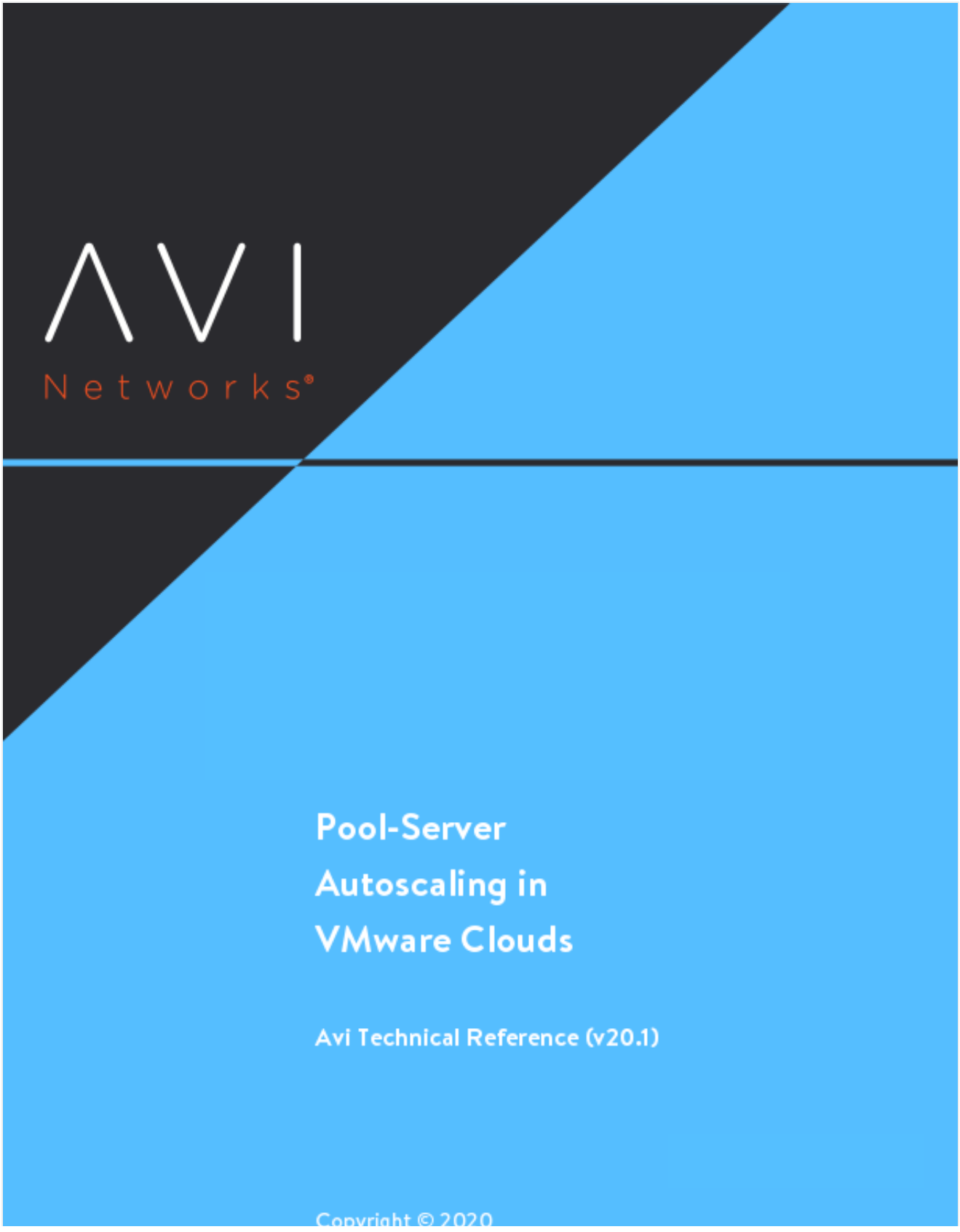
AVI
Networks®
Pool-Server
Autoscaling in
VMware Clouds
Avi Technical Reference (v20.1)
Copyright © 2020

Pool-Server Autoscaling in VMware Clouds

view online

Overview

From Avi Vantage version 18.1.5 onwards, the pyVmomi VMware automation library is available on the Avi Controller for use by ControlScripts. pyVmomi is the Python SDK for the VMware vSphere API that supports management of ESX, ESXi, and vCenter.

One common use of such automation is for back-end, pool-server autoscaling. The steps outlined in this article enable automatic scale in and scale out of pool servers, based on an autoscaling policy.

Configuring Pool-Server Autoscaling in VMware Clouds

To configure scale out for a server pool, the following configuration is required. All the below configuration steps should be carried out in the same tenant as the pool. 1. Create ControlScripts 2. Define autoscaling alerts 3. Attach the autoscaling alerts to an autoscaling policy for the server pool 4. Create scale-out and scale-in alerts triggered on the Server Autoscale Out /Server Autoscale In events to run the relevant ControlScripts

Details for these steps are as follows:

Step 1: Create ControlScripts

Name: Scaleout-Action

```
#!/usr/bin/python
import sys
from avi.vmware_ops.vmware_scale import scale_out

vmware_settings = {
    'vcenter': '10.10.10.10',
    'user': 'root',
    'password': 'vmware',
    'cluster_name': 'Cluster1',
    'template_name': 'web-app-template',
    'template_folder_name': 'Datacenter1/WebApp/Template',
    'customization_spec_name': 'web-app-centos-dhcp',
    'vm_folder_name': 'Datacenter1/WebApp/Servers',
    'resource_pool_name': None,
    'port_group': 'WebApp'}

scale_out(vmware_settings, *sys.argv)
```

The `vmware_settings` parameters are as follows:

- `vcenter` ? IP address or hostname of vCenter.
- `user` ? Username for vCenter access.
- `password` ? Password for vCenter access.

