Red Hat OpenStack Platform 16.1

Scaling Deployments with Compute Cells

A guide to creating and configuring a multi-cell overcloud

Last Updated: 2022-04-11

Red Hat OpenStack Platform 16.1 Scaling Deployments with Compute Cells

A guide to creating and configuring a multi-cell overcloud

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide provides concepts and procedures for cloud administrators to configure and manage cells to group Compute nodes in a large Red Hat OpenStack Platform (RHOSP) deployment.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

CHAPTER 1. MULTI-CELL OVERCLOUD DEPLOYMENTS

You can use cells to divide Compute nodes in large deployments into groups, each with a message queue and dedicated database that contains instance information.

By default, director installs the overcloud with a single cell for all Compute nodes. This cell contains all the Compute services and databases, and all the instances and instance metadata. For larger deployments, you can deploy the overcloud with multiple cells to accommodate a larger number of Compute nodes. You can add cells to your environment when you install a new overcloud or at any time afterwards.

In multi-cell deployments, each cell runs standalone copies of the cell-specific Compute services and databases, and stores instance metadata only for instances in that cell. Global information and cell mappings are stored in the global Controller cell, which provides security and recovery in case one of the cells fails.

CAUTION

If you add cells to an existing overcloud, the conductor in the default cell also performs the role of the super conductor. This has a negative effect on conductor communication with the cells in the deployment, and on the performance of the overcloud. Also, if you take the default cell offline, you take the super conductor offline as well, which stops the entire overcloud deployment. Therefore, to scale an existing overcloud, do not add any Compute nodes to the default cell. Instead, add Compute nodes to the new cells you create, allowing the default cell to act as the super conductor.

To create a multi-cell overcloud, you must perform the following tasks:

1. Configure and deploy your overcloud to handle multiple cells.
2. Create and provision the new cells that you require within your deployment.
3. Add Compute nodes to each cell.
4. Add each Compute cell to an availability zone.

1.1. PREREQUISITES

- You have deployed a basic overcloud with the required number of Controller nodes.

1.2. GLOBAL COMPONENTS AND SERVICES

The following components are deployed in a Controller cell once for each overcloud, regardless of the number of Compute cells.

Compute API

Provides the external REST API to users.

Compute scheduler

Determines on which Compute node to assign the instances.

Placement service

Monitors and allocates Compute resources to the instances.

API database

Used by the Compute API and the Compute scheduler services to track location information about instances, and provides a temporary location for instances that are built but not scheduled.
In multi-cell deployments, this database also contains cell mappings that specify the database connection for each cell.

cell0 database

Dedicated database for information about instances that failed to be scheduled.

Super conductor

This service exists only in multi-cell deployments to coordinate between the global services and each Compute cell. This service also sends failed instance information to the **cell0** database.

1.3. CELL-SPECIFIC COMPONENTS AND SERVICES

The following components are deployed in each Compute cell.

Cell database

Contains most of the information about instances. Used by the global API, the conductor, and the Compute services.

Conductor

Coordinates database queries and long-running tasks from the global services, and insulates Compute nodes from direct database access.

Message queue

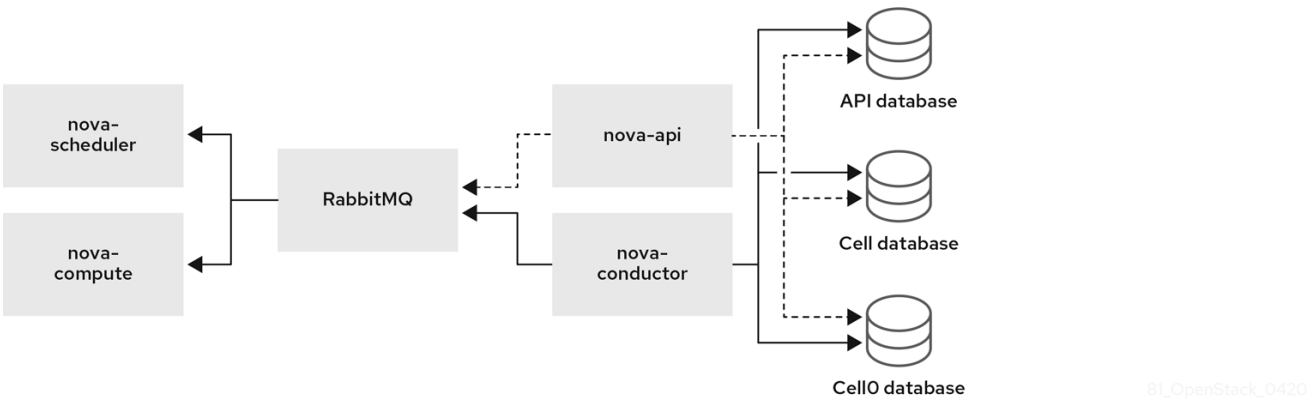
Messaging service used by all services to communicate with each other within the cell and with the global services.

