The Total Newbie's Introduction to Heat Orchestration in OpenStack

OpenStack is undeniably becoming part of the mainstream cloud computing world. It is emerging as the new standard for private clouds, and justifiably so. OpenStack is powerful, flexible, and is continuously developed. But best of all, it has a very rich API layer. You can do some really interesting things with those APIs, including automating the deployment of a whole stack of infrastructure.

This tutorial is designed to introduce a new user with very basic OpenStack knowledge to the Heat orchestration engine. I will start from very simple elements and work up to a full stack. I will walk through the process in a (hopefully) logical manner, and use lots of examples and screen captures. No special skills are required, but you should at least know how to navigate around the Horizon dashboard.

But first, some background info.

Automated delivery has its limits

As enterprises drive more and more into cloud and on-demand computing, there comes a natural gap in deployment. What used to take weeks with a manual request for a virtual machine can become radically reduced through a simple service catalog and automated deployment. But simply standing up a new virtual machine with an OS alone rarely provides any business value. There is still the often-tedious task of putting the new VM in context: that is to say, adding on a network and some application software, as well as assigning a public IP. These tasks are often completed as manual "post provisioning steps."

IT departments have attempted to solve this issue in many ways. The simplest tactic is to maintain an inventory of images or templates such as a Windows image with SQL Server installed. This solution has the virtue of being simple, but over time creates its own problems. Individual templates need to be patched, in some cases licenses applied, and can quickly fall out of date. In addition, it is not long before a huge library of image templates need be maintained. Moreover, a single server is seldom adequate for an application. So the problem becomes compounded.

What is orchestration, and why you should care

The solution lies in orchestration. Quite simply, orchestration is *the ability to automate the deployment and configuration of infrastructure*. More than just standing up virtual servers, it also manages scripts used to add VMs to networks, stands up multiple servers together as a "stack," and even installs application software.

To put in a practical context, let's say I want to stand up a simple stack consisting of a web server and a database server. I want the database server to have external storage, I want the web server to have a floating IP so that I can access it externally, and I want to do this as easily as possible, with minimal manual entry.

This is not a new problem, and as might be expected there are a variety of potential solutions. Configuration management solutions such as Puppet and Chef allow users to automate configurations through collections of scripts called "manifests" (Puppet) or "recipes" (Chef). Virtualization vendors such as VMware rely on cumbersome Run Book Automation tools (itself a separate application) to automate configuration. Some solutions have even compounded the complexity by running a master orchestrator on top of other orchestration tools, creating the so-called "orchestrator of orchestrators." Unsurprisingly, these solutions are not widely adopted.

What has been widely adopted is the solution provided by Amazon Web Services. AWS addressed this problem by developing "cloud formation" templates for their users. It has the great advantage of being a declarative script, residing in simple (and easily managed) text files.

OpenStack provides a module called **Heat** for orchestration, which is based on the AWS Cloud Formation template. As with other Amazon APIs, OpenStack can even consume and understand AWS Cloud Formation templates. More importantly, users can develop their own templates in Heat, which are simpler and arguably more powerful.

Orchestration is not configuration management

Before we go any further, the question often arises around these lines: "We have already started using Puppet/Chef to do some configurations. Can't we use it?" The answer is yes, but Heat is not Puppet/Chef, and orchestration is not configuration management. While there is certainly some overlap, Heat really shines when it comes to configuring the infrastructure (instances, volumes, floating IPs, etc.) stacks on which applications run. Puppet/Chef really shine when it comes to configuring and updating applications running on that infrastructure. In fact, the stated goal of the team that developed Heat was to avoid competing with configuration management tools.

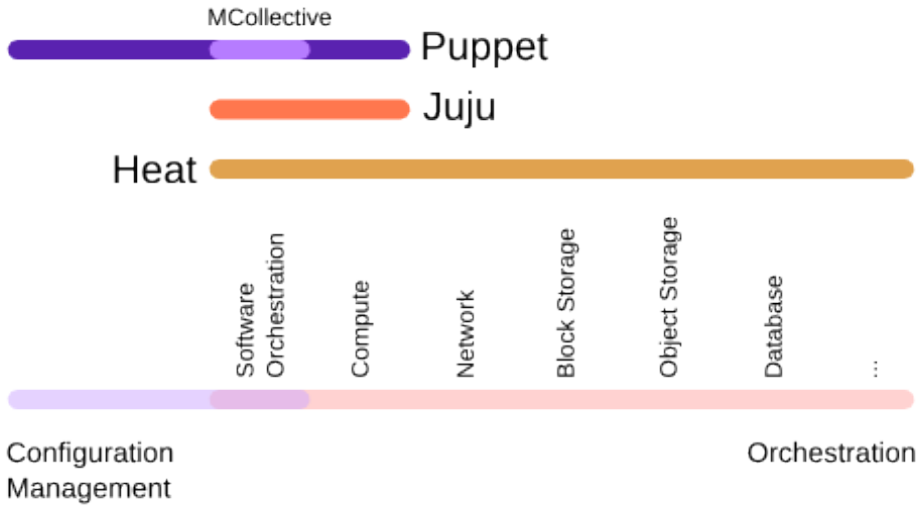

Figure 1. Orchestration vs. Config management. Source - http://www.zerobanana.com/archive/2014/05/08#heat-configuration-management

Orchestration through Heat and configuration management through Puppet or Chef work very well together, and there is no reason not to use both if you have them.