- `cluster_name` ? vCenter cluster name.
- `template_folder_name` ? Folder containing template VM, for instance, Datacenter1/Folder1/Subfolder1 or None to search all (likely more efficient to specify a folder if there are a large number of VMs).
- `template_name` ? Name of template VM.
- `vm_folder_name` ? Folder name to place new VM (or None for same as template).
- `customization_spec_name` ? Name of a customization spec to use - DHCP should be used to allocate IP addresses to the newly-created VMs.
- `resource_pool_name` ? Name of VMware Resource Pool or None for no specific resource pool (VM will be assigned to the default "hidden" resource pool).
- `port_group` ? Name of port group containing pool member IP (useful if the VM has multiple vNICs) or None to use the IP from the first vNIC.

Name: Scalein-ActionName

```
#!/usr/bin/python
import sys
from avi.vmware_ops.vmware_scale import scale_out

vmware_settings = {
    'vcenter': '10.10.10.10',
    'user': 'root',
    'password': 'vmware',
    'vm_folder_name': 'Datacenter1/WebApp/Servers'}

scale_in(vmware_settings, *sys.argv)
```

Step 2: Define Autoscaling Alerts

These alerts define the metric trigger thresholds for a server autoscaling decision to be made. Note that these alerts do not run the scale in/out ControlScripts directly. They are used in an autoscaling policy, along with the parameters on min/max servers and the number of servers to increment/decrement to generate a Server Autoscale Out or Server Autoscale In event. Refer to STEP 3.

The action on these alerts should not run the above ControlScripts, but could be set to generate informational e-mails/syslogs /traps, etc. For example:

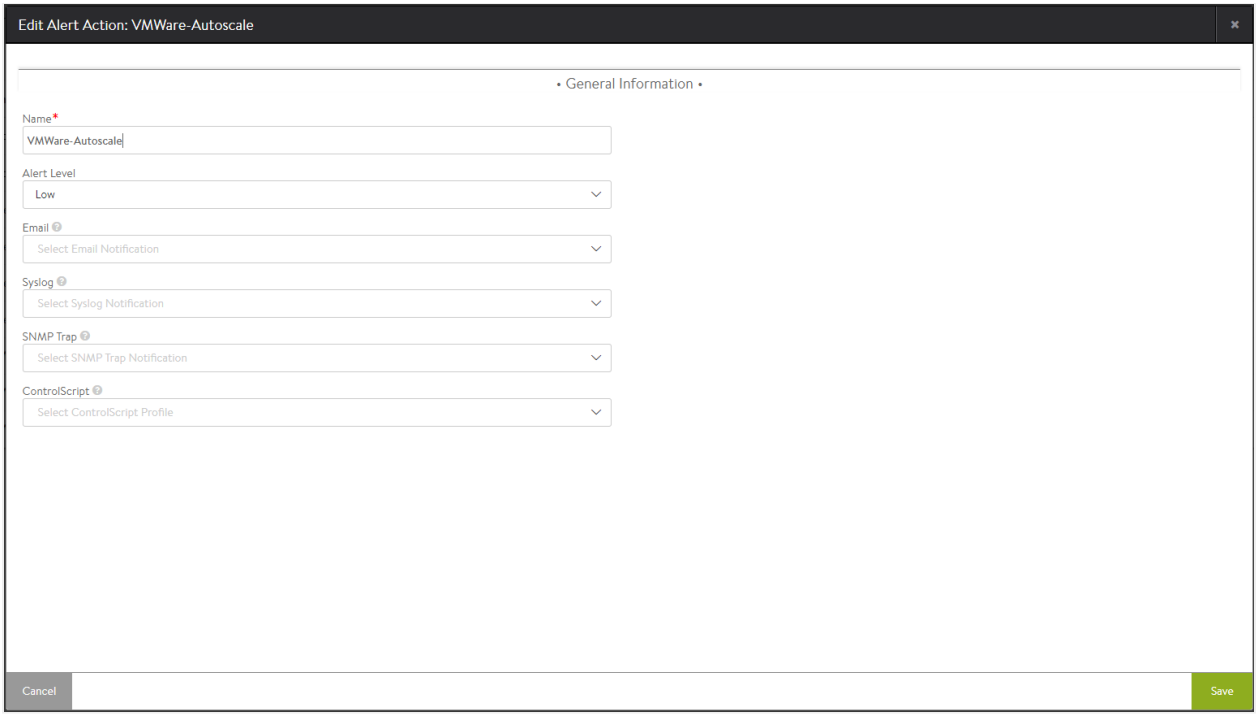

Figure 1. Alert Action Editor

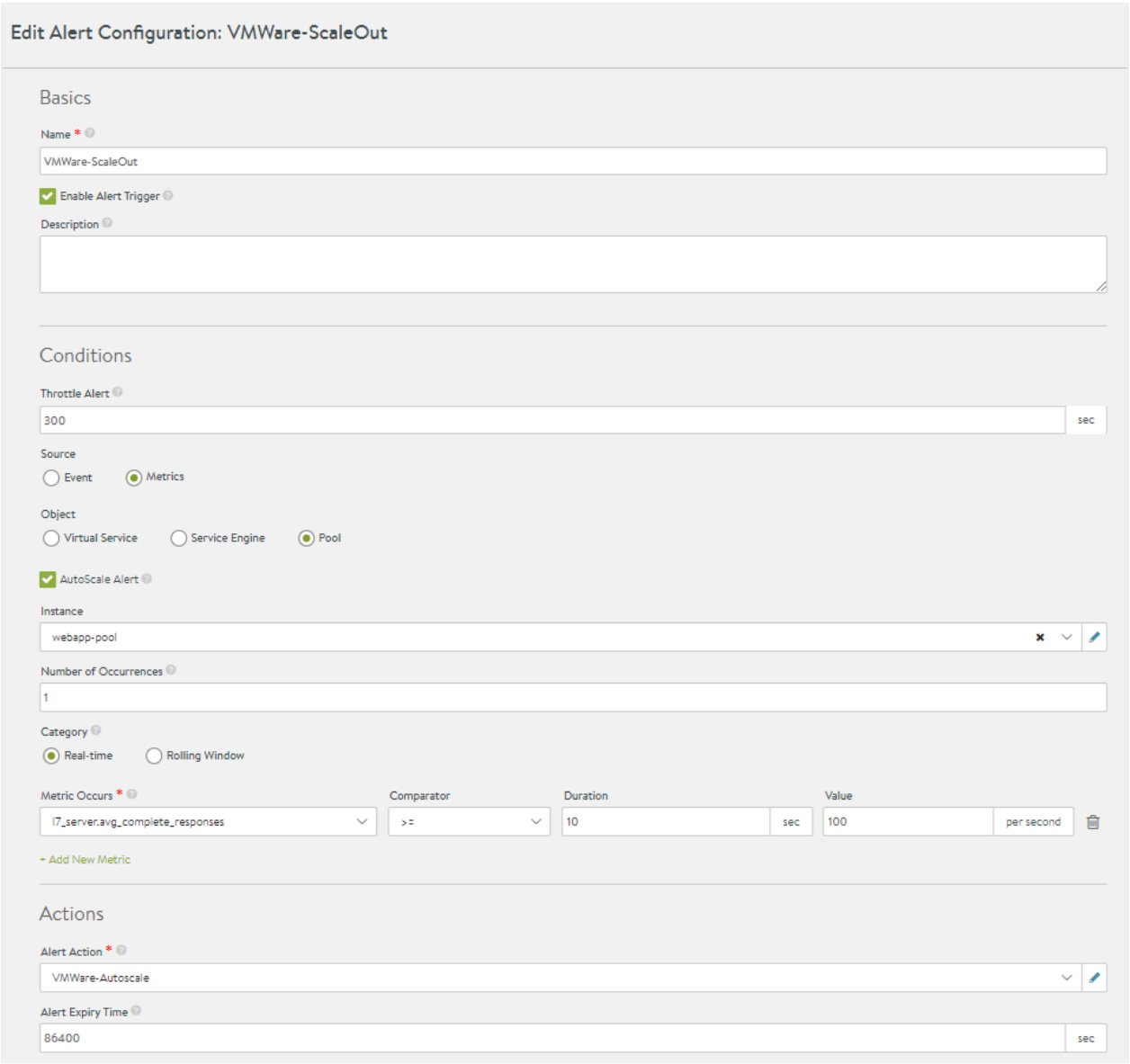

Figure 2. Alert Configuration for "VMWare-ScaleOut"

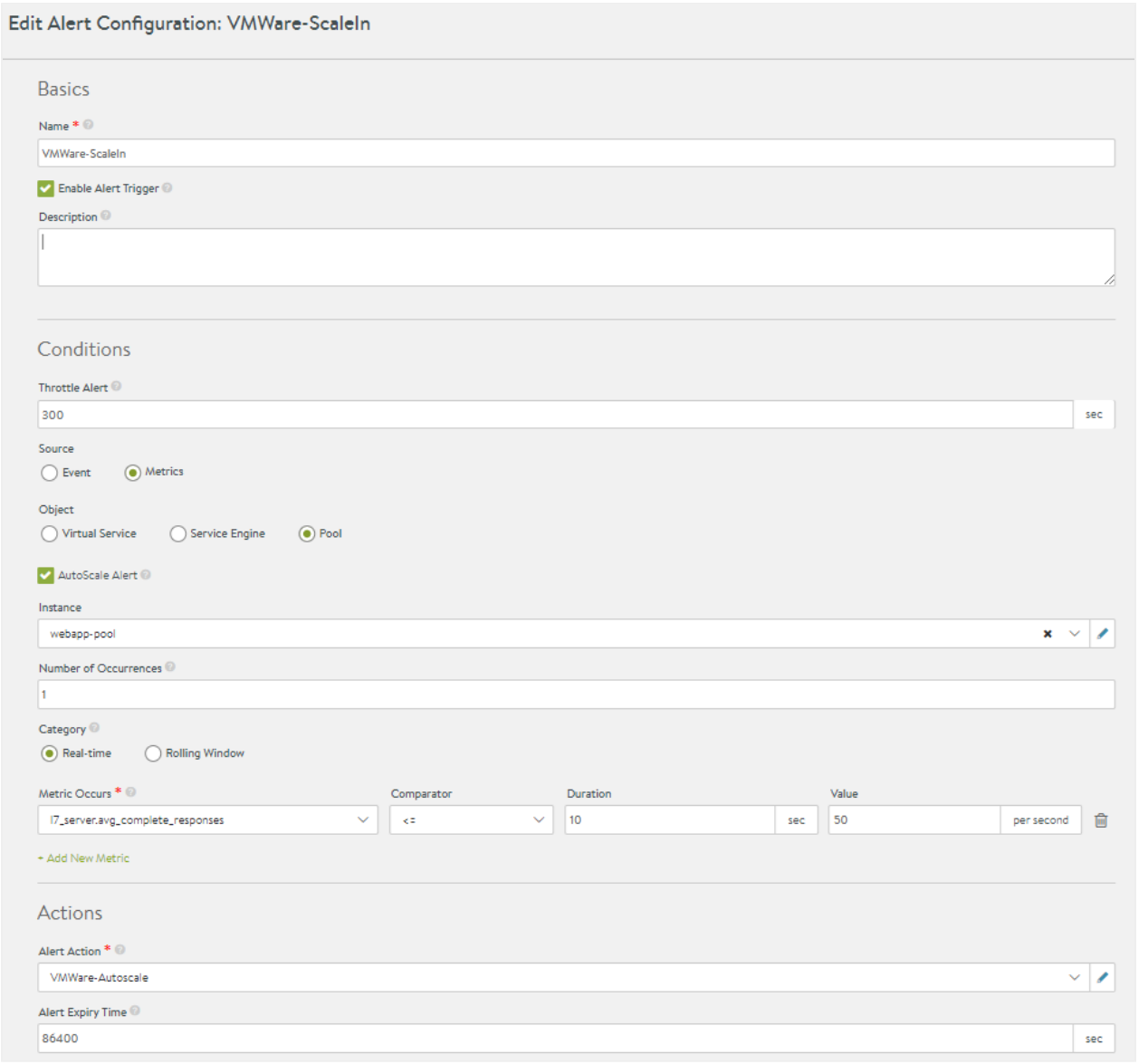

Figure 3. Alert Configuration for "VMWare-ScaleIn"