1.4. CELL DEPLOYMENTS ARCHITECTURE

The default overcloud that director installs has a single cell for all Compute nodes. You can scale your overcloud by adding more cells, as illustrated by the following architecture diagrams.

Single-cell deployment architecture

The following diagram shows an example of the basic structure and interaction in a default single-cell overcloud.

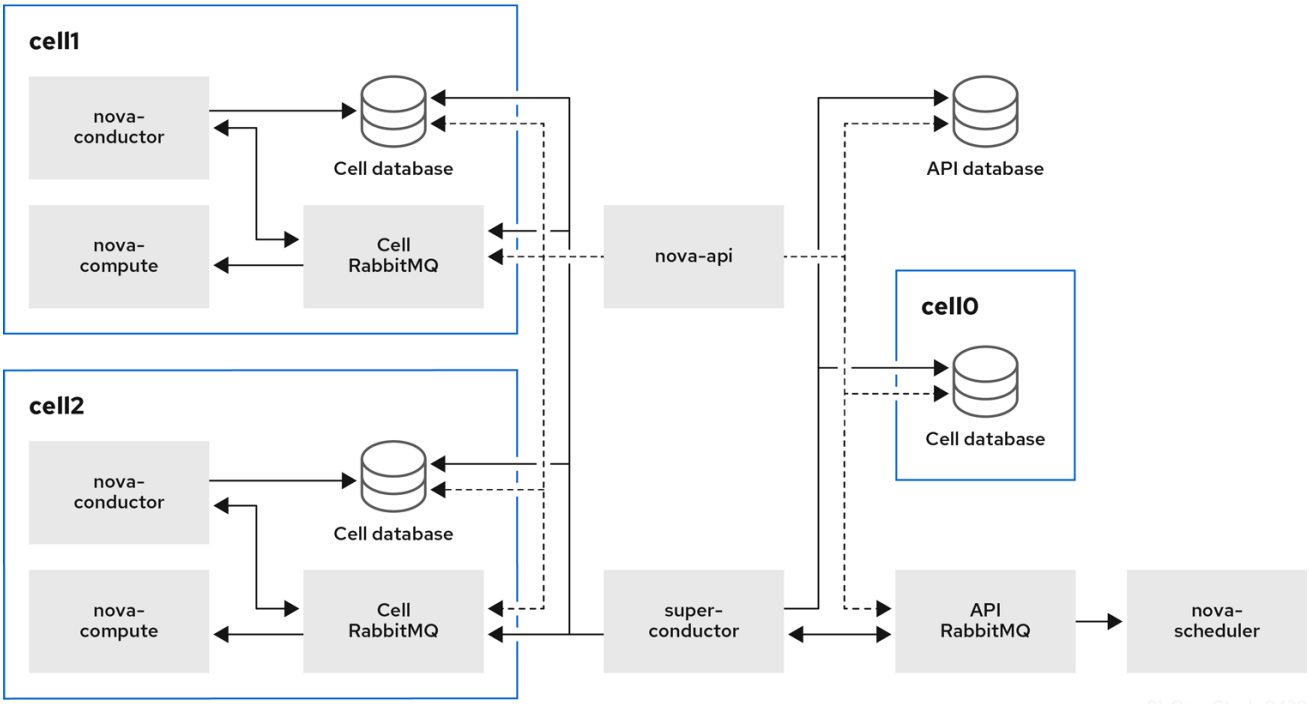


In this deployment, all services are configured to use a single conductor to communicate between the Compute API and the Compute nodes, and a single database stores all live instance data.

In smaller deployments this configuration might be sufficient, but if any global API service or database fails, the entire Compute deployment cannot send or receive information, regardless of high availability configurations.

Multi-cell deployment architecture

The following diagram shows an example of the basic structure and interaction in a custom multi-cell overcloud.



In this deployment, the Compute nodes are divided to multiple cells, each with their own conductor, database, and message queue. The global services use the super conductor to communicate with each cell, and the global database contains only information required for the whole overcloud.

The cell-level services cannot access global services directly. This isolation provides additional security and fail-safe capabilities in case of cell failure.

IMPORTANT

Do not run any Compute services on the first cell, which is named "default". Instead, deploy each new cell containing the Compute nodes separately.

1.5. CONSIDERATIONS FOR MULTI-CELL DEPLOYMENTS

Maximum number of Compute nodes in a multi-cell deployment

The maximum number of Compute nodes is 500 across all cells.

Cross-cell instance migrations

Migrating an instance from a host in one cell to a host in another cell is not supported. This limitation affects the following operations:

- cold migration
- live migration

- unshelve
- resize
- evacuation

Service quotas

Compute service quotas are calculated dynamically at each resource consumption point, instead of statically in the database. In multi-cell deployments, unreachable cells cannot provide usage information in real-time, which might cause the quotas to be exceeded when the cell is reachable again.
You can use the Placement service and API database to configure the quota calculation to withstand failed or unreachable cells.