NOTE: For a simple example of calling an external resource like Puppet or Chef inside a Heat template, see the blog post by my colleague Vallard Benincosa https://communities.cisco.com/community/technology/datacenter/cloud-solutions/openstack/cisco-openstack-private-cloud/blog/2015/06/29/openstack-heat-orchestration-with-user-data. He is using the resource "user data" to represent a configuration management tool like Puppet or Chef.

HOT or not

There are two ways to create a template that Heat understands. Both work just fine but are not always interchangeable. "CFN" stands for "CloudFormation" and is a format used in AWS. Heat natively understands this format, simplifying application portability between AWS and OpenStack. CFN templates are usually (but not always) written in JSON language.

The next is called HOT (Heat Orchestration Template). HOT templates are often (but not always) written in YAML format. YAML stands for "Yet Another Markup Language," and is refreshingly easy to read and understand by non-programmers. The simplicity makes HOT templates accessible to system admins, architects, and other non-coders. Unlike CFN, HOT is not backward compatible with AWS. However, it is OpenStack native and meant to replace CFN over time.

Understanding the rules

Some basic terminology is in order to help navigate the YAML structure. Here are the fundamentals:

Stack – This is what we are creating: a collection of VMs and their associated configuration. But in Heat, "Stack" has a very specific meaning. It refers to a collection of resources. Resources could be instances (VMs), networks, security groups, and even auto-scaling rules.

Template –Templates are the design-time definition of the resources that will make up the stack. For example, if you want to have two instances connected on a private network, you will need to define a template for each instance as well as the network. A template is made up of four different sections:

- **Resources**: These are the details of your specific stack. These are the objects that will be created or modified when the template runs. Resources could be Instances, Volumes, Security Groups, Floating IPs, or any number of objects in OpenStack.
- **Properties**: These are specifics of your template. For example, you might want to specify your CentOS instance in m1.small flavor. Properties may be hard coded in the template, or may be prompted as *Parameters*.
- **Parameters**: These are *Properties* values that must be passed when running the Heat template. In HOT format, they appear before the Resources section and are mapped to Properties.
- **Output**: This is what is passed back to the user. It may be displayed in the dashboard, or revealed in command line `heat stack-list` or `heat stack-show`. In this tutorial, we will focus on Horizon instead of command line, and not use output.

To get even deeper under the covers, Heat is architecturally composed of the sections (all installed on the Controller nodes): `heat-api`, `heat-api-cfn`, and `heat-engine`. That's all you need to know.

Crawl: Hello World

The following is a very basic template found on the OpenStack documentation page. It will define a single instance.

