OpenStack本身提供了命令行和Horizon供用户管理资源，用户也可以编写程序通过REST API来管理云资源。这些方式适合简单少量的资源管理和单一任务。对于大量资源的管理和负杂的云部署任务，需要使用编排（Orchestration）服务来提高效率。OpenStack的编排服务成为Heat。Heat是一个通过OpenStack原生REST API基于模板来编排复合云应用的服务。Heat提供一个云业务流程平台，可以让用户使用模板实现资源的自动化部署。

14.1 Heat编排服务

所谓编排就是按照一定的目的依次排列。在OpenStack环境中，可以使用编排服务来集中管理整个云架构、服务和应用的生命周期。编排可以通过预先设定来协调配置同一个节点或不同节点的部署资源和部署顺序。

用户将各种资源的需求写入模板文件中，Heat基于模板文件自动调用相关服务的接口来配置资源，从而实现自动化云部署。

在编排服务中，资源特指编排期间创建或修改的对象，可以是网络、路由器、子网、实例、卷、浮动IP、安全组等。

模板以文本文件的形式描述了云应用的基础设施，主要是需要被创建的资源的细节。

Heat 从四个方面来支持编排：

1. 首先是 OpenStack 自己提供的基础架构资源，包括计算，网络和存储等资源。通过编排这些资源，用户就可以得到最基本的 VM。
2. 在编排 VM 的过程中，用户可以提供一些简单的脚本，以便对 VM 做一些简单的配置。
3. 然后用户可以通过 Heat 提供的 Software Configuration 和 Software Deployment 等对 VM 进行复杂的配置，比如安装软件、配置软件。
4. 接着如果用户有一些高级的功能需求，比如需要一组能够根据负荷自动伸缩的 VM 组，或者需要一组负载均衡的 VM，Heat 提供了 AutoScaling 和 Load Balance 等进行支持。 如果要用户自己单独编程来完成以上这些功能，所花费的时间和编写的代码都是不菲的。通过 Heat，只需要一段长度的 Template，就可以实现这些复杂的应用。Heat 对诸如 AutoScaling 和 Load Blance 等复杂应用的支持已经非常成熟，有各种各样的模板供参考。

14.1.1 Heat与OpenStack其它组件之间的关系

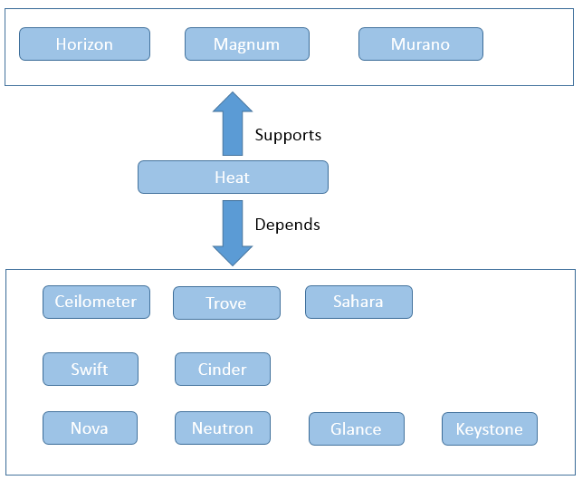


图14-1 Heat与OpenStack其它组件之间的关系

14.1.2 Heat架构

Heat的主要组件：

* heat:命令行工具，用于与heat-api通信以执行AWS CloudFormation API.终端人员也可以直接使用heat的REST API。
* heat-api:提供OpenStack本身支持的REST API，通过RPC将API请求发送给heat-engine进行处理。
* heat-api-cfn:提供兼容AWS的API，通过RPC将API请求发送给heat-engine进行处理。
* heat-engine:Heat的核心，主要实现任务调度、资源生命周期管理等作用，它自身并不会创建资源，只负责资源编排，资源的创建都会交由相应的组件去处理。

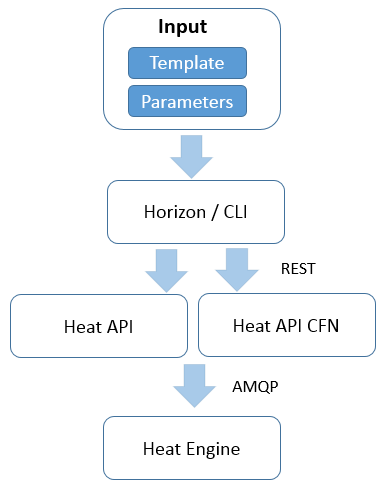


图14-2 heat架构

Heat的工作机制如下：

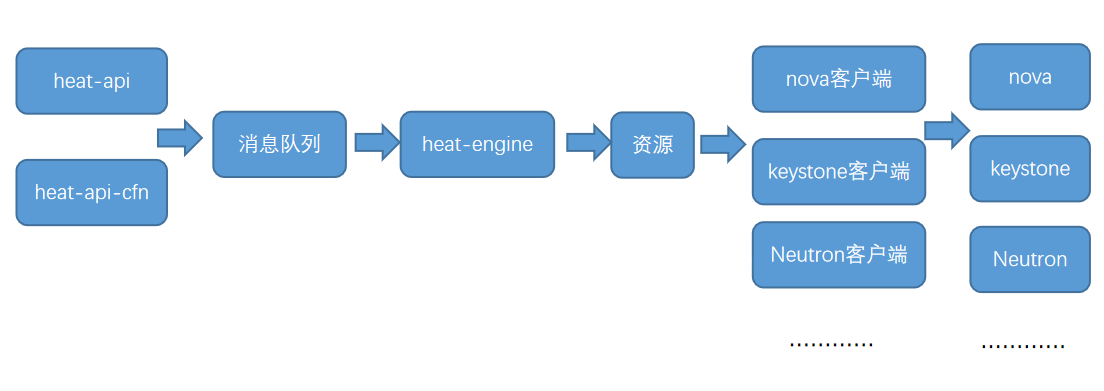


图14-3 Heat工作机制

用户在Dashboard图形界面中或者命令行中提交包含模板和参数输入的请求，Dashboard图形界面中或者命令行工具会将请求转化为REST格式的API调用，然后调用heat-api或者heat-api-cfn。heat-api或者heat-api-cfn会验证模板的有效性，然后通过AMQP异步传递给heat-engine来处理请求。