API database

The Compute API database is always global for all cells and cannot be duplicated for each cell.

Console proxies

You must configure console proxies for each cell, because console token authorizations are stored in cell databases. Each console proxy server needs to access the **database.connection** information of the corresponding cell database.

Compute metadata API

If you use the same network for all the cells in your multiple cell environment, you must run the Compute metadata API globally so that it can bridge between the cells. When the Compute metadata API is run globally it needs access to the **api_database.connection** information.
If you deploy a multiple cell environment with routed networks, you must run the Compute metadata API separately in each cell to improve performance and data isolation. When the Compute metadata API runs in each cell, the **neutron-metadata-agent** service must point to the corresponding **nova-api-metadata** service.

You use the parameter **NovaLocalMetadataPerCell** to control where the Compute metadata API runs.

CHAPTER 2. CONFIGURING AND DEPLOYING A MULTI-CELL ENVIRONMENT WITH THE SAME NETWORKS

To configure your Red Hat OpenStack (RHOSP) deployment to handle multiple cells by using the same networks, you must perform the following tasks:

1. Extract parameter information from the control plane of the overcloud stack.
2. Create a cell roles file. You can use the default **Compute** role for the Compute nodes in a cell, and the dedicated **CellController** role for the cell controller node. You can also create custom roles for use in your multi-cell environment, such as a custom role for each cell stack. For more information on creating custom roles, see Composable services and custom roles.
3. Configure a cell controller flavor for the **CellController** role.

NOTE

If you created a custom role for your multi-cell environment, you must also configure a flavor for the custom role.

4. Configure each cell.
5. Deploy each cell stack.

2.1. EXTRACTING PARAMETER INFORMATION FROM THE OVERCLOUD STACK CONTROL PLANE

Extract parameter information from the first cell, named **default**, in the basic overcloud stack.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

```
[stack@director ~]$ source ~/stackrc
```

3. Export the cell configuration and password information from the **default** cell in the overcloud stack to a new common environment file for the multi-cell deployment:

```
(undercloud)$ openstack overcloud cell export \
 --output-file common/default_cell_export.yaml cell1
```

This command exports the **EndpointMap**, **HostsEntry**, **AllNodesConfig**, **GlobalConfig** parameters, and the password information, to the common environment file.

TIP

If the environment file already exists, enter the command with the **--force-overwrite** or **-f** option.

2.2. CREATING A CELL ROLES FILE

You can create a common cell roles file for use by all cell stacks when the stacks use the same network and no custom roles are required.

Procedure

- Generate a new roles data file named **cell_roles_data.yaml** that includes the **Compute** and **CellController** roles:

```
(undercloud)$ openstack overcloud roles generate \
  --roles-path /usr/share/openstack-tripleo-heat-templates/roles \
  -o common/cell_roles_data.yaml Compute CellController
```

2.3. DESIGNATING A HOST FOR THE CELLCONTROLLER ROLE

To designate a bare metal node for the **CellController** role, you must configure a flavor and resource class to use to tag the node for the **CellController** role. The following procedure creates a flavor and a bare metal resource class for the **CellController** role.

TIP

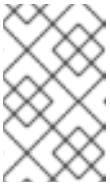
If you created a custom role for your multiple cell environment, you can follow this procedure to configure the flavor and resource class for the custom role, by substituting the cell controller names with the name of your custom role.

Procedure

1. Create the **cellcontroller** overcloud flavor for the cell controller node:

```
(undercloud)$ openstack flavor create --id auto \
  --ram <ram_size_mb> --disk <disk_size_gb> \
  --vcpus <no_vcpus> cellcontroller
```

   - Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
   - Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
   - Replace **<no_vcpus>** with the number of CPUs on the bare metal node.



NOTE

These properties are not used for scheduling instances. However, the Compute scheduler does use the disk size to determine the root partition size.

2. Retrieve a list of your nodes to identify their UUIDs:

```
(undercloud)$ openstack baremetal node list
```

3. Tag each bare metal node that you want to designate as a cell controller with a custom cell controller resource class:

```
(undercloud)$ openstack baremetal node set \
  --resource-class baremetal.CELL-CONTROLLER <node>
```

Replace **<node>** with the ID of the bare metal node.