Step 3: Attach Autoscaling Alerts to an Autoscaling Policy for the Server Pool

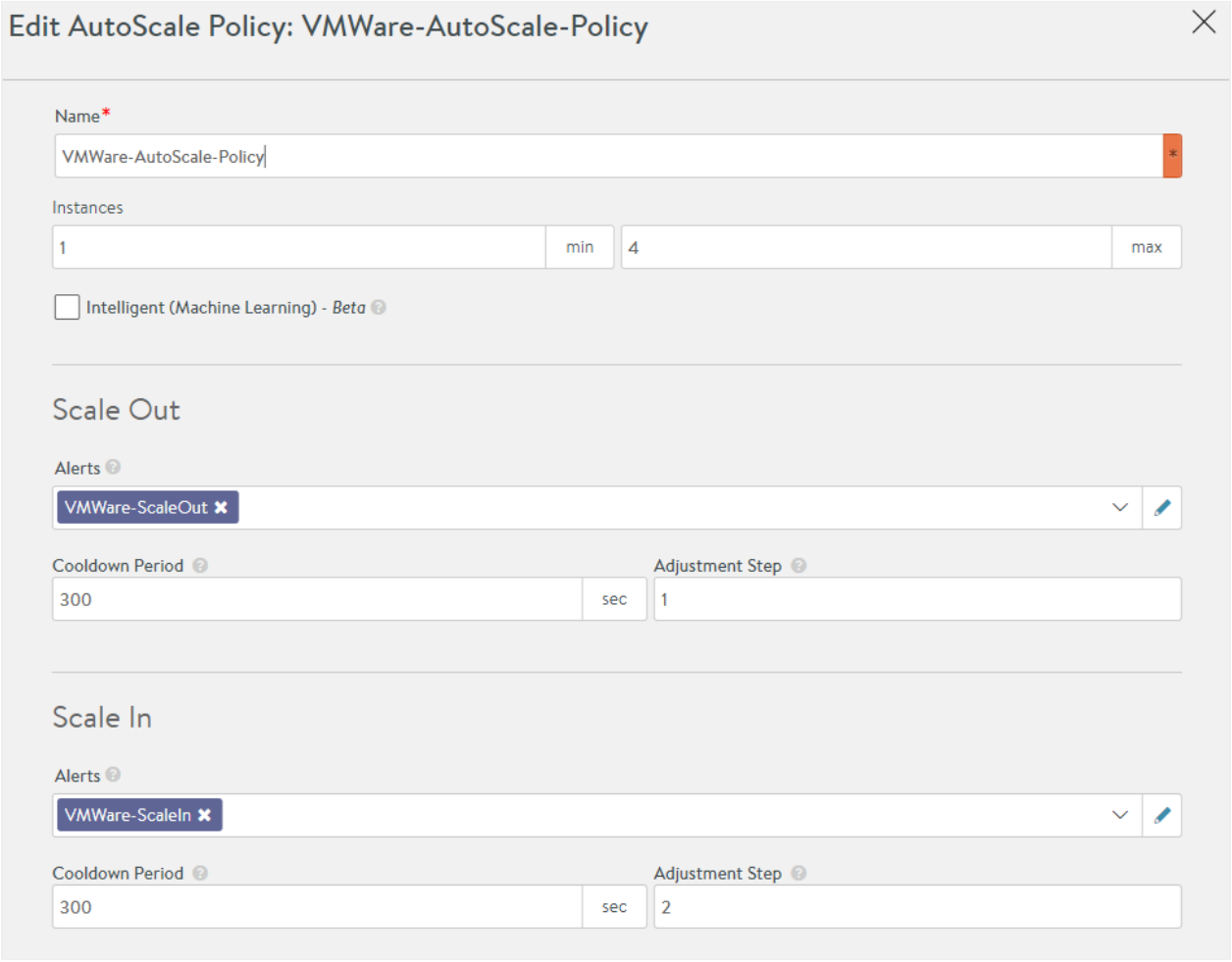

Figure 4. Autoscale policy editor

Step 4: Create Scale-out and Scale-in Alerts Triggered on Events to Run ControlScripts

