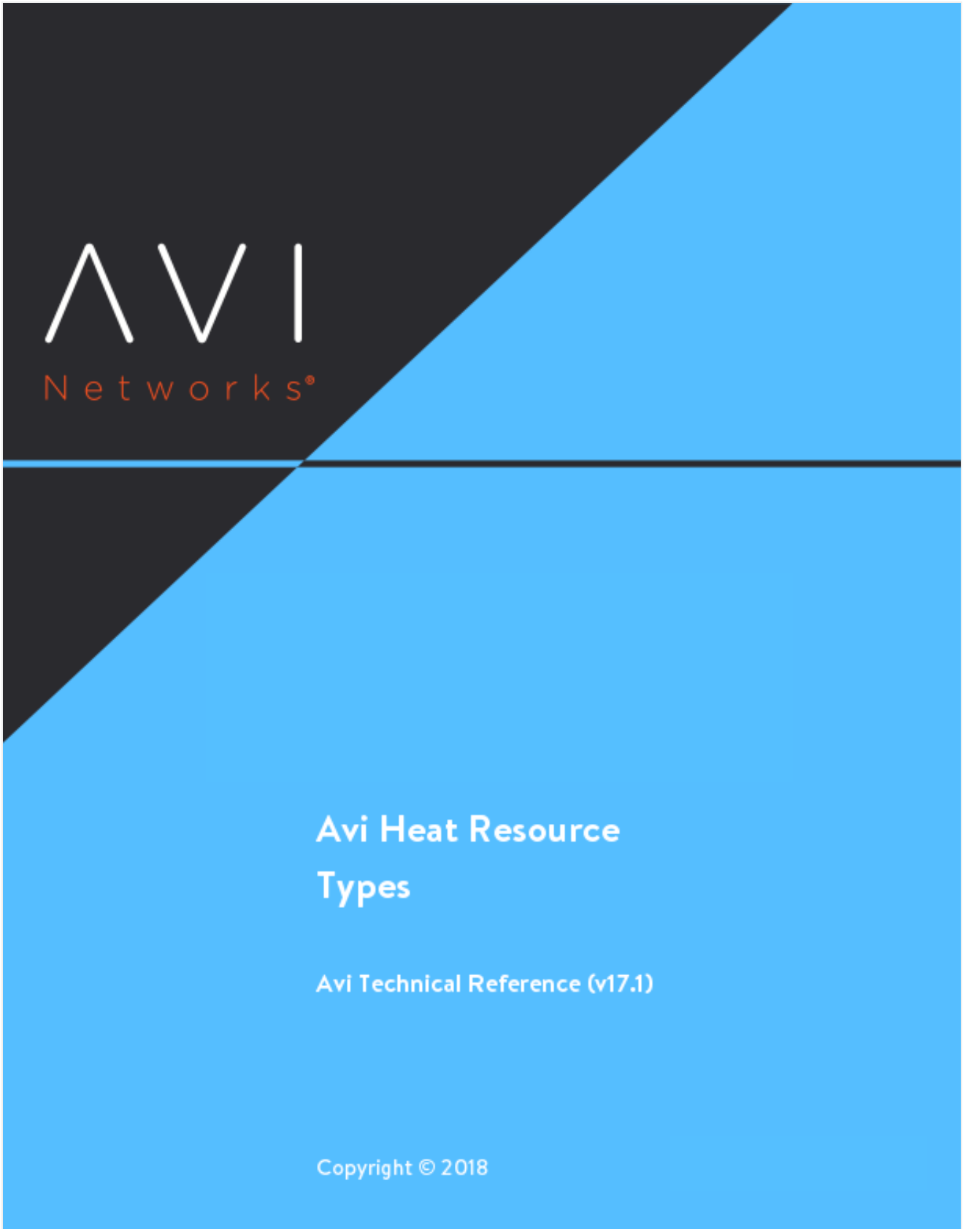

Avi Heat Resource Types

Avi Technical Reference (v17.1)

Avi Heat Resource Types

view online

This article cites benefits of using OpenStack Heat to describe an application via a template. Examples using Avi Heat resource types are included.