4. Associate the **cellcontroller** flavor with the custom cell controller resource class:

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_CELL_CONTROLLER=1 \
  cellcontroller
```

To determine the name of a custom resource class that corresponds to a resource class of a Bare Metal service node, convert the resource class to uppercase, replace each punctuation mark with an underscore, and prefix with **CUSTOM_**.

NOTE

A flavor can request only one instance of a bare metal resource class.

5. Set the following flavor properties to prevent the Compute scheduler from using the bare metal flavor properties to schedule instances:

```
(undercloud)$ openstack flavor set \
 --property resources:VCPU=0 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 cellcontroller
```

2.4. CONFIGURING AND DEPLOYING EACH CELL STACK WITH THE SAME NETWORK

You must configure each cell stack to use the networks of the overcloud stack, and identify the cell as an additional cell in the deployment. You must also configure the node flavors, and the number of Controller and Compute nodes in the cell.

Procedure

1. Create a new directory for the new cells:

```
(undercloud)$ mkdir cells
```

2. Create a new environment file for each additional cell in the cell directory, **cells**, for cell-specific parameters, for example, **/cells/cell1.yaml**.

3. Add the following parameters to each environment file, updating the parameter values for each cell in your deployment:

```
resource_registry:
  OS::TripleO::Network::Ports::OVNDBsVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/noop.yaml

parameter_defaults:
  # Disable network creation in order to use the `network_data.yaml` file from the overcloud
stack,
  # and create ports for the nodes in the separate stacks on the existing networks.
```

```
ManageNetworks: false

# Specify that this is an additional cell
NovaAdditionalCell: True

# The DNS names for the VIPs for the cell
CloudName: cell1.ooo.test
CloudNameInternal: cell1.internalapi.ooo.test
CloudNameStorage: cell1.storage.ooo.test
CloudNameStorageManagement: cell1.storagemgmt.ooo.test
CloudNameCtlplane: cell1.ctlplane.ooo.test

# Map the flavors to use for the CellController and Compute roles
OvercloudCellControllerFlavor: cellcontroller
OvercloudComputeFlavor: compute

# Number of controllers/computes in the cell
CellControllerCount: 3
ComputeCount: 1

# Node names must be unique across all cells
ComputeHostnameFormat: 'cell1-compute-%index%'
CellControllerHostnameFormat: 'cell1-cellcontroller-%index%'
```

4. To allocate a network resource to the cell and register cells to the network, add the following parameters to each environment file:

```
resource_registry:
  OS::TripleO::CellController::Net::SoftwareConfig: /home/stack/templates/nic-configs/cellcontroller.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
```

5. Add the environment files to the stack with your other environment files and deploy the cell stack:

```
(undercloud)$ openstack overcloud deploy --templates \
 --stack cell1 \
 -e [your environment files] \
 -r $HOME/common/cell_roles_data.yaml \
 -e $HOME/common/default_cell_export.yaml \
 -e $HOME/cells/cell1.yaml
```

Repeat this step for each cell stack until all your cell stacks are deployed.

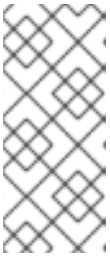
2.5. NEXT STEPS

- Creating and managing the cell within the Compute service

CHAPTER 3. CONFIGURING AND DEPLOYING A MULTI-CELL ENVIRONMENT WITH ROUTED NETWORKS

To configure your Red Hat OpenStack (RHOSP) deployment to handle multiple cells with routed networks, you must perform the following tasks:

1. Prepare the control plane for cell network routing on the overcloud stack.
2. Extract parameter information from the control plane of the overcloud stack.
3. Configure the cell network routing on the cell stacks.
4. Create cell roles files for each stack. You can use the default **Compute** role as a base for the Compute nodes in a cell, and the dedicated **CellController** role as a base for the cell controller node. You can also create custom roles for use in your multi-cell environment. For more information on creating custom roles, see Composable services and custom roles.
5. Configure a flavor for each custom role you create.



NOTE

This procedure is for an environment with a single control plane network. If your environment has multiple control plane networks, such as a spine leaf network environment, then you must also create a flavor for each role in each leaf network so that you can tag nodes into each leaf. For more information, see Creating flavors and tagging nodes for leaf networks.

6. Configure each cell.
7. Deploy each cell stack.

3.1. PREREQUISITES

- You have configured your undercloud for routed networks. For more information, see Configuring routed spine-leaf in the undercloud.

3.2. PREPARING THE CONTROL PLANE AND DEFAULT CELL FOR CELL NETWORK ROUTING

You must configure routes on the overcloud stack for the overcloud stack to communicate with the cells. To achieve this, create a network data file that defines all networks and subnets in the main stack, and use this file to deploy both the overcloud stack and the cell stacks.

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** file:

```
[stack@director ~]$ source ~/stackrc
```

3. Create a new directory for the common stack configuration:

```
(undercloud)$ mkdir common
```

4. Copy the default **network_data_subnets_routed.yaml** file to your **common** directory to add a composable network for your overcloud stack:

```
(undercloud)$ cp /usr/share/openstack-tripleo-heat-
templates/network_data_subnets_routed.yaml
~/common/network_data_routed_multi_cell.yaml
```

For more information on composable networks, see Custom composable networks in the *Advanced Overcloud Customization* guide.

5. Update the configuration in **/common/network_data_routed_multi_cell.yaml** for your network, and update the cell subnet names for easy identification, for example, change **internal_api_leaf1** to **internal_api_cell1**.

6. Ensure that the interfaces in the NIC template for each role include **<network_name>InterfaceRoutes**, for example:

```
-
  type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  addresses:
  -
    ip_netmask:
      get_param: InternalApiIpSubnet
  routes:
    get_param: InternalApiInterfaceRoutes
```

7. Add the **network_data_routed_multi_cell.yaml** file to the overcloud stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
 --stack overcloud \
 -n /home/stack/common/network_data_routed_multi_cell.yaml \
 -e [your environment files]
```

3.3. EXTRACTING PARAMETER INFORMATION FROM THE OVERCLOUD STACK CONTROL PLANE

Extract parameter information from the first cell, named **default**, in the basic overcloud stack.

