin
HOSTNAME: node-0-0
DJANGO_SETTINGS_MODULE: portal.settings_non_portal
CONTAINER: DOCKER
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_443_TCP: 'tcp://10.43.252.87:443'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_443_TCP_PORT: '443'
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_80_TCP_PROTO: tcp
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_5098_TCP_PROTO: tcp
TZ: UTC
EXTERNAL_SERVICE_654F5896F2604114B1A3FD40BEBBF809_0_PORT_5098_TCP_PORT: '5098'
SAAS_CONTROLLER: 'true'
```

When accessing any of these values in Python they can be seen in `os.environ`.

Script Arguments

The arguments that are provided to the script are provided as an array in this format:

```
['/home/admin/{{ ALERT NAME}}', '{{ ALERT DETAILS }}']
```

Alert Details

Alert details are provided as JSON data and can be parsed using JSON as `json.loads(sys.argv[1])`. Here is the pretty print json of the data provided to the ControlScript:

```
{
  "name": "System-CC-Alert-cluster-878226b4-ff2c-4e6b-a9a7-66aa58ad23f1-1580412633.0-1580412633-38875205",
  "throttle_count": 0,
  "level": "ALERT_LOW",
  "reason": "threshold_exceeded",
  "obj_name": "AWS",
  "threshold": 1,
  "events": [
    {
      "event_id": "AWS_ACCESS_FAILURE",
      "event_details": {
```

```
        "aws_infra_details": {
          "vpc_id": "vpc-0617ccd15673817c0",
          "region": "eu-central-1",
          "error_string": "AuthFailure: AWS was not able to validate the provided access credentials\\n\\tstatus code:
          "cc_id": "cloud-8a3e601b-d06d-4998-bc41-ac3004330066"
        }
      },
      "obj_uuid": "cluster-878226b4-ff2c-4e6b-a9a7-66aa58ad23f1",
      "obj_name": "AWS",
      "report_timestamp": 1580412633
    }
  ]
}
```


Examples

To understand scripting, use the examples on how to use the values, given below:

Getting Data Passed to the ControlScript

```
#!/usr/bin/python
#
# Avi Networks ControlScript
#
# This sample ControlScript will output the environment values, and alert
# arguments that are passed from the alert that triggered the alert script.
# You can use these values to help construct your python script actions to
# handle the alert.
#
#
import os
import sys

if __name__ == "__main__":
    print("Environment Vars: %s \n" % os.environ)
    print("Alert Arguments: %s \n" % sys.argv)
```

Sticky Pool Group

```
#!/usr/bin/python
import os
import sys
import json
from avi.sdk.avi_api import ApiSession
from requests.packages import urllib3

urllib3.disable_warnings()

def switch_priorities(session, pool_uuid, pool_name, retries=5):
```

```
    if retries <= 0:
        return 'Too many retry attempts - aborting!'
    query = 'refers_to=pool:%s' % pool_uuid
    pg_result = session.get('poolgroup', params=query)
    if pg_result.count() == 0:
        return 'No pool group found referencing pool %s' % pool_name

    pg_obj = pg_result.json()['results'][0]

    highest_up_pool = None
    highest_down_pool = None

    for member in pg_obj['members']:
        priority_label = member['priority_label']
        member_ref = member['pool_ref']
        pool_runtime_url = ('%s/runtime/detail' %
                            member_ref.split('/api/')[1])
        pool_obj = session.get(pool_runtime_url).json()[0]
        if pool_obj['oper_status']['state'] == 'OPER_UP':
            if (not highest_up_pool or
                int(highest_up_pool[1]) < int(priority_label)):
                highest_up_pool = (member, priority_label,
                                   pool_obj['name'])
        elif (not highest_down_pool or
              int(highest_down_pool[1]) < int(priority_label)):
            highest_down_pool = (member, priority_label,
                                 pool_obj['name'])

    if not highest_up_pool:
        return ('No action required as all pools in the '
                'pool group are now down.')
    elif not highest_down_pool:
        return ('No action required as all pools in the '
                'pool group are now up.')