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: Skunkworks_Key
      image: centos.6-4.x86-64.20120402
      flavor: m1.small
```

source http://docs.openstack.org/developer/heat/template_guide/hot_guide.html

You will note that under the section called `resources`, I have called for just one type of resource: a server. I know it is a server because the `type` tells me it is an OpenStack Nova Server (*OS::Nova::Server*). I have given it a name: "*my_instance*".

I want *my_instance* to have certain `properties`:

- I want this instance to be based on a certain `image` that I have in my OpenStack glance repository (my image is called "*centos.6-4x86-64.20120402"*), and
- I want it to be a certain size or `flavor` (in this case "*m1.small"*).
- I also want to control access to this instance by injecting a `key_name` (I used "*Skunkworks_Key"*).

If you wanted to run this same template, you need only change the *resource properties* to match your local environment.

How to actually run HOT

When it comes to running a Heat template, you have a few options. One is to save it as a file (use the 'yaml' extension such as *SimpleStack.yaml*). This way you can call the Heat engine from the command line tools or even REST calls, or from the Horizon dashboard. This is the preferred method, especially for larger templates. For smaller templates, one can just paste the code directly into Horizon. If you were to take this basic Heat template and copy it into the Horizon dashboard, it would look something like this:

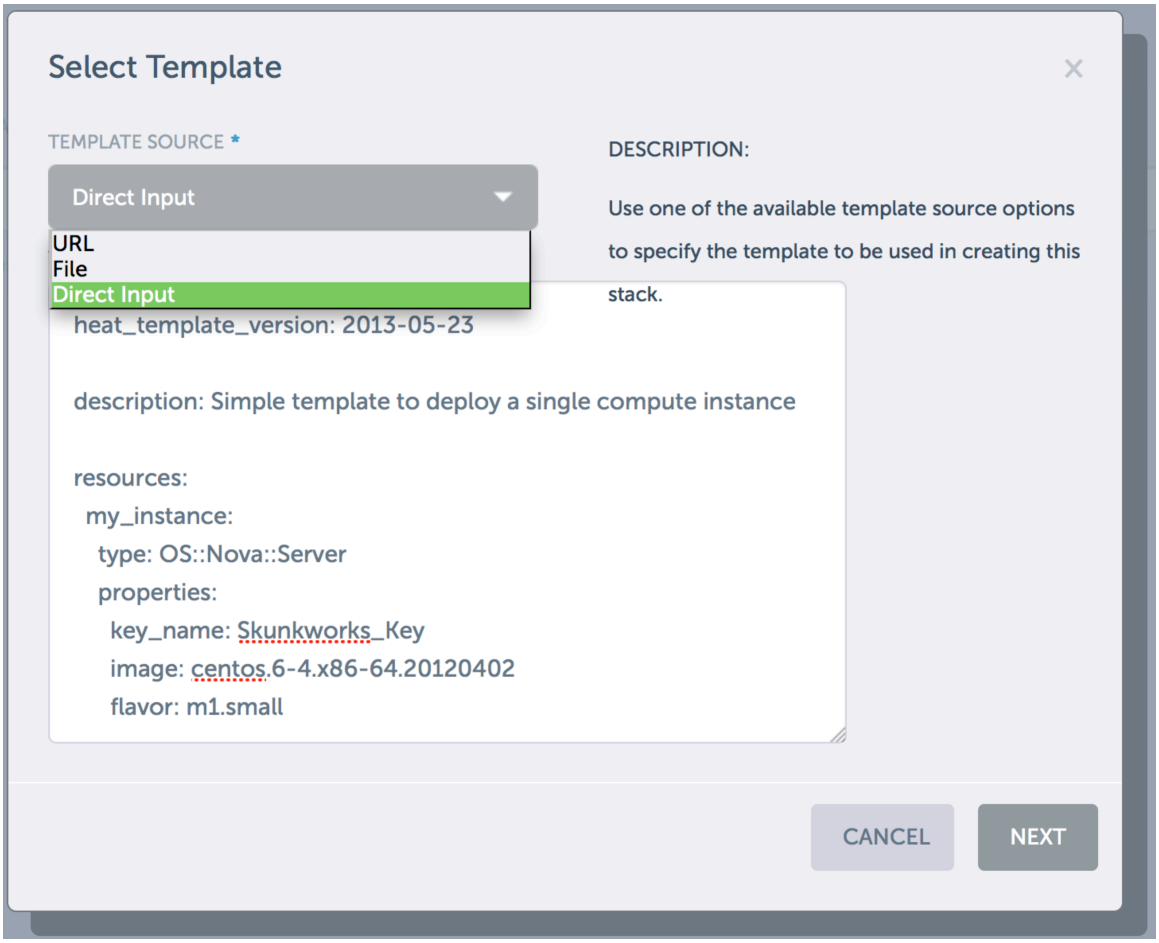

Figure 2. Running a template through the dashboard by pasting code

This is what it looks like when the script is being run:

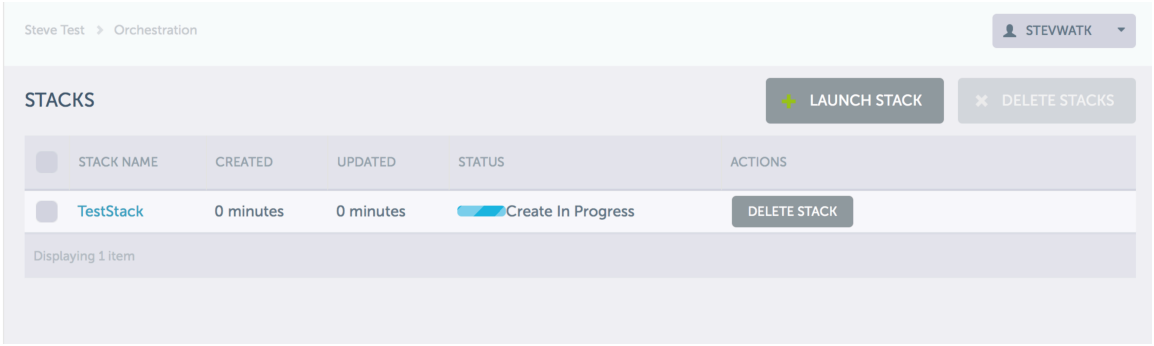

Figure 3. Running the script in Horizon

After the stack is stood up, you can see it in the Horizon dashboard view.

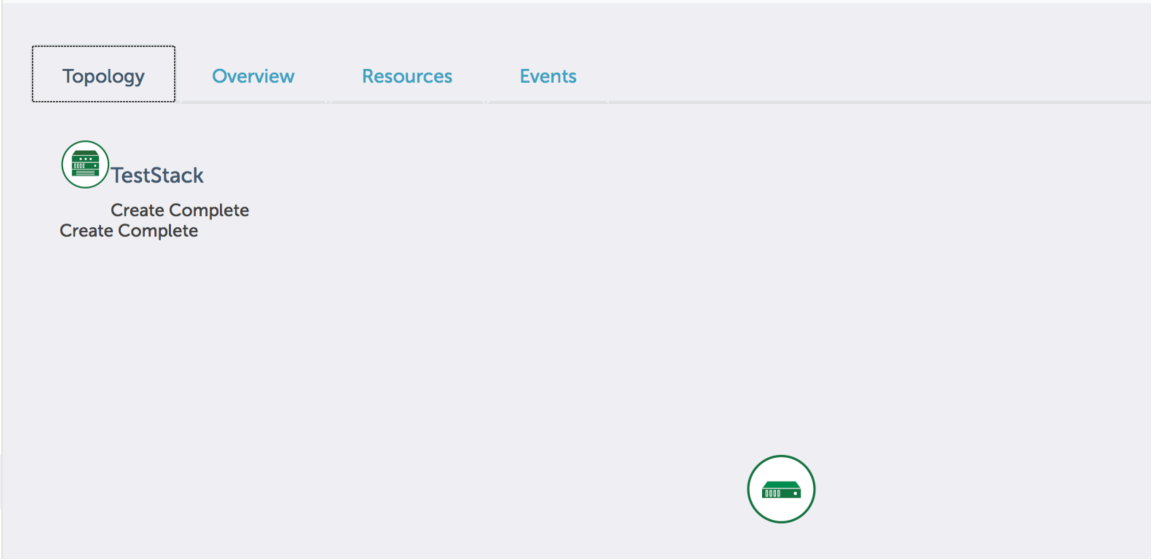

Figure 4. Topology view. Not much there in this example.

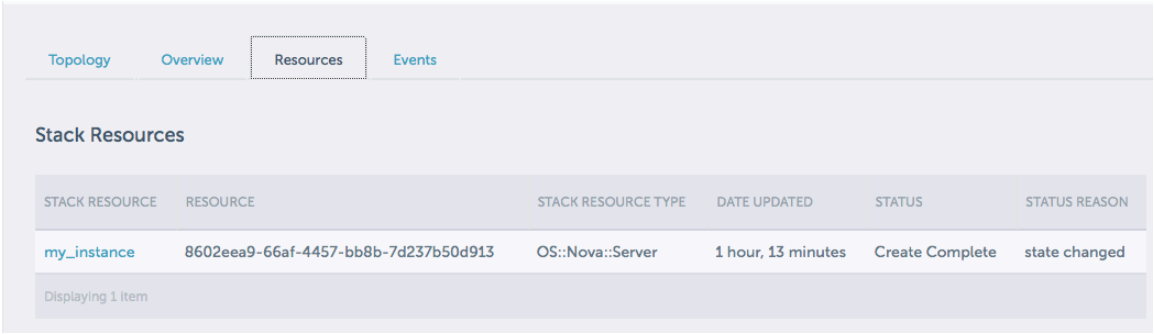

Figure 5. Resource view. Only one instance in this very simple example

Asking for input

The above example works fine, but I have hardcoded the values, which is not so useful. To make the template more dynamic so it can re-used for different images, I will add prompts to the parameters. The basic structure stays the same, but the template now expects the user to add some data.

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  image_id:
    type: string
    label: Image ID
    description: Image to be used for compute instance
  instance_type:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used
      constraints:
      - allowed_values: [ m1.tiny, m1.medium, m1.small ]
        description: Value must be one of m1.tiny, m1.small or m1.medium.

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: { get_param: image_id }
      flavor: { get_param: instance_type }
```