Procedure

1. Log in to the undercloud as the **stack** user.

2. Source the **stackrc** file:

```
[stack@director ~]$ source ~/stackrc
```

3. Export the cell configuration and password information from the **default** cell in the overcloud stack to a new common environment file for the multi-cell deployment:

```
(undercloud)$ openstack overcloud cell export \
 --output-file common/default_cell_export.yaml cell1
```

This command exports the **EndpointMap**, **HostsEntry**, **AllNodesConfig**, **GlobalConfig** parameters, and the password information, to the common environment file.

TIP

If the environment file already exists, enter the command with the **--force-overwrite** or **-f** option.

3.4. CREATING CELL ROLES FILES FOR ROUTED NETWORKS

When each stack uses a different network, create a cell roles file for each cell stack that includes a custom cell role.

NOTE

You must create a flavor for each custom role. For more information, see Designating hosts for cell roles.

Procedure

1. Generate a new roles data file that includes the **CellController** role, along with other roles you need for the cell stack. The following example generates the roles data file **cell1_roles_data.yaml**, which includes the roles **CellController** and **Compute**:

```
(undercloud)$ openstack overcloud roles generate \
  --roles-path /usr/share/openstack-tripleo-heat-templates/roles \
  -o cell1/cell1_roles_data.yaml \
  Compute:ComputeCell1 \
  CellController:CellControllerCell1
```

2. Add the **HostnameFormatDefault** to each role definition in your new cell roles file:

```
- name: ComputeCell1
  ...
  HostnameFormatDefault: '%stackname%-compute-cell1-%index%'
  ServicesDefault:
  ...
  networks:
  ...
- name: CellControllerCell1
  ...
  HostnameFormatDefault: '%stackname%-cellcontrol-cell1-%index%'
  ServicesDefault:
  ...
  networks:
  ...
```

3. Add the Networking service (neutron) DHCP and Metadata agents to the **ComputeCell1** and **CellControllerCell1** roles, if they are not already present:

```
- name: ComputeCell1
  ...
  HostnameFormatDefault: '%stackname%-compute-cell1-%index%'
  ServicesDefault:
  - OS::TripleO::Services::NeutronDhcpAgent
  - OS::TripleO::Services::NeutronMetadataAgent
  ...
  networks:
  ...
- name: CellControllerCell1
  ...
  HostnameFormatDefault: '%stackname%-cellcontrol-cell1-%index%'
  ServicesDefault:
  - OS::TripleO::Services::NeutronDhcpAgent
  - OS::TripleO::Services::NeutronMetadataAgent
  ...
  networks:
  ...
```

4. Add the subnets you configured in **network_data_routed_multi_cell.yaml** to the **ComputeCell1** and **CellControllerCell1** roles:

```
- name: ComputeCell1
  ...
  networks:
    InternalApi:
      subnet: internal_api_subnet_cell1
    Tenant:
      subnet: tenant_subnet_cell1
    Storage:
      subnet: storage_subnet_cell1
...
- name: CellControllerCell1
  ...
  networks:
    External:
      subnet: external_subnet
    InternalApi:
      subnet: internal_api_subnet_cell1
    Storage:
      subnet: storage_subnet_cell1
    StorageMgmt:
      subnet: storage_mgmt_subnet_cell1
    Tenant:
      subnet: tenant_subnet_cell1
```

3.5. DESIGNATING HOSTS FOR CELL ROLES

To designate a bare metal node for a cell role, you must configure a flavor and resource class to use to tag the node for the cell role. Perform the following procedure to create a flavor and a bare metal resource class for the **cellcontrollercell1** role. Repeat this procedure for each custom role, by substituting the cell controller names with the name of your custom role.

Procedure

1. Create the **cellcontrollercell1** overcloud flavor for the **cell1** controller node:

```
(undercloud)$ openstack flavor create --id auto \
 --ram <ram_size_mb> --disk <disk_size_gb> \
 --vcpus <no_vcpus> cellcontrollercell1
```

   - Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
   - Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
   - Replace **<no_vcpus>** with the number of CPUs on the bare metal node.

NOTE

These properties are not used for scheduling instances. However, the Compute scheduler does use the disk size to determine the root partition size.