Introduction to OpenStack Heat

?The mission of the OpenStack Orchestration program is to create a human- and machine-accessible service for managing the entire lifecycle of infrastructure and applications within OpenStack clouds.? (Source: https://wiki.openstack.org/wiki/Heat)

All OpenStack services such as Nova, Neutron, and Glance are API-driven. Though using APIs is the most basic way to obtain a service, APIs work on the granularity of individual objects. And, as any real usage of an OpenStack public cloud involves creating several objects on different services, using APIs directly for orchestration is a tedious process. For example, to create a typical three-tier application, one has to upload their application images to Glance, create storage volumes in Cinder, create different virtual networks for each tier using Neutron APIs, create appropriate security groups using Nova or Neutron APIs, and create virtual machines using Nova APIs. In addition, one may need a floating IP for the application and hence would need to make another Neutron API call.

Creation is tedious also because some of those API calls depend on the resources created by other API calls. For example, Glance image upload has to succeed before VMs can be created. And the UUID of the Glance image is needed for the VM creation. Another example is that the IP address of the front-end server VM is needed to create the floating IP association.

Updates are even worse, because you need to know the exact sequence of all objects to update. Suppose you want to use a different volume for one of your VMs. Then you need to remove the previous VM and create a new VM with that new volume. And you may need to update the floating IP if that VM is the front-end server.

Deletion requires several API calls which need to be done in the reverse order of creation to ensure no dependencies are violated.

Dealing with all objects as a single entity makes the orchestration much simpler, enabling creation or deletion of an entire application ?stack? with one command.

To that end, Heat service allows users to describe their application as a Heat template with

```
<li style="font-weight: 400;"><span style="font-weight: 400;">Support for create, update, and delete actions</s
<li style="font-weight: 400;"><span style="font-weight: 400;">Ability to operate on a whole-stack level, rather
<li style="font-weight: 400;"><span style="font-weight: 400;">Easy deletion of all elements in the stack becaus
<li style="font-weight: 400;"><span style="font-weight: 400;">Easy update and state-tracking for all elements i
```

Heat stacks are stateful, and therefore always track the current state. And any changes to the stack spec can be automatically applied by the heat-engine since it can determine the appropriate steps needed to go from a previous state to the next state.

In contrast, with stateless orchestration systems such as Ansible, one needs to remove an object explicitly (e.g., setting ? state=absent? in the YAML spec file). Then one has to remove the object/resource from their spec file so that further runs don?t try to delete it again or to save the clutter in the spec file.

Downsides

You can?t use Heat resources to manage pre-existing objects. Only those resources created via a stack can be managed by the heat-engine.

Example of a Heat Stack

Below we create a network, subnet, router, create web server, and associate a floating IP address.

```
heat_template_version: 2015-04-30

description: Web service with its own virtual network template

parameters:
  external_net:
    type: string
    label: External Net
    description: UUID of the external network for Internet access

resources:
  my_net:
        type: OS::Neutron::Net
        properties:
          name: my-net

  my_subnet:
        type: OS::Neutron::Subnet
        properties:
          network_id: { get_resource: my_net }
          cidr: 10.10.2.0/24
          gateway_ip: 10.10.2.1

  my_router:
        type: OS::Neutron::Router
        properties:
          external_gateway_info:
                network: { get_param: public_net }

  my_router_interface:
        type: OS::Neutron::RouterInterface
        properties:
          router_id: { get_resource: my_router }
          subnet_id: { get_resource: my_subnet }

  my_instance:
    type: OS::Nova::Server
    properties:
      image: web-server
      flavor: m1.medium
```

```
  floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: { get_param: public_net }

  association:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      floatingip_id: { get_resource: floating_ip }
      port_id: {get_attr: [my_instance, addresses, {get_param: net}, 0, port]}
```

Avi Heat Resource Types

All resource types are exposed via Heat. Therefore, templates can instantiate and manage the life cycle of Avi resources directly via their Heat stacks.
> Note: In mid-2016 the Heat CLI was deprecated. The CLI examples below predate that deprecation. The CLI commands translate as follows:

Heat CLI: `heat resource-type-list | grep -i avi | more`
Now: `openstack orchestration resource type list | grep -i avi | more`

Heat CLI: `heat resource-type-show Avi::LBaaS::Pool | more`
Now: `openstack orchestration resource type show Avi::LBaaS::Pool | more`
One can check the resource types by invoking `heat resource-type-list` and grepping for `Avi::LBaaS::`.

```
user@host:~# heat resource-type-list | grep -i avi | more
...
...
| Avi::LBaaS::Pool                          |
| Avi::LBaaS::Pool::Server                  |
...
...
```