Notice that the parameters are defined first: It is expecting a string (text) to provide the name of the key-pair, the image desired, and the flavor. The second part of the template is almost identical to the first example, except I have replaced the hard-coded parameter values with "`{ get_param: paramName }`."

Notice that the flavor parameter is different from the others. I have used `constraints` to define a list of acceptable flavors for this template. Whoever runs this template will get a pick list of values instead of being forced to type in the values freehand. Also note that the options appear in the order described in the template, not alphabetically.

Running this template into the dashboard will give me a different experience:

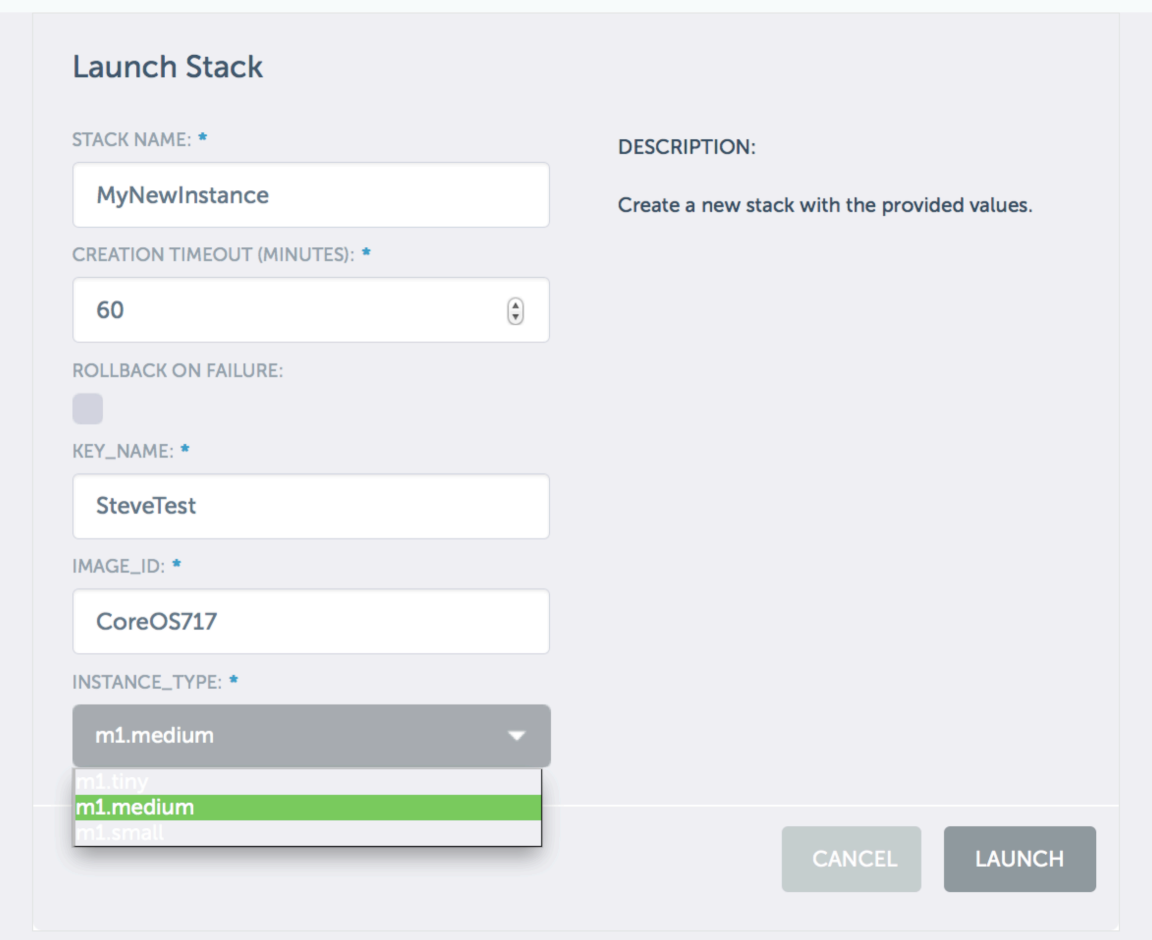

Figure 6. Parameter values are entered in Horizon at runtime

Walk: Assembling a real stack

So far, I have only used Heat to stand up a single server. In truth I could accomplish the same task quite simply through the Horizon dashboard, or via CLI or REST commands. Heat really comes into value when standing up multiple servers, adding some storage, and assigning Floating IPs. In other words, creating a stack.

I'll build on the previous template to add more servers. It's as simple as copying previous content and changing the name. In my case, I will create two instances as web and database servers.

```
heat_template_version: 2013-05-23
description: Simple template to deploy a two compute instances

parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance

resources:
  my_web_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: Cirros.0.3.1.raw
      flavor: m1.small
      networks:
        - network: demo1-1029

  my_DB_Instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: Cirros.0.3.1.raw
      flavor: m1.medium
      networks:
    -  network: demo1-1029
```

In this case, I have copied and pasted the YAML code and just changed the name of the resource. I chose clear names ("*my_web_instance*") and kept the key name as a prompt. Note that I have also added a `network` property. In my case, the image only has one virtual NIC and so only one network is required. If my image had multiple networks and I wanted to add another, or even define ports and security groups, I could do it here.

Pump up the Volume

Building on the above, let's add a Volume to the database server. A Volume is a new resource type in our Heat template (remember that the other resource type is OS::Nova::Server). The new resource `type` is *OS::Cinder::VolumeAttachment*. A quick look at the OpenStack Heat reference tells me the parameters required for a resource of this type (here is the link: http://docs.openstack.org/developer/heat/template_guide/openstack.html#OS::Cinder::VolumeAttachment). The challenge is that I will need the UUID of the server to attach, as well as the ID of the Volume.

> It's not intuitive, but the function to attach one resource to another resource is considered a '`resource`' in Heat. So it follows the same document alignment as creating a new Instance in the Heat template. Did I confuse you yet?

To get the Volume ID, I can always find it either through command line tools or even the Horizon dashboard. I can then make the Volume ID a prompt just like the key name.