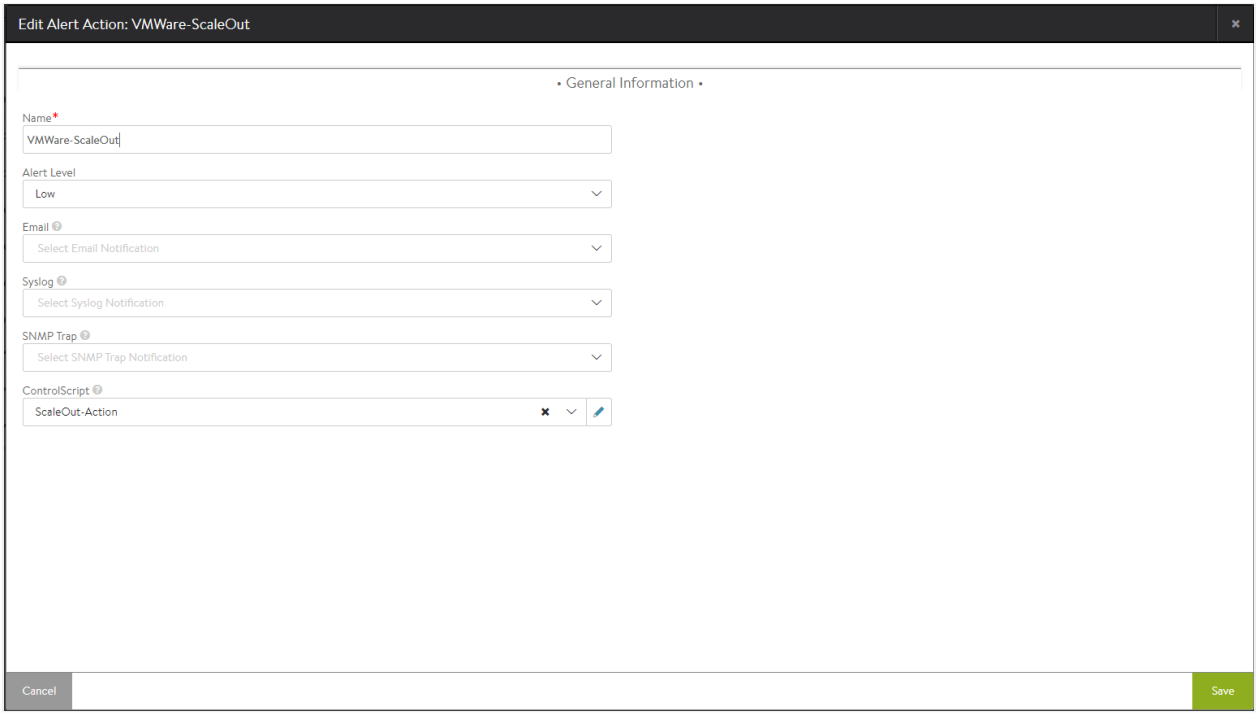

Figure 5. Associating the ScaleOut-Action ControlScript to an alert action

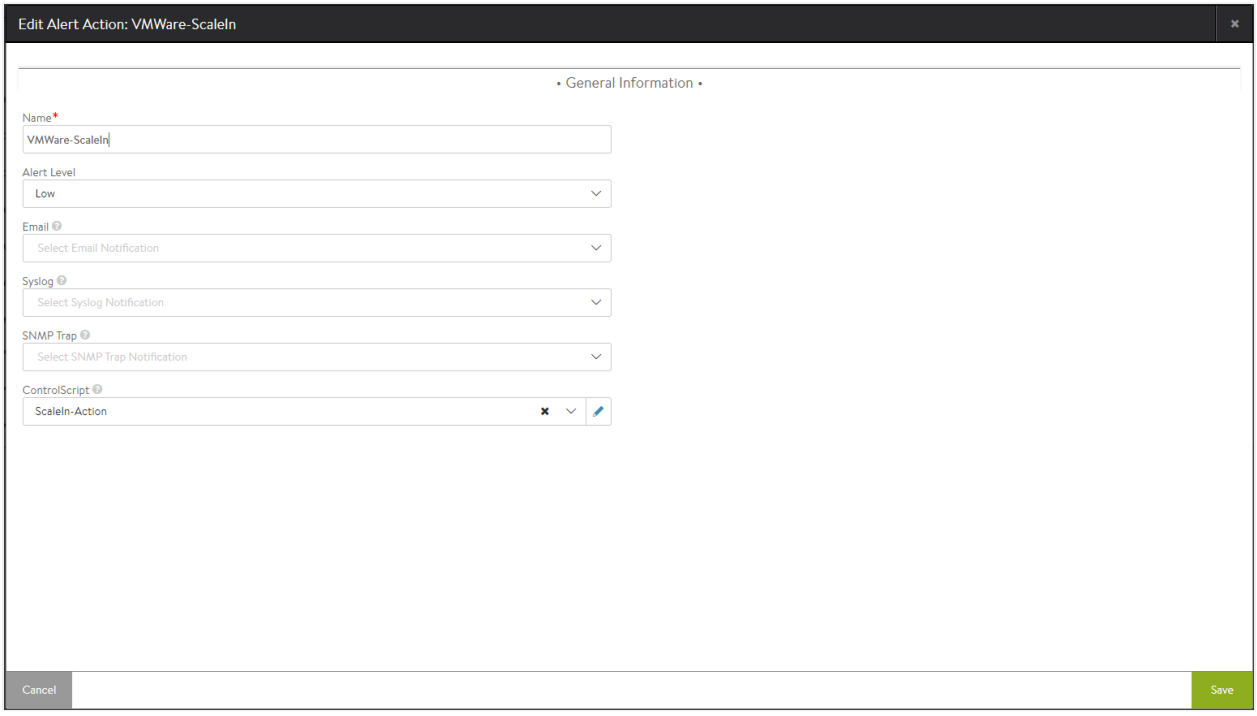

Figure 6. Associating the ScaleIn-Action ControlScript to an alert action

