



Heat OpenStack infrastructure orchestration

2024-02-20 | Björn Hagemeier | Juelich Supercomputing Centre

Overview

- What is Heat?
- Guide to documentation
- Resource types
- Internal references
- Parameters
- Writing templates
- Nested templates and stacks

What is Heat

- Heat is a service to **orchestrate composite cloud applications** using a declarative template format through an OpenStack-native REST API
- relationships between resources
- allow creation of **most OpenStack resource types** (such as instances, floating ips, volumes, security groups, users, etc), as well as some more advanced functionality such as instance high availability, instance **autoscaling, and nested stacks**

Documentation

A brief guide

The documentation can be intimidating, these are the most important pointers

- General information: <https://docs.openstack.org/heat/latest/>
- Reference
 - **Template guide (overview):** https://docs.openstack.org/heat/latest/template_guide/index.html
 - Heat Orchestration Template (HOT) guide: https://docs.openstack.org/heat/latest/template_guide/hot_guide.html
 - Heat Orchestration Template specification: https://docs.openstack.org/heat/latest/template_guide/hot_spec.html
 - OpenStack resource types: https://docs.openstack.org/heat/latest/template_guide/openstack.html
- Examples:
 - <https://opendev.org/openstack/heat-templates/>

A simple Heat template

Whetting the appetite

```
heat_template_version: wallaby
```

```
description: Simple template to deploy a single compute instance
```

```
resources:
```

```
  my_instance:
```

```
    type: OS::Nova::Server
```

```
    properties:
```

```
      key_name: my_key
```

```
      image: Ubuntu Jammy 22.04 LTS
```

```
      flavor: t1
```

The Heat Stack

Instantiation and general commands

```
$ openstack stack create -t template.yaml --parameter par=val  
  ↪ --parameter ... stack-name  
$ openstack stack update -t template.yaml --parameter par=val2  
  ↪ --parameter ... stack-name  
$ openstack stack delete stack-name  
$ openstack stack list  
$ openstack stack show stack-name  
$ openstack stack resource list  
$ openstack stack resource show  
$ openstack stack event list stack-name  
  
$ openstack help stack
```

The Heat Stack

Instantiation and general commands

```
$ openstack stack create -t t  
  ↪ --parameter ... stack-name  
$ openstack stack update -t t  
  ↪ --parameter ... stack-name  
$ openstack stack delete stack  
$ openstack stack list  
$ openstack stack show stack-  
$ openstack stack resource li  
$ openstack stack resource sh  
$ openstack stack event list  
  
$ openstack help stack
```

The screenshot displays the HDF OpenStack dashboard. The top navigation bar shows 'Default' and 'admin'. The breadcrumb trail is 'Project / Orchestration / Stacks / hifis-hmgu-reviews'. The left sidebar contains a menu with 'Stacks' highlighted in blue. The main content area shows the 'hifis-hmgu-reviews' stack details, including a 'Topology' tab and a 'Check Complete' button. A network diagram is visible, showing a central node connected to three other nodes.



Main ingredients

Template structure

heat_template_version: 2016-10-14

description:

a description of the template

parameter_groups:

a declaration of input parameter groups and order

parameters:

declaration of input parameters

resources:

declaration of template resources

outputs:

declaration of output parameters

conditions:

declaration of conditions

Template version

- determines the validated and supported features
 - date or codename of the Heat release
 - starting with "Newton", code names are possible
 - "Stein", "Train", "Ussuri" were skipped
 - We are currently running "Yoga" supporting the 2021-04-16 or wallaby version
- 2013-05-23
 - 2014-10-16
 - 2015-04-30
 - 2015-10-15
 - 2016-04-08
 - 2016-10-14 | newton
 - 2017-02-24 | ocata
 - 2017-09-01 | pike
 - 2018-03-02 | queens
 - 2018-08-31 | rocky
 - 2021-04-16 | wallaby

Pseudo parameters

Three additional parameters for use within template

- `OS::stack_name` – Name of the running stack
- `OS::stack_id` – ID of the running stack
- `OS::project_id` – ID of the project under which stack is running



Resource Types

Resource types

- describe infrastructure components such as **server**, **network**, **volume**, etc.
 - resources can be linked to each other depending on the type
 - generic and type-specific properties
- OS::Cinder::Volume
 - OS::Cinder::VolumeAttachment
 - OS::Glance::WebImage
 - OS::Heat::AutoscalingGroup
 - OS::Heat::CloudConfig
 - OS::Heat::Delay
 - OS::Heat::InstanceGroup
 - OS::Neutron::FloatingIP
 - OS::Neutron::FloatingIPAssociation
 - OS::Neutron::Router
 - ...