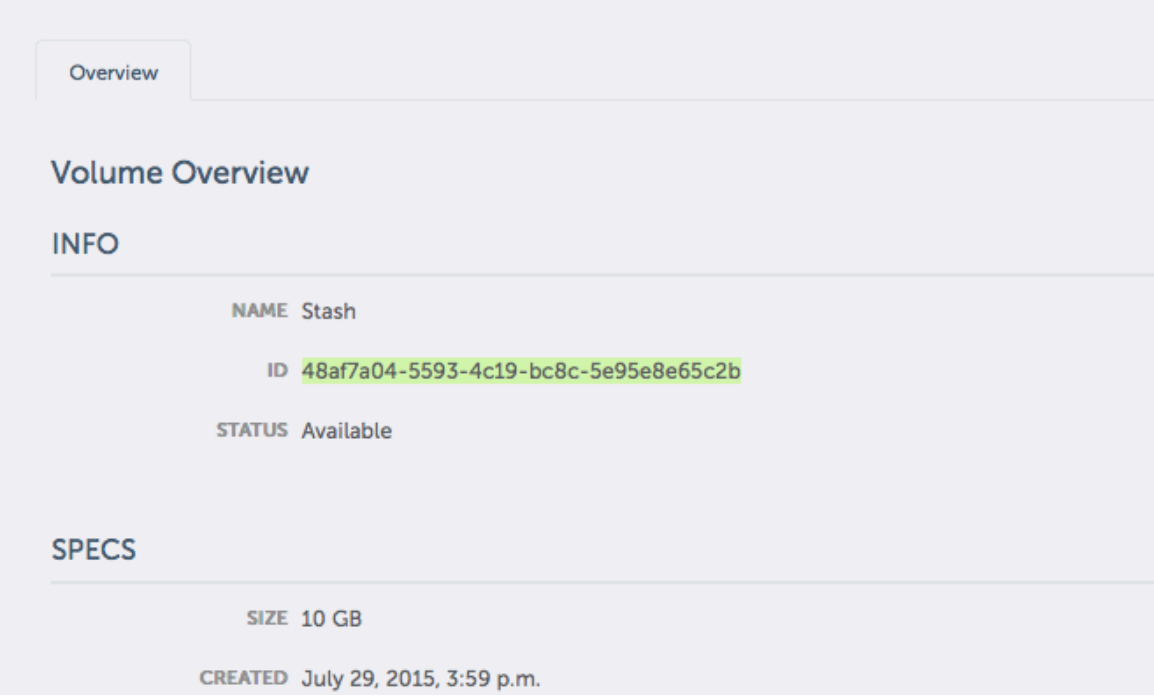

Figure 7. Get the ID of the Volume called Stash in the dashboard

The next challenge is getting the UUID of the server we are creating. Since it is not yet created, I can't look it up like the Volume ID. Fortunately, Heat includes some nifty functions to reference other items in the template. Similar to the "get_parame" function we used to prompt the user for info, Heat provides a "get_resource" function. This function will return the unique ID of a resource. Since I need the UUID of "my_DB_Instance," I will add the following code:

```
{ get_resource: my_DB_Instance }
```

Armed with that ID, I can run the following HOT template:

```
heat_template_version: 2013-05-23


description: Simple template to deploy a two compute instances, create volume
and attach it


parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  DB_Volume:
    type: string
    label: DB_Volume
    description: This is the unique ID of the Volume.  I got it from the Horizon
dashboard properties tab


resources:
  my_web_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: Cirros.0.3.1.raw
      flavor: m1.small
      networks:
        - network: demo1-1029
  my_DB_Instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: Cirros.0.3.1.raw
      flavor: m1.medium
      networks:
        - network: demo1-1029
  DB_Volume_att:
     type:  OS::Cinder::VolumeAttachment
     properties:
        instance_uuid: { get_resource: my_DB_Instance }
        volume_id: { get_param: DB_Volume }
```

When I run the above template, I get two new instances, and attach one of them to my existing volume. In Horizon it looks like this:

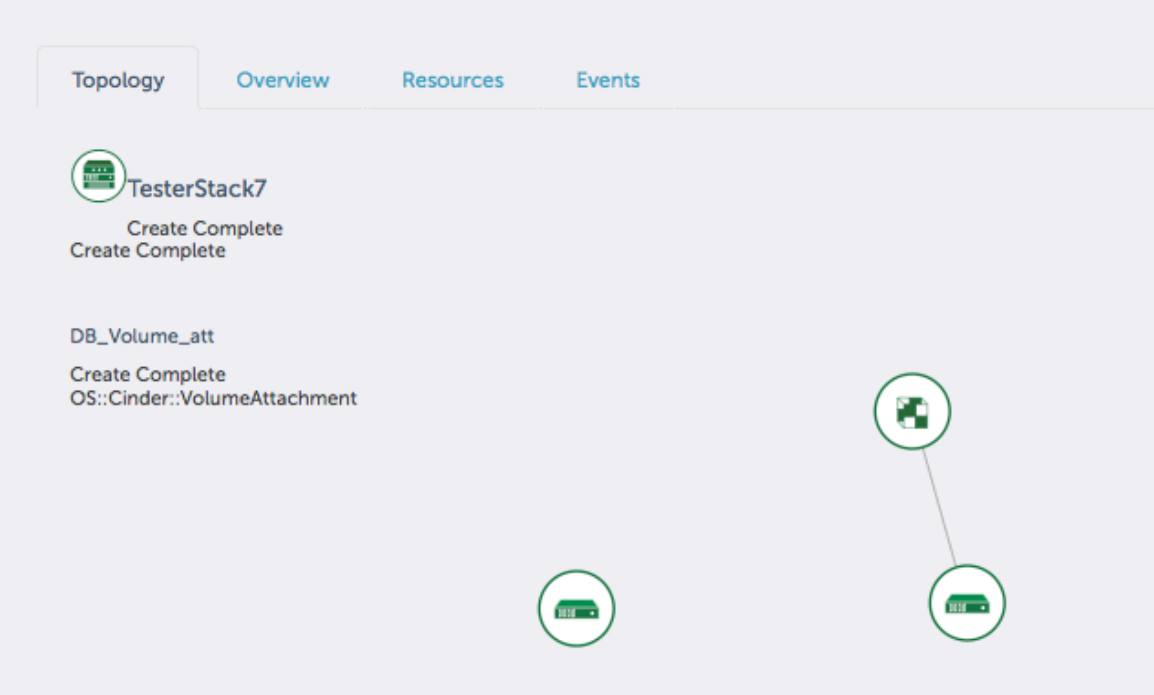

Figure 8. Cinder volume attached to my_DB_Instance server

It's not intuitive, but the function to attach a resource is considered a '`resource`' in Heat. So it follows the same document alignment as creating a new Instance in the Heat template. It appears as a resource in Horizon's stack view. Note that "DB_Volume_att" has no hyperlink under the *Resources* column.