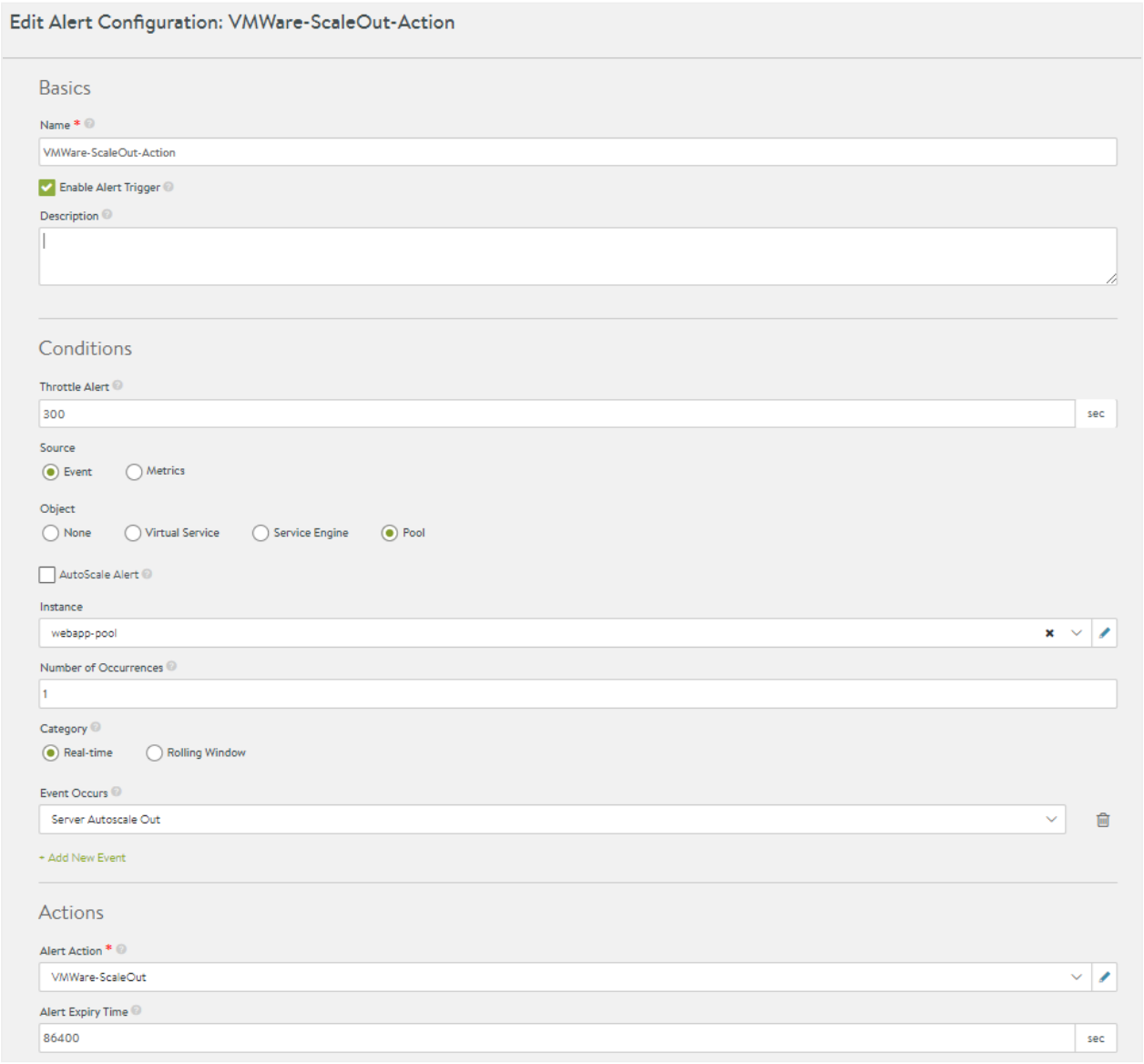

Figure 7. Establishing conditions to trigger the action named "VMWare-ScaleOut-Action"

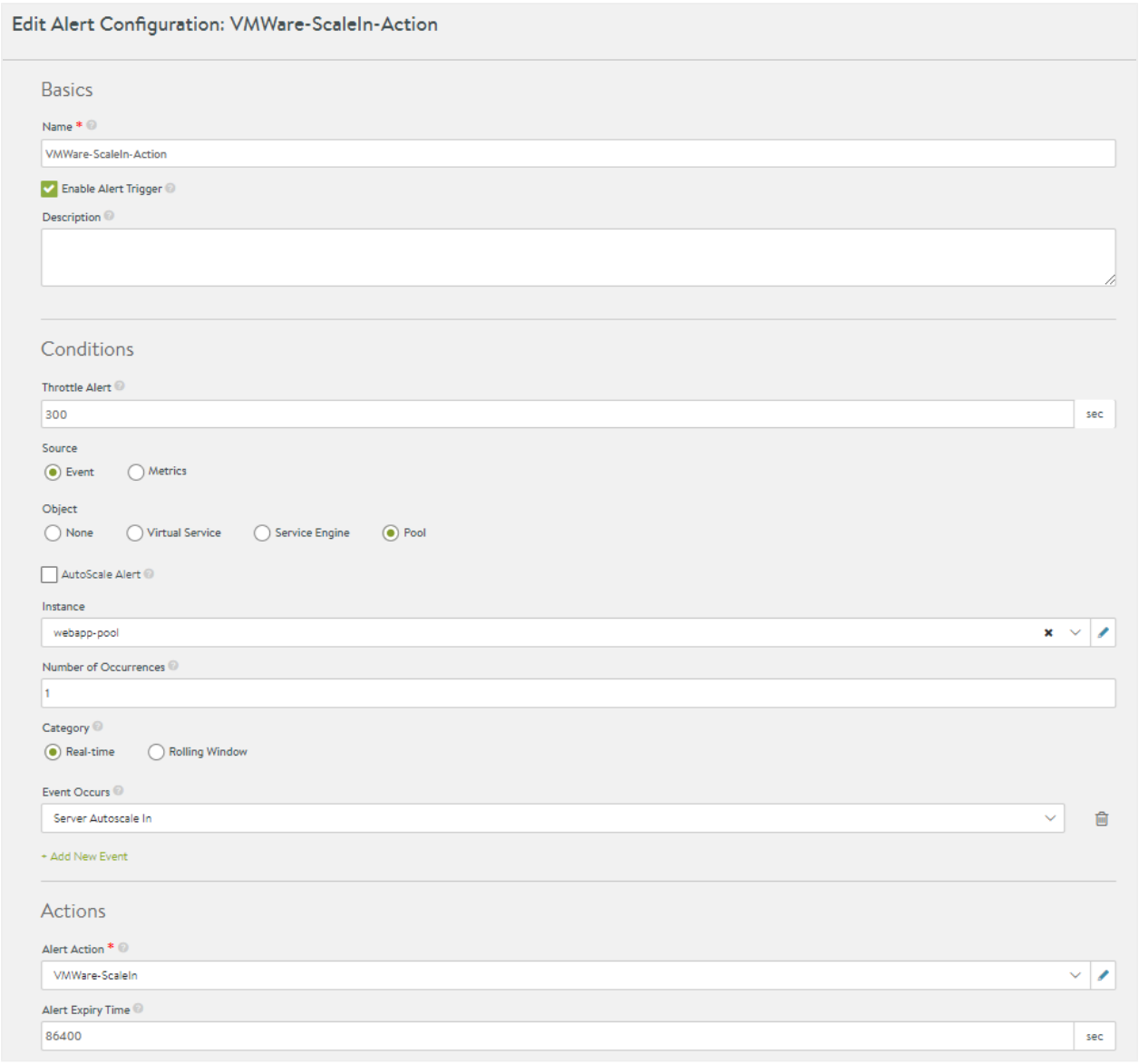

Figure 8. Establishing conditions to trigger the action named "VMWare-ScaleIn-Action"