核心组件heat-engine提供Heat最主要的协作功能。当heat-engine收到请求后，会将请求解析为各种类型的资源，每种资源都对应OpenStack其他服务的客户端，然后发送REST请求给其他服务。

组件heat-engine的作用可以分为三个层面。第一个层面负责处理Heat层面的请求，根据模板和输入参数创建栈（由各种资源组成）。第二个层面解析栈中各种资源的依赖关系，以及栈和嵌套栈的关系，第三个层面就是根据解析出来的次序、依赖关系和嵌套关系，依次调用各种服务客户端来创建各种资源。

14.2 Heat编排模板

Heat采用了模板方式来设计和定义编排，用户只需要使用文本编辑器编写包含若干节、键值对（key-value pair）代码的模板文件，来实现资源的编排。Heat目前支持两种格式：一种是基于YAML的HOT（Heat Orchestration Template）模板;另一种基于JSON格式的CFN（CloudFormation-compatible）模板，CFN主要为了兼容AWS。HOT是Heat自由的模板，资源类型更加丰富，更能体现Heat的特点，比CFN更好。主要讲解HOT模板。

14.2.1 HOT结构

OpenStack官方模板向导：

<https://docs.openstack.org/heat/latest/template_guide/index.html>

HOT模板由以下几部分构成：

* 模板版本信息。必填部分，用于指定相应的版本信息，不同版本都向下兼容。
* 模板参数列表。可选部分，用于定义模板的输入参数。
* 模板资源列表。必填部分，设计用户需要创建哪些资源，也可以指定资源间的关联及绑定关系。
* 模板输出信息。可选部分，用于指定Stack暴露的原信息，此信息可以为其他stack使用。

HOT的结构如下

（参考：

[https://docs.openstack.org/heat/latest/template\_guide/hot\_spec.html#hot-spec-template-version](https://docs.openstack.org/heat/latest/template_guide/hot_spec.html" \l "hot-spec-template-version)）：

heat\_template\_version: 2016-10-14

#heat\_template\_version: 必填字段，指定对应的模板版本，Heat会根据版本进行检验和 #支持的功能。

#命令heat template-version-list可查看所有模板版本

description:

 # description: 选填字段，用于模板的描述

parameter\_groups:

 #parameter\_groups: 选填字段，允许指定如何对输入参数进行分组，以及提供参数的顺#序，这部分在生产中比较少用。

parameters:

  # 选填字段，指输入参数列表。允许指定实例化模板时必须提供的输入参数。

#当参数比较多时，可以通过Environment文件批量传入参数的值。当模板设计完成之

#后，如果作者想对HOT模板中的参数值进行修改，只需要修改Environment中的参数 #即可。可通过heat resource-type-show {*resource\_name*}查看*resource\_name支持的参数。*

resources:

  # 必填字段，指生成的 Stack 所包含的各种资源，比如服务器、卷、对象、存储和网

#络资源。可以定义资源间的依赖关系。

outputs:

  # 选填字段，指生成的 Stack 暴露出来的信息，可以用来给用户使用，也可以用来作

#为输入提供给其它的 Stack。

conditions:

  # 选填字段，定义了一个或多个条件，这些条件根据用户创建或更新stack时提供的输

#入参数值进行评估。条件可以与资源、资源属性和输出相关联。（在生产中很少用到）

14.2.2 参数列表

parameters: #自定义参数

Image: #参数①

type: string #参数类型

label: image #参数标签

description: Image use to boot a server #描述

default: cirros #未输入参数值时选择默认值

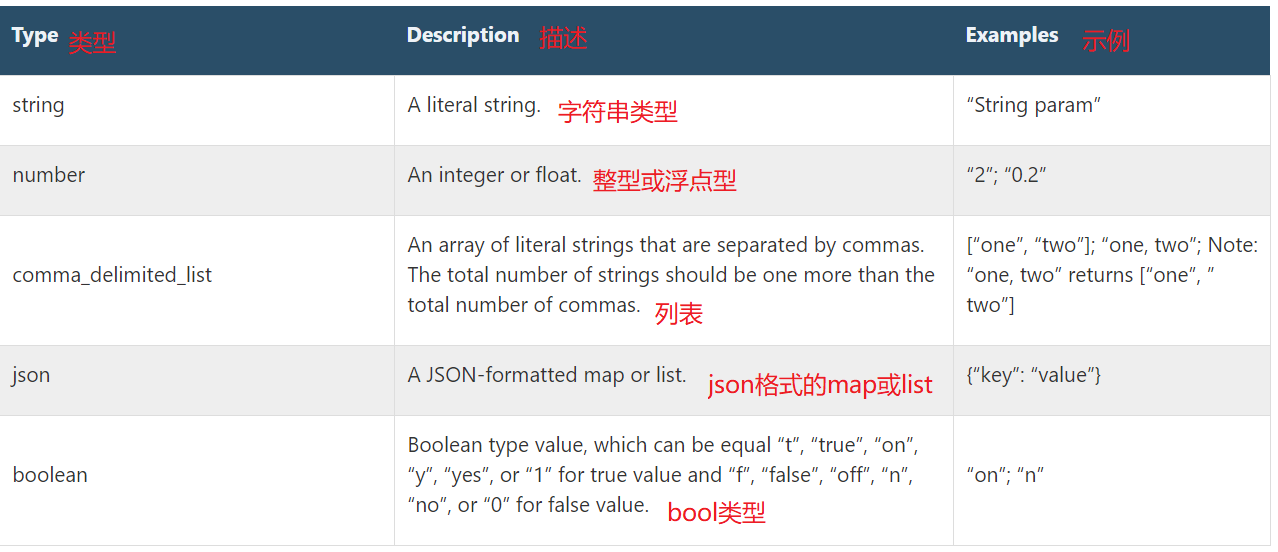
Net: #参数②

type: string

label: net

description: Network for the server

对于HOT模板中定义的参数，需要指定参数的类型，如果用户设置的是一个非法类型的参数，那么Heat在进行模板校验时就会报错。HOT模板支持的参数类型如下：



定义参数时，只有参数名字和参数类型是必选项，label、description是可选性。

14.2.3 资源列表

resources: #描述的资源（如网络、实例、云硬盘等）

server1: #资源名称或资源ID

type: OS::Nova::Server #资源类型（为云主机）（heat resource-type-list可查看所有资源类型）

properties: #云主机属性

name: "Test Heat server" #云主机名称

image: { get\_param: Image } #镜像

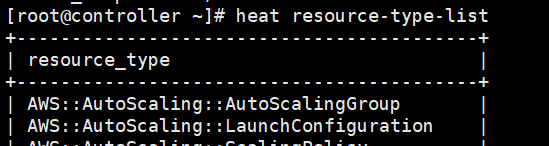
flavor: "2" #云主机类型（cpu、内存、硬盘等）

networks: #网络（可以有不止一个）

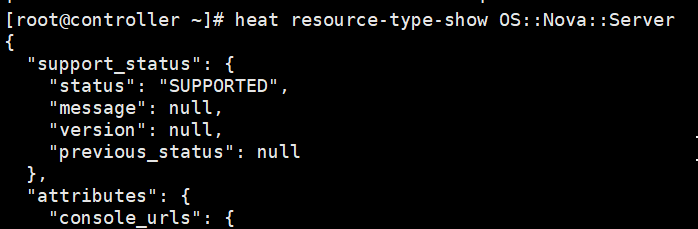
- network: { get\_param: Net }

metadata: #资源特定的元数据

查看heat支持的资源：



列出资源的详情：



14.2.4 输出列表

outputs: #返回值

server1\_private\_ip: #返回值名称

description: IP address of the server in the private network #描述

value: { get\_attr: [ server1, first\_address ] }

两个常用的内置函数介绍：

get\_param函数：引用模板中指定的参数。如get\_param: Image，引用parameters中Image的值。  
 get\_attr函数：引用资源属性。如get\_attr: [ server1, first\_address ]，引用server1资源中first\_address的值。first\_address必须是模板resouce段中指定的资源。

14.2.5 HOT模板常见错误

     在写HOT模板时格式由于不对齐或少空格经常出错。

    如果报错信息为：

      expected <block end>, but found BlockMappingStar

　　  解决的方法是：

       每个配置行前需要有空格，每个“：”两边需要有空格，数组中间加空格，还有注释掉的参数不能在后边加空格不然报错。

       如果报错信息为：

mapping values are not allowed here

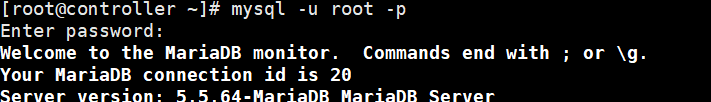
     原因是冒号 “:”后面没有空格。原因分析：yaml文件中，键值对是以":"作为分隔符，而值经常会包含冒号，比如服务器地址。在yaml解析器解析过程中，如果不在键值对中加特殊符号，还真是难以根据键解析出值来。