    if int(highest_down_pool[1]) <= int(highest_up_pool[1]):
        return ('No action required. The highest-priority available '
                'pool (%s) already has a higher priority than the '
                'highest-priority non-available pool (%s)' %
                (highest_up_pool[2], highest_down_pool[2]))

    highest_up_pool[0]['priority_label'] = highest_down_pool[1]
    highest_down_pool[0]['priority_label'] = highest_up_pool[1]

    p_result = session.put('poolgroup/%s' % pg_obj['uuid'], pg_obj)
    if p_result.status_code < 300:
        return ', '.join(['Pool %s priority changed to %s' % (p[0], p[1])
                          for p in ((highest_up_pool[2],
                                     highest_down_pool[1]),
                                    (highest_down_pool[2],
                                     highest_up_pool[1]))])
```

```
    if p_result.status_code == 412:
        return switch_priorities(session, pool_uuid,
                                 pool_name, retries - 1)

    return 'Error setting pool priority: %s' % p_result.text

def sticky_pool_group():
    token = os.environ.get('API_TOKEN')
    user = os.environ.get('USER')
    tenant = os.environ.get('TENANT')
    alert_dict = json.loads(sys.argv[1])
    events = alert_dict.get('events', [])
    if len(events) > 0:
        pool_uuid = events[0]['obj_uuid']
        pool_name = events[0]['obj_name']
        try:
            with ApiSession("localhost", user,
                            token=token,
                            tenant=tenant) as session:
                result = switch_priorities(session, pool_uuid, pool_name)
        except Exception as e:
            result = str(e)
    else:
        result = 'No event data for ControlScript'

    return result

# Use with a ControlScript and Alert(s) to perform 'sticky' failover of pool groups.
#
# Alert should trigger on 'Pool Up' and 'Pool Down' events.
#
# Create a ControlScript as follows:
#
# #!/usr/bin/python
#
# from avi.sdk.samples.sticky_pool_group import sticky_pool_group
#
# if __name__ == '__main__':
#   print sticky_pool_group()
#
```

Add Route to GCP SE

```
#!/usr/bin/python

import sys, os, json, traceback, re, time
from avi.sdk.avi_api import ApiSession
from oauth2client.client import GoogleCredentials
from googleapiclient import discovery
```

```
'''
This ControlScript is executed on the Avi Controller every time there is a
CC_IP_ATTACHED or a CC_IP_DETACHED event.

CC_IP_ATTACHED: Event is triggered when a VIP is attached to a SE
CC_IP_DETACHED: Event is triggered when a VIP is detached from a SE, usually
when a SE goes down or a scale in occurs

The goal of this script is to add a route to GCP with the destination as the
VIP and nextHopIp as the GCP instance IP on which the Avi SE is running after a
CC_IP_ATTACHED event. After a CC_IP_DETACHED event, the goal of the script is
to remove the corresponding route.

Script assumptions:

1) The Avi Controller GCP instance has scope=compute-rw to be able to modify
routes in GCP
2) 'description' field in the Avi Service Engine Group is configured as a
JSON encoded string containing GCP project, zone and network

Event details contain the Avi SE UUID and the VIP.

1) GET Avi SE object from UUID and extract Avi SE IP address (which is
the same as the GCP instance IP address) and Avi Service Engine Group link

2) GET Avi Service Engine Group object. The 'description' field in the
Service Engine Group is a JSON encoded string containing GCP project and
network URL. Extract project and network from the 'description' field

3) Extract all routes matching destRange as VIP from GCP

4) If event is CC_IP_DETACHED, remove matching route with
destRange as vip and nextHopIp as instance IP in the appr network
If event is CC_IP_ATTACHED and no matching route exists already, add a new
route with destRange as vip and nextHopIp as instance IP in appr network
'''

def parse_avi_params(argv):
    if len(argv) != 2:
        return {}
    script_parms = json.loads(argv[1])
    return script_parms

def create_avi_endpoint():
    token=os.environ.get('API_TOKEN')
    user=os.environ.get('USER')
    # tenant=os.environ.get('TENANT')
    return ApiSession.get_session("localhost", user, token=token,
                                  tenant='admin')

def google_compute():
```

```
    credentials = GoogleCredentials.get_application_default()
    return discovery.build('compute', 'v1', credentials=credentials)

def gcp_program_route(gcp, event_id, project, network, inst_ip, vip):
    # List all routes for vip
    result = gcp.routes().list(project=project,
                               filter='destRange eq %s' % vip).execute()
    if (('items' not in result or len(result['items']) == 0)
        and event_id == 'CC_IP_DETACHED'):
        print(('Project %s destRange %s route not found' %
              (project, vip)))
        return