All available options for a resource type can be obtained by `heat resource-type-show <RESOURCE_NAME>`

```
user@host:~# heat resource-type-show Avi::LBaaS::Pool | more
...
...
    "default_server_port": {
      "type": "number",
      "required": false,
      "update_allowed": true,
      "description": "Traffic sent to servers will use this destination server port unless overridden by the server's s
      "immutable": false
    },
...
...
```

Below is an example Heat template that creates a HTTP health monitor, a pool with two web servers (with IP addresses), and a virtual service to load balance traffic against that pool.

```
heat_template_version: 2015-04-30
description: Example template that uses Avi Heat resources
resources:
  hm:
    type: Avi::LBaaS::HealthMonitor
    properties:
      name: "mytesthm"
      receive_timeout: 2
      failed_checks: 2
      successful_checks: 6
      send_interval: 2
      type: HEALTH_MONITOR_HTTP
      http_monitor:
        http_response_code:
          - HTTP_2XX
          - HTTP_3XX
          - HTTP_4XX
        http_request: "GET / HTTP/1.0"
  pool:
    type: Avi::LBaaS::Pool
    properties:
      name: "mytestpool"
      default_server_port: 8092
      health_monitor_uuids:
        - {get_resource: hm}
      servers:
        - ip:
            addr: 10.10.10.10
            type: V4
          port: 8080
        - ip:
            addr: 10.10.10.20
            type: V4
          port: 8081
  vs:
    type: Avi::LBaaS::VirtualService
    properties:
      avi_version: 17.1.1
      name: "mytestvs"
      pool_uuid: {get_resource: pool}
      vip:
        - ip_address:
            addr: 10.10.10.100
            type: V4
          vip_id: myvip
```

```
      services:
        - port: 80
      application_profile_uuid: get_avi_uuid_by_name:System-Secure-HTTP
```

Note that in the above example, servers that are part of a pool are explicitly specified in the template. But in an auto-scaling case, back-end servers are dynamically created and destroyed based on the intensity of the traffic (or other variables). Since several servers may need to be created or destroyed at once and in parallel, updating the pool resource in the above example template is not an option. So, to support parallel addition or deletion of pool members, please use the Avi resource type `Avi::LBaaS::Pool::Member`. A snippet of an example template using that resource type is as follows:

```
heat_template_version: 2015-04-30

description: >
Example template to add a member to an existing pool on Avi. Existing pool is specified using name instead of UUID.

# Make sure that testpool already exists on Avi
resources:
  member:
    type: Avi::LBaaS::Pool::Server
    properties:
      pool_uuid:  get_avi_uuid_by_name:testpool
      ip:
        addr: 10.10.10.50
        type: V4
      port: 8080
```

Obtaining Avi UUIDs by Using Names

For properties that need UUIDs, you can specify a name but prefixed with string `get_avi_uuid_by_name:`. Internally, the heat-engine performs an API call to resolve the provided name into the Avi UUID. The example in the previous section uses this feature to get the UUID of the pool named `testpool`. This avoids the need for users to explicitly lookup the UUID of that pool on Avi in order to specify it in their templates.

Sometimes users may want to provide the name for a UUID property via an input parameter to their template rather than hard-coding it as in the example above. This can be achieved as follows:

```
heat_template_version: 2015-04-30
description: >
  Example template that takes pool_name as an input
  parameter

# Make sure that pool_name provided already exists on Avi.
parameters:
   pool_name:
    type: string
    label: Pool Name

resources:
  member:
```



```
    type: Avi::LBaaS::Pool::Server
    properties:
      pool_uuid:
        str_replace:
          template: get_avi_uuid_by_name:pname
          params:
            pname: { get_param: pool_name }
      ip:
        addr: 10.10.10.50
        type: V4
      port: 8080
```

Versioning

Starting with release 17.1, the Avi API is versioned so that Avi Controllers support API calls with different versions. By default, the heat-engine uses 16.4.2 as the version for all Avi API calls for implementing Avi Heat resource types. A user wishing to use an option in a resource type introduced in a later version (say 17.1.1, as is the case with `vip` field in the longer virtual service example above) needs to explicitly specify the `avi_version` property in the template for that resource.

Avi Resource Types vs. OpenStack LBaaS Resource Types

OpenStack Heat ships with LBaaS v1 (deprecated now) and LBaaS v2 resource types. One can use Avi resource types in templates to leverage the extensive features available in Avi Vantage, especially for the features that are not available via LBaaS.

Installation

The Avi Heat repository is open source and available online on GitHub at https://github.com/avinetworks/avi-heat. Please follow the installation instructions within the README on the main page.