14.3 Heat服务安装和配置

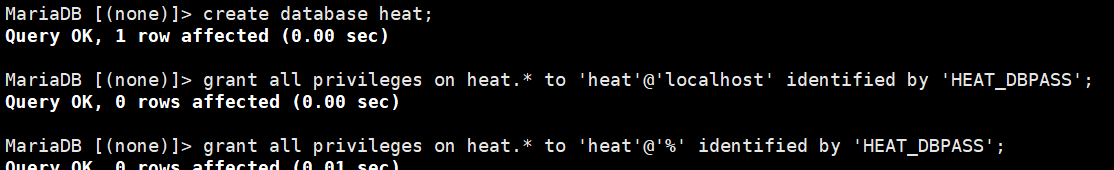
14.3.1 基础服务

1. 创库授权

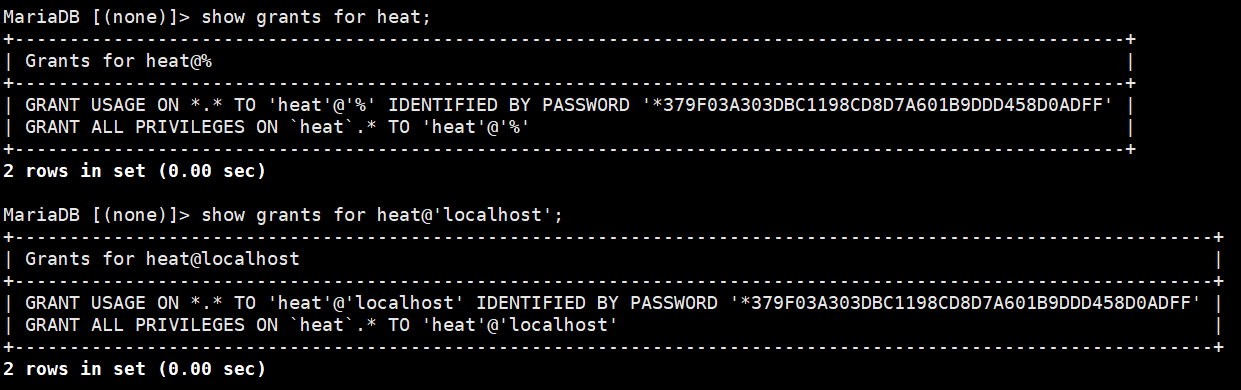
连接：



创库授权：

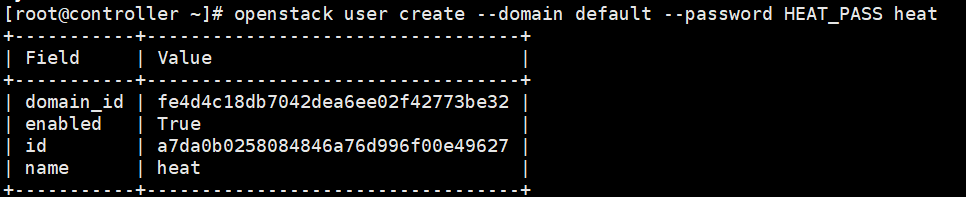


验证：



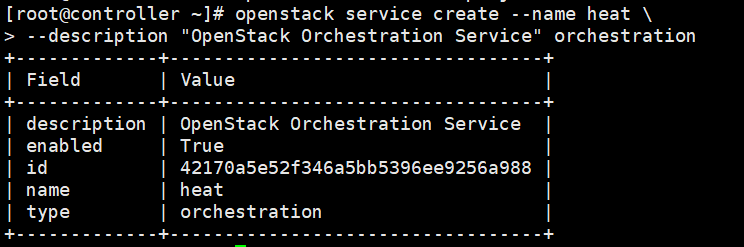
退出数据库。

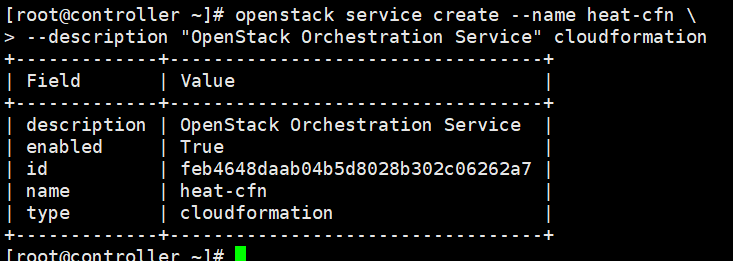
1. 在keystone上创建heat用户，并授权为service项目中的管理员admin角色；



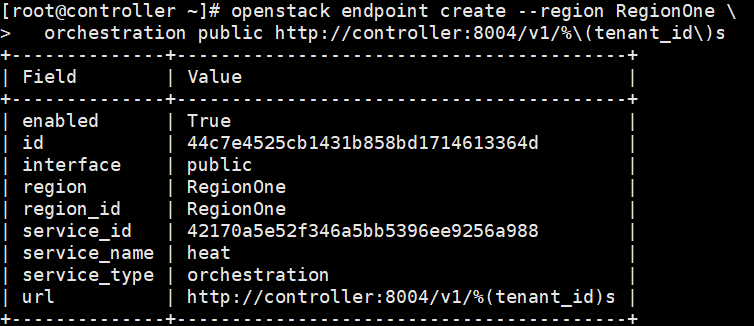


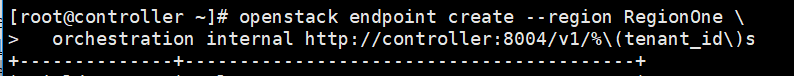
1. 创建``heat`` 和 heat-cfn 服务实体





1. 创建 Orchestration 服务的 API 端点







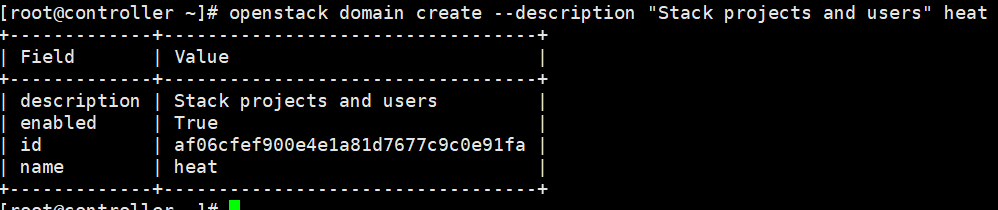


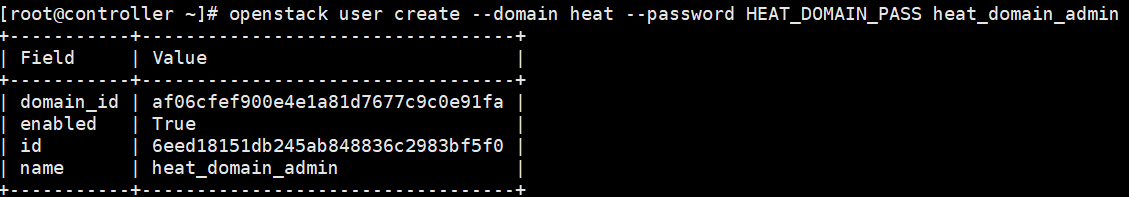




1. 在认证服务中Orchestration需要更多信息以管理栈

A.为栈创建 heat 包含项目和用户的域：

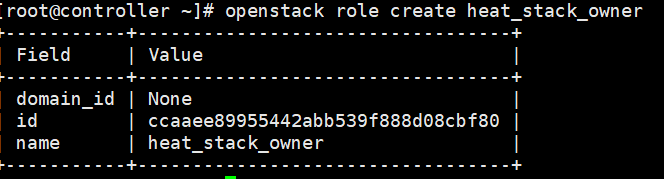
 B.在 heat 域中创建管理项目和用户的``heat\_domain\_admin``用户：