    if event_id == 'CC_IP_DETACHED':
        # Remove route for vip nextHop instance
        for r in result['items']:
            if (r['network'] == network and r['destRange'] == vip and
                r['nextHopIp'] == inst_ip):
                result = gcp.routes().delete(project=project,
                                             route=r['name']).execute()
                print(('Route %s delete result %s' % (r['name'], str(result))))
                # Wait until done or retries exhausted
                if 'name' in result:
                    start = int(time.time())
                    for i in range(0, 20):
                        op_result = gcp.globalOperations().get(project=project,
                                operation=result['name']).execute()
                        print(('op_result %s' % str(op_result)))
                        if op_result['status'] == 'DONE':
                            if 'error' in result:
                                print(('WARNING: Route delete had errors '
                                       'result %s' % str(op_result)))
                            else:
                                print(('Route delete done result %s' %
                                       str(op_result)))
                            break
                        if int(time.time()) - start > 20:
                            print(('WARNING: Wait exhausted last op_result %s' %
                                  str(op_result)))
                            break
                        else:
                            time.sleep(1)
                else:
                    print('WARNING: Unable to obtain name of route delete '
                          'operation')
    elif event_id == 'CC_IP_ATTACHED':
        # Add routes to instance
        # Route names can just have - and alphanumeric chars
        rt_name = re.sub('[./]+', '-', 'route-%s-%s' % (inst_ip, vip))
        route = {'name': rt_name,
            'destRange': vip, 'network': network,
```

```
            'nextHopIp': inst_ip}
        result = gcp.routes().insert(project=project,
                                     body=route).execute()
        print(('Route VIP %s insert result %s' %
               (vip, str(result))))

def handle_cc_alert(session, gcp, script_parms):
    se_name = script_parms['obj_name']
    print(('Event Se %s %s' % (se_name, str(script_parms))))
    if len(script_parms['events']) == 0:
        print ('WARNING: No events in alert')
        return

    # GET SE object from Avi for instance IP address and SE Group link
    rsp = session.get('serviceengine?uuid=%s' %
                      script_parms['events'][0]['event_details']['cc_ip_details']['se_vm_uuid'])
    if rsp.status_code in range(200, 299):
        se = json.loads(rsp.text)
        if se['count'] == 0 or len(se['results']) == 0:
            print(('WARNING: SE %s no results' %
                script_parms['events'][0]['event_details']['cc_ip_details']['se_vm_uuid']))
            return
        inst_ip = next((v['ip']['ip_addr']['addr'] for v in
                se['results'][0]['mgmt_vnic']['vnic_networks']
                if v['ip']['mask'] == 32 and v['mode'] != 'VIP'), '')
        if not inst_ip:
            print(('WARNING: Unable to find IP with mask 32 SE %s' % str(se['results'][0])))
            return

        # GET SE Group object for GCP project, zones and network
        # https://localhost/api/serviceenginegroup/serviceenginegroup-99f78850-4d1f-4b7b-9027-311ad1f8c60e
        seg_ref_list = se['results'][0]['se_group_ref'].split('/api/')
        seg_rsp = session.get(seg_ref_list[1])
        if seg_rsp.status_code in range(200, 299):
            vip = '%s/32' % script_parms['events'][0]['event_details']['cc_ip_details']['ip']['addr']
            seg = json.loads(seg_rsp.text)
            descr = json.loads(seg.get('description', '{}'))
            project = descr.get('project', '')
            network = descr.get('network', '')
            if not project or not network:
                print(('WARNING: Project, Network is required descr %s' %
                      str(descr)))
                return
            gcp_program_route(gcp, script_parms['events'][0]['event_id'],
                              project, network, inst_ip, vip)
        else:
            print(('WARNING: Unable to retrieve SE Group %s status %d' %
                  (se['results'][0]['se_group_ref'], seg_rsp.status_code)))
            return
    else:
        print(('WARNING: Unable to retrieve SE %s' %
```

```
                script_parms['events'][0]['obj_uuid']))


# Script entry

if __name__ == "__main__":
    script_parms = parse_avi_params(sys.argv)
    try:
        admin_session = create_avi_endpoint()
        gcp = google_compute()
        handle_cc_alert(admin_session, gcp, script_parms)
    except Exception:
        print(('WARNING: Exception with Avi/Gcp route %s' %
               traceback.format_exc()))
```

Denial of Service Attack Handling

```
#!/usr/bin/python
import sys, os, json
from avi.sdk.avi_api import ApiSession


'''
This control script will be executed in the Avi Controller when an
alert due to a DOS_ATTACK event is generated.