Python Utility Scripts

The following Python utility scripts should be copied to the Controller to be available for the ControlScripts to use. Create a directory named `vmware_scale` under `/opt/avi/python/lib` and create the following two files (`sudo` required):

`vmutils.py`

```
from pyVmomi import vim
from pyVim.connect import SmartConnectNoSSL, Disconnect
import time
```

```
def _get_obj(content, vimtype, name, folder=None):
    """
    Get the vsphere object associated with a given text name
    """
    obj = None
    container = content.viewManager.CreateContainerView(
        folder or content.rootFolder, vimtype, True)
    for c in container.view:
        if c.name == name:
            obj = c
            break
    return obj

def _get_child_folder(parent_folder, folder_name):
    obj = None
    for folder in parent_folder.childEntity:
        if folder.name == folder_name:
            if isinstance(folder, vim.Datacenter):
                obj = folder.vmFolder
            elif isinstance(folder, vim.Folder):
                obj = folder
            else:
                obj = None
            break
    return obj

def get_folder(si, name):
    subfolders = name.split('/')
    parent_folder = si.RetrieveContent().rootFolder
    for subfolder in subfolders:
        parent_folder = _get_child_folder(parent_folder, subfolder)
        if not parent_folder:
            break
    return parent_folder

def get_vm_by_name(si, name, folder=None):
    """
    Find a virtual machine by its name and return it
    """
    return _get_obj(si.RetrieveContent(), [vim.VirtualMachine], name,
                    folder)

def get_resource_pool(si, name, folder=None):
    """
    Find a resource pool by its name and return it
    """
    return _get_obj(si.RetrieveContent(), [vim.ResourcePool], name,
                    folder)

def get_cluster(si, name, folder=None):
    """
```

```
    Find a cluster by it's name and return it
    """
    return _get_obj(si.RetrieveContent(), [vim.ComputeResource], name,
                    folder)

def wait_for_task(task, timeout=300):
    """
    Wait for a task to complete
    """
    timeout_time = time.time() + timeout
    timedout = True
    while time.time() < timeout_time:
        if task.info.state == 'success':
            return (True, task.info.result)
        if task.info.state == 'error':
            return (False, None)
        time.sleep(1)
    return (None, None)

def wait_for_vm_status(vm, condition, timeout=300):
    timeout_time = time.time() + timeout
    timedout = True
    while timedout and time.time() < timeout_time:
        if (condition(vm)):
            timedout = False
        else:
            time.sleep(3)

    return not timedout

def net_info_available(vm):
    return (vm.runtime.powerState == vim.VirtualMachinePowerState.poweredOn
            and
            vm.guest.toolsStatus == vim.vm.GuestInfo.ToolsStatus.toolsOk
            and
            vm.guest.net)

class vcenter_session:
    def __enter__(self):
        return self.session
    def __init__(self, host, user, pwd):
        session = SmartConnectNoSSL(host=host, user=user, pwd=pwd)
        self.session = session
    def __exit__(self, type, value, traceback):
        if self.session:
            Disconnect(self.session)
```