2. Retrieve a list of your nodes to identify their UUIDs:

```
(undercloud)$ openstack baremetal node list
```

3. Tag each bare metal node that you want to designate as a cell controller with a custom cell controller resource class:

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.CELL-CONTROLLER <node>
```

   Replace **<node>** with the ID of the bare metal node.

4. Associate the **cellcontrollercell1** flavor with the custom cell controller resource class:

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_CELL_CONTROLLER=1 \
  cellcontrollercell1
```

   To determine the name of a custom resource class that corresponds to a resource class of a Bare Metal service node, convert the resource class to uppercase, replace each punctuation mark with an underscore, and prefix with **CUSTOM_**.

NOTE

A flavor can request only one instance of a bare metal resource class.

5. Set the following flavor properties to prevent the Compute scheduler from using the bare metal flavor properties to schedule instances:

```
(undercloud)$ openstack flavor set \
 --property resources:VCPU=0 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0 cellcontrollercell1
```

3.6. CONFIGURING AND DEPLOYING EACH CELL STACK WITH ROUTED NETWORKS

Perform the following procedure to configure one cell stack, **cell1**. Repeat the procedure for each additional cell stack you want to deploy until all your cell stacks are deployed.

Procedure

1. Create a new environment file for the additional cell in the cell directory for cell-specific parameters, for example, **/home/stack/cell1/cell1.yaml**.

2. Add the following parameters to the environment file:

```
resource_registry:
  OS::TripleO::CellControllerCell1::Net::SoftwareConfig: /home/stack/templates/nic-configs/cellcontroller.yaml
  OS::TripleO::ComputeCell1::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Network::Ports::OVNDBsVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml
  OS::TripleO::Network::Ports::RedisVipPort: /usr/share/openstack-tripleo-heat-templates/network/ports/noop.yaml

parameter_defaults:
  #Disable network creation in order to use the `network_data.yaml` file from the overcloud stack,
  # and create ports for the nodes in the separate stacks on the existing networks.
  ManageNetworks: false

  # Specify that this is an additional cell
  NovaAdditionalCell: True

  # The DNS names for the VIPs for the cell
  CloudName: cell1.ooo.test
  CloudNameInternal: cell1.internalapi.ooo.test
  CloudNameStorage: cell1.storage.ooo.test
  CloudNameStorageManagement: cell1.storagemgmt.ooo.test
  CloudNameCtlplane: cell1.ctlplane.ooo.test

  # Map the flavors to use for the CellController and Compute roles
  OvercloudCellControllerCell1Flavor: cellcontrollercell1
  OvercloudComputeCell1Flavor: computecell1

  # Number of controllers/computes in the cell
  CellControllerCell1Count: 3
  ComputeCell1Count: 1
```

3. To run the Compute metadata API in each cell instead of in the global Controller, add the following parameter to your cell environment file:

```
parameter_defaults:
  NovaLocalMetadataPerCell: True
```

4. Add the virtual IP address (VIP) information for the cell to your cell environment file:

```
parameter_defaults:
  ...
  VipSubnetMap:
    InternalApi: internal_api_cell1
    Storage: storage_cell1
    StorageMgmt: storage_mgmt_cell1
    External: external_subnet
```

This creates virtual IP addresses on the subnet associated with the L2 network segment that the cell Controller nodes are connected to.

5. Add the environment files to the stack with your other environment files and deploy the cell stack:

```
(undercloud)$ openstack overcloud deploy --templates \
 --stack cell1 \
 -e [your environment files] \
 -r /home/stack/cell1/cell1_roles_data.yaml \
 -n /home/stack/common/network_data_spine_leaf.yaml \
 -e /home/stack/common/default_cell_export.yaml \
 -e /home/stack/cell1/cell1.yaml
```

3.7. ADDING A NEW CELL SUBNET AFTER DEPLOYMENT

To add a new cell subnet to your overcloud stack after you have deployed your multi-cell environment, you must update the value of **NetworkDeploymentActions** to include **'UPDATE'**.

Procedure

1. Add the following configuration to an environment file for the overcloud stack to update the network configuration with the new cell subnet:

```
parameter_defaults:
  NetworkDeploymentActions: ['CREATE','UPDATE']
```

2. Add the configuration for the new cell subnet to **/common/network_data_routed_multi_cell.yaml**.

3. Deploy the overcloud stack:

```
(undercloud)$ openstack overcloud deploy --templates \
 --stack overcloud \
 -n /home/stack/common/network_data_routed_multi_cell.yaml \
 -e [your environment files]
```

4. Optional: Reset **NetworkDeploymentActions** to the default for the next deployment:

```
parameter_defaults:
  NetworkDeploymentActions: ['CREATE']
```

3.8. NEXT STEPS

- Creating and managing the cell within the Compute service

CHAPTER 4. CREATING AND MANAGING THE CELL WITHIN THE COMPUTE SERVICE

After you have deployed the overcloud with your cell stacks, you must create the cell within the Compute service. To create the cell within the Compute service you create entries for the cell and message queue mappings in the global API database. You can then add Compute nodes to the cells by running the cell host discovery on one of the Controller nodes.