C.添加``admin``角色到 heat 域 中的``heat\_domain\_admin``用户，启用``heat\_domain\_admin``用户管理栈的管理权限：



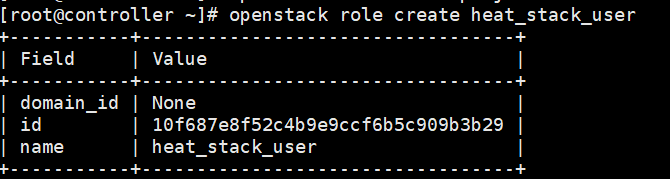
D.创建 heat\_stack\_owner 角色：



E.添加``heat\_stack\_owner`` 角色到``demo`` 项目和用户，启用``demo`` 用户管理栈。



F.创建 heat\_stack\_user 角色：



Orchestration 自动地分配 ``heat\_stack\_user``角色给在 stack 部署过程中创建的用户。默认情况下，这个角色会限制 API 的操作。

为了避免冲突，请不要为用户添加 ``heat\_stack\_owner``角色。

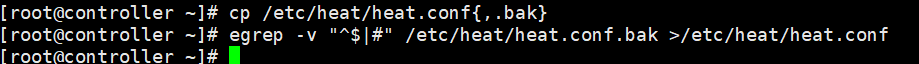
14.3.2安装并配置heat组件

（1）安装软件包



1. 编辑文件 /etc/heat/heat.conf

A.修改配置文件之前先备份：



B.在 [database] 部分，配置数据库访问：

[database]

connection = mysql+pymysql://heat:HEAT\_DBPASS@*controller*/heat

将 HEAT\_DBPASS 替换为你为 Orchestration 数据库选择的密码。

C.在 “[DEFAULT]” 和 “[oslo\_messaging\_rabbit]”部分，配置 “RabbitMQ” 消息队列访问：

[DEFAULT]

rpc\_backend = rabbit

[oslo\_messaging\_rabbit]

rabbit\_host = *controller*

rabbit\_userid = openstack

rabbit\_password = RABBIT\_PASS

用你在 “RabbitMQ” 中为 “openstack” 选择的密码替换 “RABBIT\_PASS”。

D.在``[keystone\_authtoken]``， [trustee]，[clients\_keystone]``和 ``[ec2authtoken] 部分， 配置认证服务访问：

[keystone\_authtoken]

auth\_uri = http://*controller*:5000

auth\_url = http://*controller*:35357

memcached\_servers = *controller*:11211

auth\_type = password

project\_domain\_name = default

user\_domain\_name = default

project\_name = service

username = heat

password = HEAT\_PASS

[trustee]

auth\_plugin = password

auth\_url = http://*controller*:35357

username = heat

password = HEAT\_PASS

user\_domain\_name = default

[clients\_keystone]

auth\_uri = http://*controller*:35357

[ec2authtoken]

auth\_uri = http://*controller:*5000/v2.0

将``HEAT\_PASS`` 替换为你在认证服务中为 heat 用户选择的密码。

E.在``[DEFAULT]`` 部分，配置元数据和 等待条件URLs：

[DEFAULT]

heat\_metadata\_server\_url = http://*controller*:8000

heat\_waitcondition\_server\_url = http://*controller*:8000/v1/waitcondition

在 [DEFAULT] 部分，配置栈域与管理凭据：

[DEFAULT]

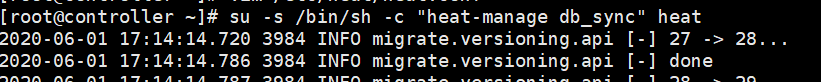
stack\_domain\_admin = heat\_domain\_admin

stack\_domain\_admin\_password = *HEAT\_DOMAIN\_PASS*

stack\_user\_domain\_name = heat

将 HEAT\_DOMAIN\_PASS 替换为你在认证服务中为``heat\_domain\_admin`` 用户选择的密码。

14.3.3 同步数据库



同步成功后，显示heat库有16张表。

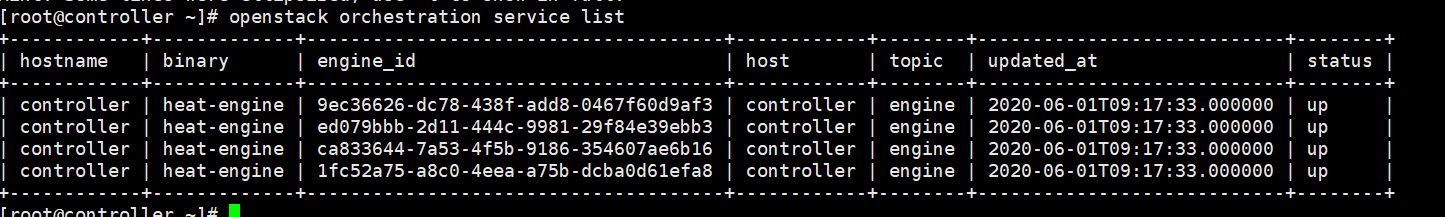


14.3.4 启动服务





14.3.5 验证heat服务



必须包含4个host engine进程服务才算安装成功。

14.4 通过HOT模板创建虚拟机

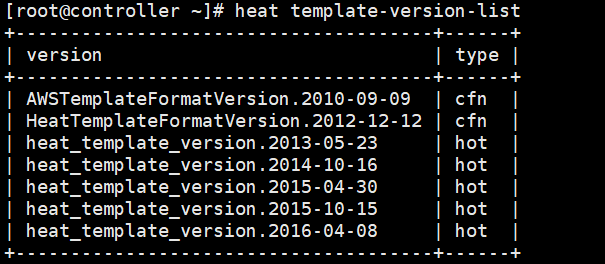
14.4.1 准备编排服务所需的资源

对于创建实例的栈，主要需要准备镜像、实例类型、网络和密钥对等基本资源。

例如通过openstack image list查看镜像；openstack flavor list查看已有的实例类型等。

14.4.2 创建模板

确认heat支持的模板版本：



注意：按照如下编写，注意每个":"后都要有空格，即时没有字符也要有空格。

heat\_template\_version: 2016-04-08

description: Launch a basic instance with cirros image using the "m1.nano" flavor,private network.