vmware_scale.py

```
from __future__ import print_function
from pyVmomi import vim
from avi.sdk.samples.autoscale import vmutils
from avi.sdk.samples.autoscale.samplescaleout import scaleout_params
import uuid
from avi.sdk.avi_api import ApiSession
import json
import os

def getAviApiSession(tenant='admin', api_version=None):
    """
    Create local session to avi controller
    """
    token = os.environ.get('API_TOKEN')
    user = os.environ.get('USER')
    tenant = os.environ.get('TENANT')
    api = ApiSession.get_session("localhost", user, token=token,
                                 tenant=tenant, api_version=api_version)
    return api, tenant

def do_scale_in(vmware_settings, instance_names):
    """
    Perform scale in of pool.
    vmware_settings:
        vcenter: IP address or hostname of vCenter
        user: Username for vCenter access
        password: Password for vCenter access
        vm_folder_name: Folder containing VMs (or None for same as
            template)
    instance_names: Names of VMs to destroy
    """

    with vmutils.vcenter_session(host=vmware_settings['vcenter'],
                                 user=vmware_settings['user'],
                                 pwd=vmware_settings['password']) as session:

        vm_folder_name = vmware_settings.get('vm_folder_name', None)

        if vm_folder_name:
            vm_folder = vmutils.get_folder(session, vm_folder_name)
        else:
            vm_folder = None

        for instance_name in instance_names:
            vm = vmutils.get_vm_by_name(session, instance_name, vm_folder)
            if vm:
                print('Powering off VM %s...' % instance_name)
                power_off_task = vm.PowerOffVM_Task()
```

```
                (power_off_task_status,
                 power_off_task_result) = vmutils.wait_for_task(
                     power_off_task)
                if power_off_task_status:
                    print('Deleting VM %s...' % instance_name)
                    destroy_task = vm.Destroy_Task()
                    (destroy_task_status,
                     destroy_task_result) = vmutils.wait_for_task(
                         destroy_task)
                    if destroy_task_status:
                        print('VM %s deleted!' % instance_name)
                    else:
                        print('VM %s deletion failed!' % instance_name)
                else:
                    print('Unable to power off VM %s!' % instance_name)
            else:
                print('Unable to find VM %s!' % instance_name)

def do_scale_out(vmware_settings, pool_name, num_scaleout):
    """
    Perform scale out of pool.
    vmware_settings:
        vcenter: IP address or hostname of vCenter
        user: Username for vCenter access
        password: Password for vCenter access
        cluster_name: vCenter cluster name
        template_folder_name: Folder containing template VM, e.g.
            'Datacenter1/Folder1/Subfolder1' or None to search all
        template_name: Name of template VM
        vm_folder_name: Folder to place new VM (or None for same as
            template)
        customization_spec_name: Name of a customization spec to use
        resource_pool_name: Name of VMWare Resource Pool or None for default
        port_group: Name of port group containing pool member IP
    pool_name: Name of the pool
    num_scaleout: Number of new instances
    """

    new_instances = []

    with vmutils.vcenter_session(host=vmware_settings['vcenter'],
                                 user=vmware_settings['user'],
                                 pwd=vmware_settings['password']) as session:

        template_folder_name = vmware_settings.get('template_folder_name',
                                                   None)
        template_name = vmware_settings['template_name']
        if template_folder_name:
            template_folder = vmutils.get_folder(session,
                                                 template_folder_name)
            template_vm = vmutils.get_vm_by_name(
```

```
            session, template_name,
            template_folder)
    else:
        template_vm = vmutils.get_vm_by_name(
            session, template_name)

    vm_folder_name = vmware_settings.get('vm_folder_name', None)
    if vm_folder_name:
        vm_folder = vmutils.get_folder(session, vm_folder_name)
    else:
        vm_folder = template_vm.parent

    csm = session.RetrieveContent().customizationSpecManager
    customization_spec = csm.GetCustomizationSpec(
        name=vmware_settings['customization_spec_name'])

    cluster = vmutils.get_cluster(session,
                                  vmware_settings['cluster_name'])

    resource_pool_name = vmware_settings.get('resource_pool_name', None)

    if resource_pool_name:
        resource_pool = vmutils.get_resource_pool(session,
                                                  resource_pool_name)
    else:
        resource_pool = cluster.resourcePool
    relocate_spec = vim.vm.RelocateSpec(pool=resource_pool)

    clone_spec = vim.vm.CloneSpec(powerOn=True, template=False,
                                  location=relocate_spec,
                                  customization=customization_spec.spec)

    port_group = vmware_settings.get('port_group', None)

    clone_tasks = []

    for instance in range(num_scaleout):
        new_vm_name = '%s-%s' % (pool_name, str(uuid.uuid4()))

        print('Initiating clone of %s to %s' % (template_name,
                                                new_vm_name))

        clone_task = template_vm.Clone(name=new_vm_name,
                                       folder=vm_folder,
                                       spec=clone_spec)
        print('Task %s created.' % clone_task.info.key)
        clone_tasks.append(clone_task)

    for clone_task in clone_tasks:
        print('Waiting for %s...' % clone_task.info.key)
```

```
            clone_task_status, clone_vm = vmutils.wait_for_task(clone_task,
                                                                timeout=600)

            ip_address = None

            if clone_vm:
                print('Waiting for VM %s to be ready...' % clone_vm.name)
                if vmutils.wait_for_vm_status(clone_vm,
                                condition=vmutils.net_info_available,
                                timeout=600):
                    print('Getting IP address from VM %s' % clone_vm.name)
                    for nic in clone_vm.guest.net:
                        if port_group is None or nic.network == port_group:
                            for ip in nic.ipAddress:
                                if '.' in ip:
                                    ip_address = ip
                                    break
                        if ip_address:
                            break
                else:
                    print('Timed out waiting for VM %s!' % clone_vm.name)

                if not ip_address:
                    print('Could not get IP for VM %s!' % clone_vm.name)
                    power_off_task = clone_vm.PowerOffVM_Task()
                    (power_off_task_status,
                     power_off_task_result) = vmutils.wait_for_task(
                         power_off_task)
                    if power_off_task_status:
                        destroy_task = clone_vm.Destroy_Task()
                else:
                    print('New VM %s with IP %s' % (clone_vm.name,
                                                    ip_address))
                    new_instances.append((clone_vm.name, ip_address))
            elif clone_task_status is None:
                print('Clone task %s timed out!' % clone_task.info.key)

    return new_instances

def scale_out(vmware_settings, *args):
    alert_info = json.loads(args[1])
    api, tenant = getAviApiSession()
    (pool_name, pool_uuid,
     pool_obj, num_scaleout) = scaleout_params('scaleout',
                                               alert_info,
                                               api=api,
                                               tenant=tenant)
    print('Scaling out pool %s by %d...' % (pool_name, num_scaleout))

    new_instances = do_scale_out(vmware_settings,
                                 pool_name, num_scaleout)
```

```
    # Get pool object again in case it has been modified
    pool_obj = api.get('pool/%s' % pool_uuid, tenant=tenant).json()

    new_servers = pool_obj.get('servers', [])

    for new_instance in new_instances:
        new_server = {
            'ip': {'addr': new_instance[1], 'type': 'V4'},
            'hostname': new_instance[0]}
        new_servers.append(new_server)
    pool_obj['servers'] = new_servers
    print('Updating pool with %s' % new_server)
    resp = api.put('pool/%s' % pool_uuid, data=json.dumps(pool_obj))
    print('API status: %d' % resp.status_code)

def scale_in(vmware_settings, *args):
    alert_info = json.loads(args[1])
    api, tenant = getAviApiSession()
    (pool_name, pool_uuid,
     pool_obj, num_scaleout) = scaleout_params('scalein',
                                               alert_info,
                                               api=api,
                                               tenant=tenant)

    remove_instances = [instance ['hostname'] for instance in
                        pool_obj['servers'][-num_scaleout:]]
    pool_obj['servers'] = pool_obj['servers'][:-num_scaleout]
    print('Scaling in pool %s by %d...' % (pool_name, num_scaleout))
    resp = api.put('pool/%s' % pool_uuid, data=json.dumps(pool_obj))
    print('API status: %d' % resp.status_code)

    do_scale_in(vmware_settings, remove_instances)
```