General structure

```
the_resource:  
  type: <resource type>  
  properties:  
    prop_1: ...  
  metadata: <resource specific metadata>  
  depends_on: <resource ID or list of ID>  
  update_policy: <update policy>  
  deletion_policy: <deletion policy>  
  external_id: <external resource ID>  
  condition: <condition name or expression or boolean>
```

- everything below properties is **optional**
- resource types reference

Nova Server (VM)

OS::Nova::Server

A most basic template comprises

```
---  
heat_template_version: wallaby  
  
resources:  
  server:  
    type: OS::Nova::Server  
    properties:  
      networks:  
        - network: internal  
    image: Debian Bullseye 11  
    flavor: t1  
    key_name: myKey
```

Exercise

Nova Server

- 1 Upload an SSH key
- 2 Create template referencing key
- 3 Start a stack from the template
- 4 Login to your VM
 - `ssh -J 134.94.199.24` ↷
`<your-vms-static-ip>`

Exercise

Nova Server

- 1 Upload an SSH key
 - 2 Create template referencing key
 - 3 Start a stack from the template
 - 4 Login to your VM
 - `ssh -J 134.94.199.24 <your-vms-static-ip>`
- why did this work?
 - network and subnet already exist
 - security group already configured accordingly

Cinder Volume (VM)

OS::Cinder::Volume

```
the_resource:  
  type: OS::Cinder::Volume  
  properties:  
    availability_zone: String  
    backup_id: String  
    description: String  
    image: String  
    metadata: {...}  
    name: String  
    read_only: Boolean  
    scheduler_hints: {...}  
    size: Integer  
    snapshot_id: String  
    source_volid: String  
    volume_type: String
```

Neutron Network

```
the_net:  
  type: OS::Neutron::Net  
  properties:  
    admin_state_up: Boolean  
    availability_zone_hints: [Value, Value, ...]  
    dhcp_agent_ids: [Value, Value, ...]  
    dns_domain: String  
    name: String  
    port_security_enabled: Boolean  
    qos_policy: String  
    shared: Boolean  
    tags: [String, String, ...]  
    tenant_id: String  
    value_specs: {...}
```

Neutron Subnet

the_subnet:

type: OS::Neutron::Subnet

properties:

allocation_pools: [{"start": String, "end": String}, {"start": String, "end": String}]

cidr: String

dns_nameservers: [Value, Value, ...]

enable_dhcp: Boolean

gateway_ip: String

host_routes: [{"destination": String, "nexthop": String}, {"destination": String, "nexthop": String}]

ip_version: Integer

ipv6_address_mode: String

ipv6_ra_mode: String

name: String

network: String

prefixlen: Integer

segment: String

subnetpool: String

tags: [String, String, ...]

tenant_id: String

value_specs: {...}



References

References

References allow to express resource dependencies and link resources among each other.

```
user_data:
  str_replace:
    template: |-
      #cloud-config
      write_files:
        - path: /run/integration-test.sh
          content: |
            ...
            ping -c1 -W 1 other_host ; ping_succ=$?
            if [ $ping_succ -eq 0 -a $ping_ext_succ -eq 0 ]; then
              curl_cli --data-binary '{"status": "SUCCESS", "data": "SUCCESS"}'
            else
              ... // signal failure
            fi
      runcmd:
        - /bin/bash /run/integration-test.sh
  params:
    curl_cli: {get_attr: [wait_cond_handle, curl_cli]}
    other_host: {get_attr: [port_01, fixed_ips, 0, ip_address]}
```



Functions

Functions

- Available functions depend on template version. Typically only additions and no removal with increasing versions.
- Check HOT specification
- Condition functions

```
equals  
get_param  
not  
and  
or  
yaql  
contains
```

```
digest  
filter  
get_attr  
get_file  
get_param  
get_resource  
list_join  
make_url  
list_concat  
list_concat_unique  
contains  
map_merge  
map_replace  
repeat  
resource_facade  
str_replace  
str_replace_strict  
str_replace_vstrict  
str_split  
yaql  
if
```




Nested templates

Nested templates

main.yaml

```
main-server-group:
  type: OS::Heat::ResourceGroup
  properties:
    count: { if: [k8s-join-main-defined, {
  ↪ get_param: main-servers }, 1] }
  resource_def:
    type: lb_server.yaml
    properties:
      cluster_name: { get_param:
  ↪ "OS::stack_name" }
      ...
```

lb_server.yaml

```
parameters:
  cluster_name:
    type: string
resources:
  server:
    type: OS::Nova::Server ...
  pool_member:
    type: OS::Octavia::PoolMember
  properties:
    address: {get_attr: [server, addresses,
  ↪ {get_param: net_1}, 0, addr]}
```

Nested stacks

```
$ openstack stack resource list 5255a289-a098-4721-a66a-4a7f50303b36
+-----+-----+-----+
| resource_name      | physical_resource_id | resource_type          |
+-----+-----+-----+
| kube_cluster_deploy | f7b3def2-...         | OS::Heat::SoftwareDeployment |
| kube_cluster_config | b53a6da4-...         | OS::Heat::SoftwareConfig    |
| kube_minions        | 8e457e7f-...         | OS::Heat::ResourceGroup     |
| api_lb              | ace825da-...         | file:///var/.../lb_api.yaml  |
...
$ openstack stack resource show 5255a289-a098-4721-a66a-4a7f50303b36 api_lb
...
$ openstack stack show ace825da-...
...
$ openstack stack resource show 5255a289-a098-4721-a66a-4a7f50303b36 api_lb -c links -f json
...
```

Stack examination

- Health check
- Walking the resources
- `stack list`
- `stack resource list`
- `stack resource show`
- `stack template show`