To create your cells, you must perform the following tasks:

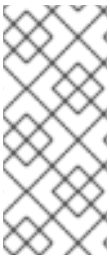
1. Use the **nova-manage** utility to create the cell and message queue mapping records in the global API database.
2. Add Compute nodes to each cell.
3. Create an availability zone for each cell.
4. Add all the Compute nodes in each cell to the availability zone for the cell.

4.1. PREREQUISITES

- You have configured and deployed your overcloud with multiple cells.

4.2. CREATING THE CELL WITHIN THE COMPUTE SERVICE

After you deploy the overcloud with a new cell stack, you must create the cell within the Compute service. To create the cell within the Compute service you create entries for the cell and message queue mappings in the global API database.



NOTE

You must repeat this process for each cell that you create and launch. You can automate the steps in an Ansible playbook. For an example of an Ansible playbook, see the Create the cell and discover Compute nodes section of the OpenStack community documentation. Community documentation is provided as-is and is not officially supported.

Procedure

1. Get the IP addresses of the control plane and cell controller:

```
$ CTRL_IP=$(openstack server list -f value -c Networks --name overcloud-controller-0 | sed 's/ctlplane=//')
$ CELL_CTRL_IP=$(openstack server list -f value -c Networks --name cell1-cellcontroller-0 | sed 's/ctlplane=//')
```

2. Add the cell information to all Controller nodes. This information is used to connect to the cell endpoint from the undercloud. The following example uses the prefix **cell1** to specify only the cell systems and exclude the controller systems:

```
(undercloud)$ CELL_INTERNALAPI_INFO=$(ssh heat-admin@${CELL_CTRL_IP} \
 egrep cell1.*\.internalapi /etc/hosts)
(undercloud)$ ansible -i /usr/bin/tripleo-ansible-inventory \
```

```
Controller -b -m lineinfile -a "dest=/etc/hosts line=\"$CELL_INTERNALAPI_INFO\""
```

3. Get the message queue endpoint for the controller cell from the **transport_url** parameter, and the database connection for the controller cell from the **database.connection** parameter:

```
(undercloud)$ CELL_TRANSPORT_URL=$(ssh heat-admin@${CELL_CTRL_IP} \
 sudo crudini --get /var/lib/config-data/nova/etc/nova/nova.conf \
 DEFAULT transport_url)
(undercloud)$ CELL_MYSQL_VIP=$(ssh heat-admin@${CELL_CTRL_IP} \
 sudo crudini --get /var/lib/config-data/nova/etc/nova/nova.conf \
 database connection | awk -F[@/] '{print $4}')
```

4. Log in to one of the global Controller nodes and create the cell:

```
$ ssh heat-admin@${CTRL_IP} sudo podman \
 exec -i -u root nova_api \
 nova-manage cell_v2 create_cell --name cell1 \
 --database_connection "{scheme}://{username}:{password}@$CELL_MYSQL_VIP/nova?
{query}" \
 --transport-url "$CELL_TRANSPORT_URL"
```

5. Check that the cell is created and appears in the cell list:

```
$ ssh heat-admin@${CTRL_IP} sudo podman \
 exec -i -u root nova_api \
 nova-manage cell_v2 list_cells --verbose
```

6. Restart the Compute services on the Controller nodes:

```
$ ansible -i /usr/bin/tripleo-ansible-inventory Controller -b -a \
"systemctl restart tripleo_nova_api tripleo_nova_conductor tripleo_nova_scheduler"
```

7. Check that the cell controller services are provisioned:

```
(overcloud)$ openstack compute service list -c Binary -c Host -c Status -c State
+----------------+-------------------------+---------+-------+
| Binary         | Host                    | Status  | State |
+----------------+-------------------------+---------+-------+
| nova-conductor | controller-0.ostest     | enabled | up    |
| nova-scheduler | controller-0.ostest     | enabled | up    |
| nova-conductor | cellcontroller-0.ostest | enabled | up    |
| nova-compute   | compute-0.ostest        | enabled | up    |
| nova-compute   | compute-1.ostest        | enabled | up    |
+----------------+-------------------------+---------+-------+
```

4.3. ADDING COMPUTE NODES TO A CELL

Run the cell host discovery on one of the Controller nodes to discover the Compute nodes and update the API database with the node-to-cell mappings.