parameters:

image:

type: string

default: cirros

flavor:

type: string

default: m1.nano

netID:

type: string

description: Network ID to use for the instance.

resources:

server:

type: OS::Nova::Server

properties:

image: { get\_param: image }

flavor: { get\_param: flavor }

key\_name: mykey

networks:

- network: { get\_param: NetID }

outputs:

instance\_name:

description: Name of the instance.

value: { get\_attr: [ server,name ] }

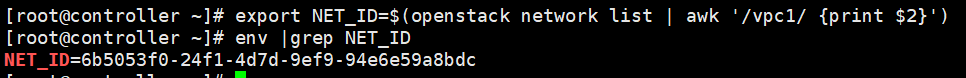
instance\_ip:

description: IP address of the instance.

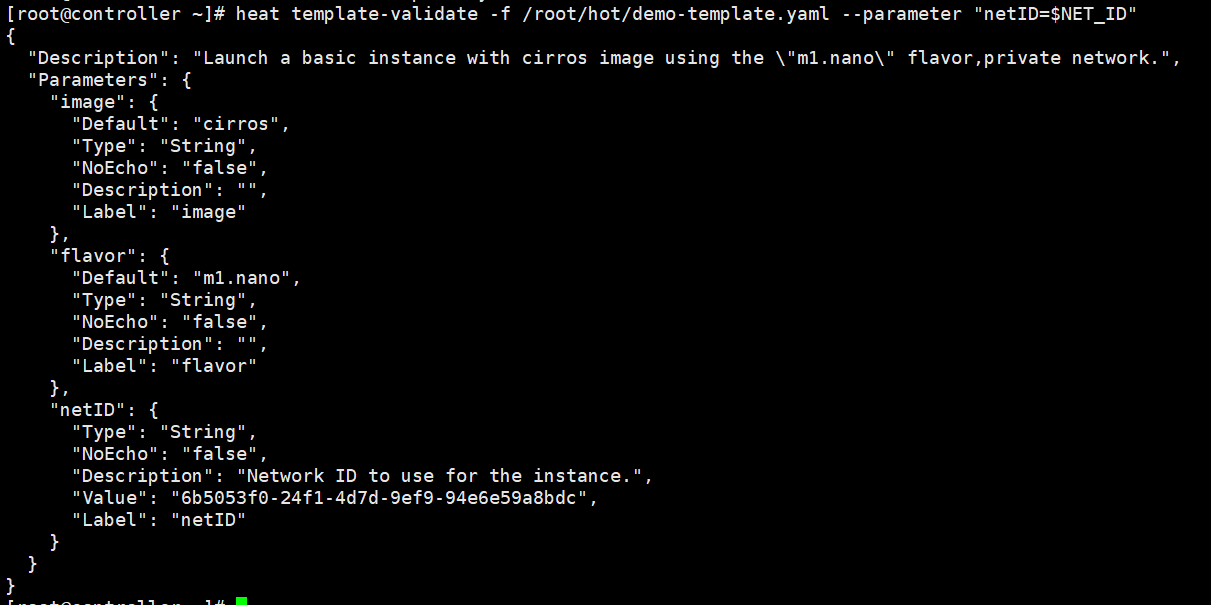
value: {get\_attr: [ server, first\_address ] }

14.4.3 创建栈

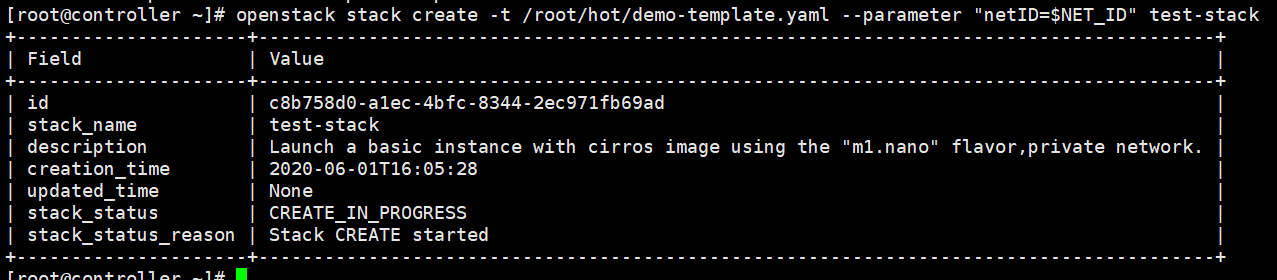
1. 加载admin凭据的环境变量
2. 确定可用的网络。openstack network list查看网络列表。
3. 设置环境变量NET\_ID，本例使用私有网络vpc1。通过以下命令获得vpc1的ID：