An example params passed to the control script dos-attack.py is as follows

params = [u'/home/admin/Dos_Attack-l4vs',
           '{"name": "Dos_Attack-virtualservice-d1093604-e1f0-476a-ad91-01c5224c5641-1461261720.83-1461261716-77911185"
             "throttle_count": 0,
             "level": "ALERT_HIGH",
             "reason": "threshold_exceeded",
             "obj_name": "l4vs",
             "threshold": 1,
             "events":
             [
                 {
                      "event_id": "DOS_ATTACK",
                      "event_details":
                      {
                          "dos_attack_event_details":
                          {
                              "attack_count": 2150.0,
                              "attack": "SYN_FLOOD",
                              "ipgroup_uuids": [
                                  "ipaddrgroup-f6883289-39fa-418f-94c2-3b8f8093cd7a"
                               ],
                               "src_ips": ["10.10.90.67"]
                          }
```

```
                    },
                    "obj_uuid": "virtualservice-d1093604-e1f0-476a-ad91-01c5224c5641",
                    "obj_name": "l4vs",
                    "report_timestamp": 1461261716
                }
             ]
            }'
         ]
The DOS_ATTACK event was generated due to a SYN_FLOOD from client 10.10.90.67. It was
traffic to the Virtual Service : "l4vs".

The offending client ip is added as NETWORK_SECURITY_POLICY_ACTION_TYPE_DENY in the
network seurity policy for the virtual service

'''

def ParseAviParams(argv):
    if len(argv) != 2:
        return
    alert_dict = json.loads(argv[1])
    return alert_dict

def create_avi_endpoint():
    token=os.environ.get('API_TOKEN')
    user=os.environ.get('USER')
    # tenant=os.environ.get('TENANT')
    return ApiSession.get_session("localhost", user, token=token,
                                  tenant='admin')

def add_ns_rules_dos(session, dos_params):
    vs_name = dos_params['obj_name']
    vs_uuid = ''
    client_ips = []
    vs_name = dos_params['obj_name']
    for event in dos_params['events']:
        vs_uuid = event['obj_uuid']
        dos_attack_event_details = event['event_details']['dos_attack_event_details']
        if dos_attack_event_details['attack'] != 'SYN_FLOOD':
            continue
        for ip in dos_attack_event_details['src_ips']:
            client_ips.append(ip)
    if len(client_ips) == 0:
        print ('DOS ATTACK is not SYN_FLOOD. Ignoring')
        return

    print('VS name : ' + vs_name + ' VS UUID : ' + vs_uuid + ' Client IPs : ' + str(client_ips))
    ip_list = []
    for ip in client_ips:
        ip_addr_obj = {
            'addr' : ip,
            'type' : 'V4'
```

```
        }
        ip_list.append(ip_addr_obj)

    match_obj = {
        'match_criteria' : 'IS_IN',
        'addrs' : ip_list
    }
    ns_match_target_obj = {
        'client_ip' : match_obj
    }
    ns_rule_dos_obj = {
        'enable' : True,
        'log'    : True,
        'match'  : ns_match_target_obj,
        'action' : 'NETWORK_SECURITY_POLICY_ACTION_TYPE_DENY'
    }
    ns_policy_dos_obj = {
        'vs_name' : vs_name,
        'vs_uuid' : vs_uuid,
        'rules'   : [
            ns_rule_dos_obj,
        ]
    }
    print('ns_policy_dos_obj : ' + str(ns_policy_dos_obj))
    try :
        session.post(path='networksecuritypolicydos?action=block',
                 data=ns_policy_dos_obj)
    except Exception as e:
        print(str(e))
    print(('Added Client IPs ' + str(client_ips) + \
           ' in the blocked list for VS : ' + vs_name))

if __name__ == "__main__":
    alert_dict = ParseAviParams(sys.argv)
    try :
        admin_session = create_avi_endpoint()
    except Exception as e:
        print('login failed to Avi Controller!' + str(e))
        sys.exit(0)
    add_ns_rules_dos(admin_session, alert_dict)
```