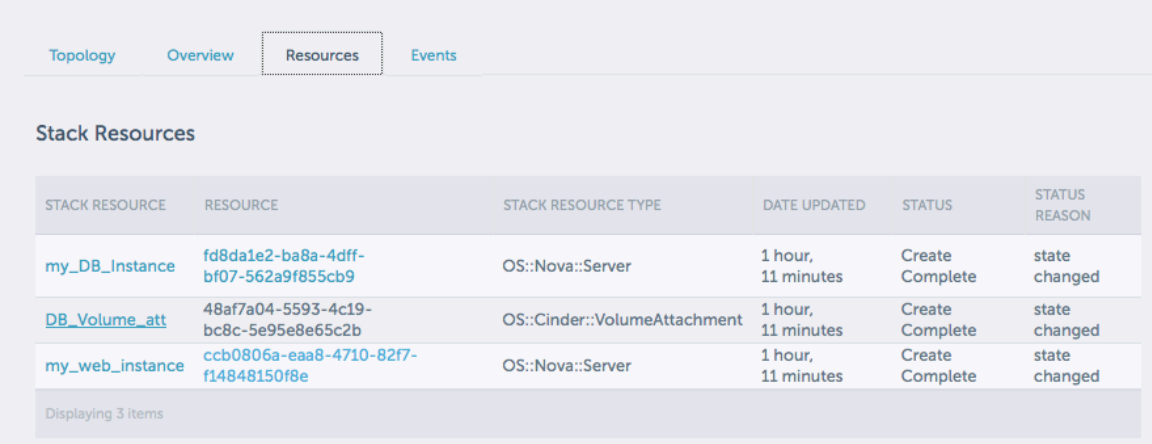

Figure 9. Horizon view of attachment as a resource

Run: Creating new resources on the fly

Now lets take it up a notch. Instead of attaching existing volumes, I want to create a new one. Another quick check on the documentation (http://docs.openstack.org/developer/heat/template_guide/openstack.html#OS::Cinder::VolumeType) tells me creating a new volume is easy. The only property required is the size of the volume (representing gigabytes). And I already know that the "get_resource" function will give me the new Volume's ID. So I will add the following code to my template:

```
DB_Volume:
     type:  OS::Cinder::Volume
     properties:
        size: 20

  DB_Volume_att:
     type:  OS::Cinder::VolumeAttachment
     properties:
    instance_uuid: { get_resource: my_DB_Instance }
        volume_id: { get_resource: DB_Volume }
```

I have created a new Volume (just like we create new Instances) and passed the new Volume's ID to the "VolumeAttachment" method. Note that these are actually two additional resource types: one to create the Volume, the other to attach it to an instance.

I can do the same thing for attaching a floating IP to my web server. The format is exactly the same, but the resource types are different.

```
web_floating_IP:
     type: OS::Nova::FloatingIP
     properties:
        pool: demo1

  web_floating_IP_att:
     type: OS::Nova::FloatingIPAssociation
     properties:
        floating_ip:  { get_resource: web_floating_IP }
        server_id: { get_resource: my_web_instance }
```

Now I can dynamically create servers, new floating IPs, and Volumes, and put them together to form a simple stack. Here is the final HOT template:

```
heat_template_version: 2013-05-23
description: Simple template to deploy a two compute instances, create volume and
attach it.  Also creates new floating IP and attaches to web server


parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance


resources:
  my_web_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: Cirros.0.3.1.raw
      flavor: m1.small
      networks:
        - network: demo1-1029

  web_floating_IP:
     type: OS::Nova::FloatingIP
     properties:
        pool: demo1

  web_floating_IP_att:
     type: OS::Nova::FloatingIPAssociation
     properties:
        floating_ip:  { get_resource: web_floating_IP }
        server_id: { get_resource: my_web_instance }

  my_DB_Instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: Cirros.0.3.1.raw
```

```
    flavor: m1.medium
    networks:
      - network: demo1-1029

DB_Volume:
   type:  OS::Cinder::Volume
   properties:
      size: 20

DB_Volume_att:
   type:  OS::Cinder::VolumeAttachment
   properties:
      instance_uuid: { get_resource: my_DB_Instance }
      volume_id: { get_resource: DB_Volume }
```

When I run this HOT template, I get the following view in Horizon:

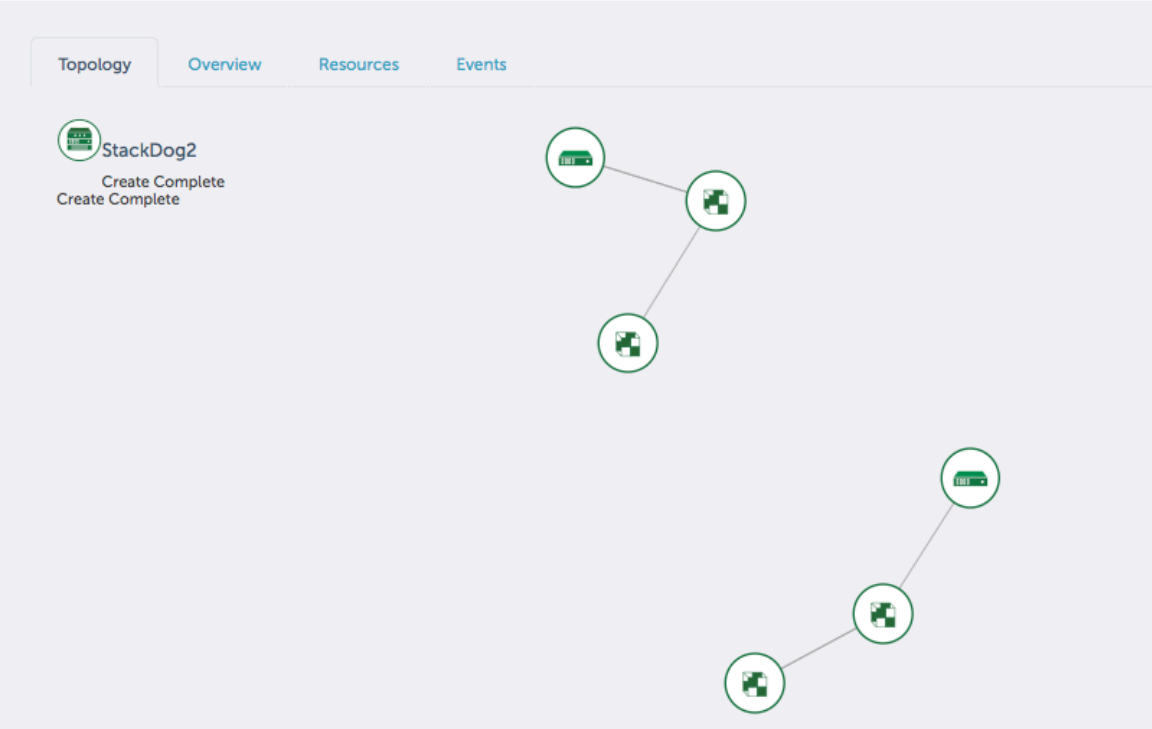

Figure 10. The graphical stack view

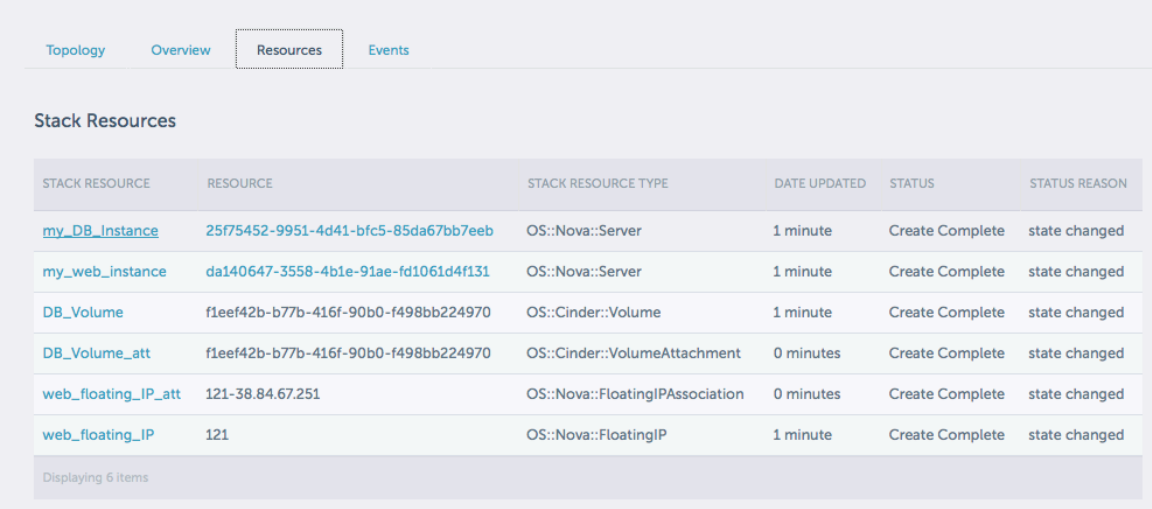

Figure 11. The full stack resource view in Horizon

Note that this is a fairly limited stack. I have not added the security group rules, or called external scripts (`user_data`) to install and configure software. So there is lots of room to improve. The goal of this paper was to provide an introduction to Heat. Hopefully, with this basic knowledge a new user can continue to explore and experiment to build out their Heat skills.

Where to go for more help

There are a lot of great resources out there. The OpenStack Foundation provides some very good documentation. In particular, I recommend the Foundation template and the resource guide.

- OpenStack Foundation: http://docs.openstack.org/developer/heat/template_guide/index.html
- Guide to resource types. This is a great place to find the HOT code needed to perform actions like creating and attaching resources. http://docs.openstack.org/developer/heat/template_guide/openstack.html#

In addition, there are lots of tutorials and blogs on the Internet and in user groups online.

Good luck, and happy orchestrating.

For More Information

Visit our website to read more Cisco OpenStack Private Cloud features and benefits.
To access technical tutorials about this product, visit our Community page

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **www.cisco.com/go/offices.**

Printed in USA CXX-XXXXXX-XX 10/11