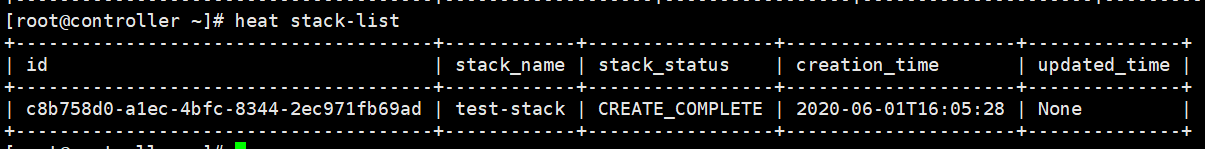
1. 验证模板的合法性



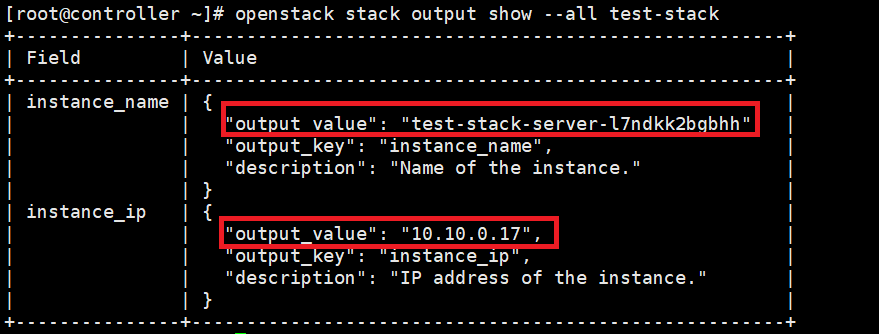
1. 使用该模板创建一个名为test-stack的栈，用于创建使用vpc1网络的cirros实例。



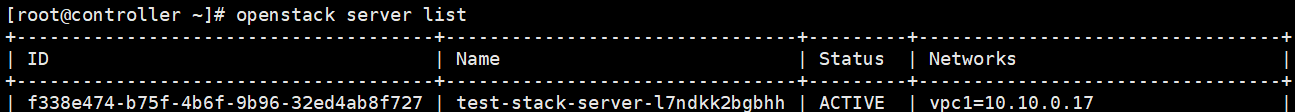
当stack创建完成后，再通过heat stack-list查看。此时stack的状态已变为CREATE\_COMPLETE.



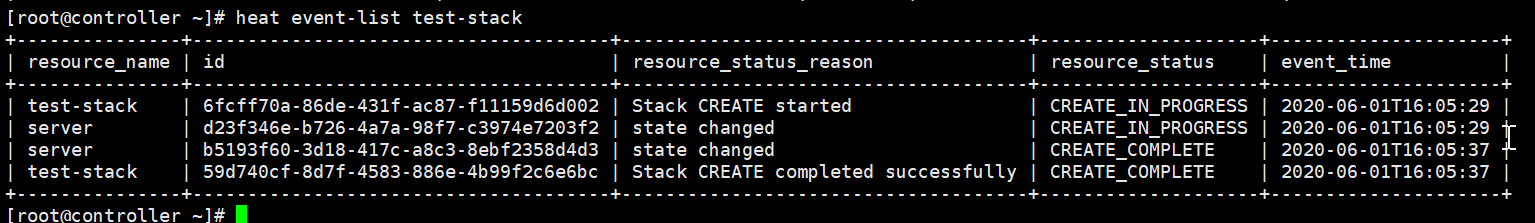
1. 显示该实例的名称和IP地址



或者



1. 当需要查看Stack创建到哪一步时，可以使用event-list查看：



如上说明test-stack已经创建完成。若Stack创建失败，可以查看/var/log/heat/heat-api.log或者/var/log/heat/heat-engine.log.

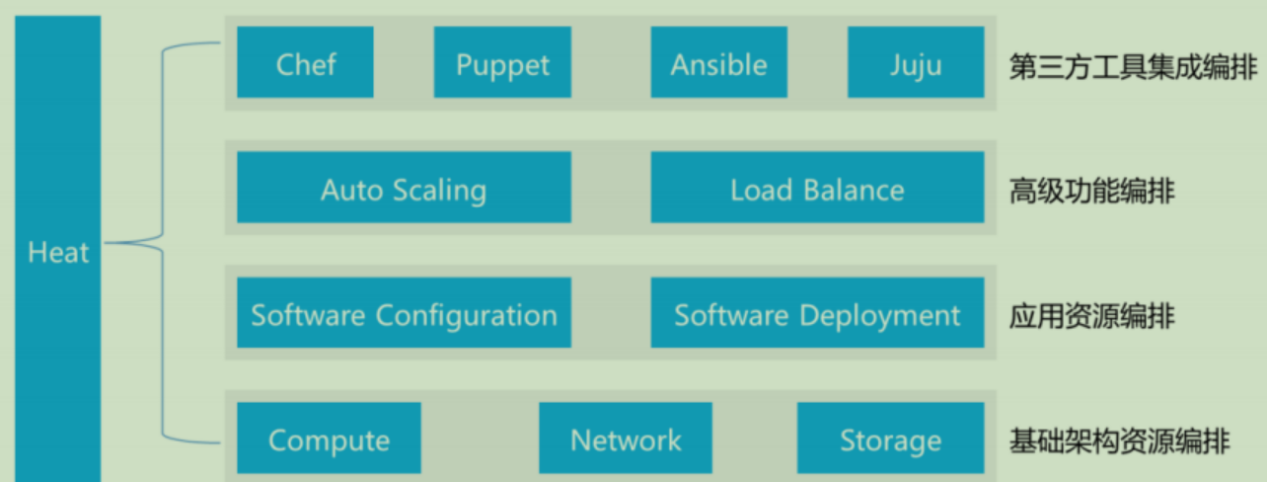


与heat相关的操作参考：

<https://specs.openstack.org/openstack/heat-specs/specs/mitaka/python-openstackclient.html>