Procedure

1. Log in to the undercloud as the **stack** user.

2. Get the IP address of the control plane for the cell and enter the host discovery command to expose and assign Compute hosts to the cell:

```
$ CTRL_IP=$(openstack server list -f value -c Networks --name overcloud-controller-0 | sed
's/ctlplane=//')

$ ssh heat-admin@${CTRL_IP} sudo podman exec -i -u root nova_api \
  nova-manage cell_v2 discover_hosts --by-service --verbose
```

3. Verify that the Compute hosts were assigned to the cell:

```
$ ssh heat-admin@${CTRL_IP} sudo podman exec -i -u root nova_api \
  nova-manage cell_v2 list_hosts
```

4.4. CREATING A CELL AVAILABILITY ZONE

You must create an availability zone (AZ) for each cell to ensure that instances created on the Compute nodes in that cell are migrated only to other Compute nodes in the same cell. Migrating instances between cells is not supported.

After you create the cell AZ you must add all the Compute nodes in the cell to the cell AZ. The default cell must be in a different availability zone from the Compute cells.

Procedure

1. Source the **overcloudrc** file:

```
(undercloud)$ source ~/overcloudrc
```

2. Create the AZ for the cell:

```
(overcloud)# openstack aggregate create \
 --zone <availability_zone> \
 <aggregate_name>
```

   - Replace **<availability_zone>** with the name you want to assign to the availability zone.
   - Replace **<aggregate_name>** with the name you want to assign to the host aggregate.

3. Optional: Add metadata to the availability zone:

```
(overcloud)# openstack aggregate set --property <key=value> \
  <aggregate_name>
```

   - Replace **<key=value>** with your metadata key-value pair. You can add as many key-value properties as required.
   - Replace **<aggregate_name>** with the name of the availability zone host aggregate.

4. Retrieve a list of the Compute nodes assigned to the cell:

```
$ ssh heat-admin@${CTRL_IP} sudo podman exec -i -u root nova_api \
  nova-manage cell_v2 list_hosts
```

5. Add the Compute nodes assigned to the cell to the cell availability zone:

```
(overcloud)# openstack aggregate add host <aggregate_name> \
 <host_name>
```

- Replace **<aggregate_name>** with the name of the availability zone host aggregate to add the Compute node to.
- Replace **<host_name>** with the name of the Compute node to add to the availability zone.

NOTE

- You cannot use the **OS::TripleO::Services::NovaAZConfig** parameter to automatically create the AZ during deployment, because the cell is not created at this stage.
- Migrating instances between cells is not supported. To move an instance to a different cell, you must delete it from the old cell and re-create it in the new cell.

For more information on host aggregates and availability zones, see Creating and managing host aggregates.

4.5. DELETING A COMPUTE NODE FROM A CELL

To delete a Compute node from a cell, you must delete all instances from the cell and delete the host names from the Placement database.

Procedure

1. Delete all instances from the Compute nodes in the cell.

NOTE

Migrating instances between cells is not supported. You must delete the instances and re-create them in another cell.

2. On one of the global Controllers, delete all Compute nodes from the cell.

```
$ CTRL_IP=$(openstack server list -f value -c Networks --name overcloud-controller-0 | sed
's/ctlplane=//')

$ ssh heat-admin@${CTRL_IP} sudo podman  \
 exec -i -u root nova_api \
 nova-manage cell_v2 list_hosts

$ ssh heat-admin@${CTRL_IP} sudo podman  \
 exec -i -u root nova_api \
 nova-manage cell_v2 delete_host --cell_uuid <uuid> --host <compute>
```

3. Delete the resource providers for the cell from the Placement service, to ensure that the host name is available in case you want to add Compute nodes with the same host name to another cell later:

```
(undercloud)$ source ~/overcloudrc

(overcloud)$ openstack resource provider list
+--------------------------------------+---------------------------------------+------------+
| uuid                                 | name                                  | generation |
+--------------------------------------+---------------------------------------+------------+
| 9cd04a8b-5e6c-428e-a643-397c9bebcc16 | computecell1-novacompute-0.site1.test |
11 |
+--------------------------------------+---------------------------------------+------------+

(overcloud)$ openstack resource provider  \
 delete 9cd04a8b-5e6c-428e-a643-397c9bebcc16
```

4.6. DELETING A CELL

To delete a cell, you must first delete all instances and Compute nodes from the cell, as described in Deleting a Compute node from a cell . Then, you delete the cell itself and the cell stack.

Procedure

1. On one of the global Controllers, delete the cell.

```
$ CTRL_IP=$(openstack server list -f value -c Networks --name overcloud-controller-0 | sed
's/ctlplane=//')

$ ssh heat-admin@${CTRL_IP} sudo podman  \
 exec -i -u root nova_api \
 nova-manage cell_v2 list_cells

$ ssh heat-admin@${CTRL_IP} sudo podman  \
 exec -i -u root nova_api \
 nova-manage cell_v2 delete_cell --cell_uuid <uuid>
```

2. Delete the cell stack from the overcloud.

```
$ openstack stack delete <stack name> --wait --yes && openstack  \
 overcloud plan delete <stack_name>
```

NOTE

If you deployed separate cell stacks for a Controller and Compute cell, delete the Compute cell stack first and then the Controller cell stack.