14.5 创建模板定制编排任务

Heat管理和使用最重要的工作就是创建模板，模板决定了具体的编排功能和目标。Heat项目提供了大量的模板例子，可供参考。使用编排的场景如下：



14.5.1 对基础设施的编排

对基础设施的编排是最基本的编排功能。通过OpenStack自己提供的基础设施资源（如计算、网络和存储等）创建最基本的虚拟机实例。Heat能够启动应用、创建虚拟机并自动处理整个流程。

1. 管理虚拟机实例，

对于实例创建，需要通过模板预定义虚拟机创建时所需要的资源，为此Heat提供了资源类型OS::Nova:Server。OS::Nova:Server的主要属性有key、image、flavor等，这些属性可以直接指定，也可以由客户在创建栈时以输入参数的形式提供，还可以由上下文其他属性获得。在部署一个由多台虚拟机组成的业务集群时，还可以定义虚拟机创建时的依赖关系。

创建安全组应使用资源类型OS::Neutron::SecurityGroup。将安全组关联到一个实例应定义资源类型OS::Neutron::Port的security\_groups属性，将安全组关联到一个端口，再将该端口关联到实例。

示例：下面的列子创建一个允许端口80的传入连接的安全组，并将其连接到一个实例端口。

resources:

web\_secgroup:

type: OS::Neutron::SecurityGroup

properties:

rules:

- protocol: tcp

remote\_ip\_prefix: 0.0.0.0/0

port\_range\_min: 80

port\_range\_max: 80

instance\_port:

type: OS::Neutron::Port

properties:

network: *private*

security\_groups:

- default

- { get\_resource: web\_secgroup }

fixed\_ips:

- subnet\_id: *private-subnet*

instance:

type: OS::Nova:Server

properties:

flavor: *m1.nano*

image: *cirros*

networks:

- port: { get\_resource: instance\_port }

对于浮动IP及其关联的实例，可以使用OS::Nova:FloatingIP资源创建一个浮动IP，再使用OS::Nova:FloatingIPAssociation资源将其关联到实例，也可以使用Neutron:FloatingIP资源创建一个浮动IP，再使用OS::Neutron:FloatingIPAssociation资源将其关联到实例。

1. 管理网络

使用OS::Neutron::Net资源创建网络，使用OS::Neutron::Subnet资源创建网络的子网。下面是一个简单示例。

resources:

new\_net:

type: OS::Neutron::Net

net\_subnet:

type: OS::Neutron::Subnet

properties:

network\_id: { get\_resource: new\_net }

cidr: "10.10.9.0/24"

dns\_nameservers: [ "223.5.5.5", "119.29.29.29" ]

ip\_version: 4

使用OS::Neutron::Router资源创建路由器，可以通过external\_gateway\_info属性为路由器定义网关。

1. 管理卷存储

使用OS::Cinder::Volume资源创建一个块存储卷，再使用OS::Cinder::VolumeAttachment将卷连接到实例。

示例如下：

resources:

new\_volume:

type: OS::Cinder::Volume

properties:

size: 1

new\_instance:

type: OS::Nova:Server

properties:

flavor: ml.nano

image: cirros

volume\_attachment:

type: OS::Cinder::VolumeAttachment

properties:

volume\_id: { get\_resource: new\_volume }

instance\_uuid: { get\_resource: new\_instance }

如果要从一个卷启动实例，可以使用OS::Nova::Server资源的block\_device\_mapping属性来定义要用样式启动该实例的卷。

14.5.2 对软件配置的编排

Heat提供多种资源类型来支持对软件配置部署的编排，例如OS::Heat::CloudConfig用于实例引导程序启动时的配置，由OS::Nova::Server引用；OS::Heat::SoftwareConfig用于描述软件配置；OS::Heat::SoftwareDeployment用于执行软件部署；OS::Heat::SoftwareDeploymentGroup用于对一组实例执行软件部署。

软件配置大致分为以下3种类型。

1. 定制构建的镜像
2. 用户数据（user-data）启动脚本和cloud-init

启动实例时可以定义传递给该实例的用户数据，用户数据可通过配置驱动器（config-drive）或元数据服务获得。可使用user-data属性定义实例启动执行的脚本。

resources:

my\_server\_test\_cloud-init:

type: OS::Nova::Server

properties:

#image，flavor，etc

user\_data\_format: RAW

config\_drive: True

user\_data: |

#!/bin/bash

touch /root/test\_cloud\_init.txt

echo "test\_cloud\_init..." >/root/test\_cloud\_init.txt

启动配置脚本也可以作为自身的资源来管理，允许一次定义配置，在多个服务器资源中运行，这些software-config资源通过专用的Orchestration API调用来存储和检索。资源OS::Heat::SoftwareConfig用于存储由脚本提供的配置。示例如下：

resources:

boot\_scripts:

type: OS::Heat::SoftwareConfig

properties:

group: ungrouped

config: |

#!/bin/bash

echo "Running boot\_scripts...."

#....

server\_with\_boot\_scripts:

type: OS::Nova:Server

properties:

# flavor,image etc

user\_data\_format: SOFTWARE\_CONFIG

user\_data: { get\_resource boot\_scripts }

1. 软件部署资源

OS::Heat::SoftwareDeployment资源可在一个实例的生命周期内添加或删除任意数量的软件配置。

OS::Heat::SoftwareConfig资源用于存储软件配置，而OS::Heat::SoftwareDeployment将一个资源关联到实例。OS::Heat::SoftwareConfig的group属性指定熬使用配置内容的工具。

配置输入可以映射到Shell环境变量脚本，脚本可以通过写入名为$heat\_outputs\_path.output的文件与输出进行通信。

OpenStack Heat应用部署参考:

<https://www.sohu.com/a/106842671_355135>

<https://blog.51cto.com/11555417/2438861>

Heat两个简单示例：

OpenStack社区模板参考：

git clone git://git.openstack.org/openstack/heat-templates