Red Hat OpenStack Services on OpenShift 18.0

在 OpenShift 部署中自定义 Red Hat OpenStack Services

在 Red Hat OpenShift Container Platform 集群上自定义部署的 Red Hat OpenStack Services

Last Updated: 2024-10-19

Red Hat OpenStack Services on OpenShift 18.0 在 OpenShift 部署中自定义 Red Hat OpenStack Services

在 Red Hat OpenShift Container Platform 集群上自定义部署的 Red Hat OpenStack Services

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

了解如何在 Red Hat OpenShift Container Platform 集群上的 OpenShift control plane 上自定义 Red Hat OpenStack Services，并自定义 data plane。

目录

对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

在 JIRA 中提供文档反馈

使用 Create Issue 表单在 OpenShift (RHOSO)或更早版本的 Red Hat OpenStack Platform (RHOSP)上提供有关 Red Hat OpenStack Services 文档的反馈。当您为 RHOSO 或 RHOSP 文档创建问题时，这个问题将在 RHOSO Jira 项目中记录，您可以在其中跟踪您的反馈的进度。

要完成 Create Issue 表单，请确保您已登录到 JIRA。如果您没有红帽 JIRA 帐户，您可以在 https://issues.redhat.com 创建一个帐户。

1. 点击以下链接打开 **Create Issue** 页面： Create Issue
2. 完成 **Summary** 和 **Description** 字段。在 **Description** 字段中，包含文档 URL、章节号以及问题的详细描述。不要修改表单中的任何其他字段。
3. 点 **Create**。

第 1 章 自定义 CONTROL PLANE

OpenShift (RHOSO)控制平面上的 Red Hat OpenStack Services 包含用于管理云的 RHOSO 服务。RHOSO 服务作为 Red Hat OpenShift Container Platform (RHOCP)工作负载运行。您可以使用环境所需的服务自定义部署的 control plane。

1.1. 先决条件

- RHOSO 环境部署在 RHOCP 集群中。如需更多信息，请参阅在 OpenShift 上部署 Red Hat OpenStack Services。
- 以具有 **cluster-admin** 权限的用户身份登录到可访问 RHOCP 集群的工作站。

1.2. 启用禁用的服务

如果您通过设置 **enabled: true** 来启用禁用的服务，则必须将 **template: {}** 添加到服务定义中来为服务创建一个空模板，以确保设置了服务的默认值，或者指定一些或所有模板参数值。例如，要使用默认服务值启用 Dashboard 服务(horizon)，请在 **OpenStackControlPlane** 自定义资源(CR)中添加以下配置：

```
spec:
  ...
  horizon:
    apiOverride: {}
    enabled: true
    template: {}
```

如果要为特定服务参数设置值，请将以下配置添加到 **OpenStackControlPlane** 自定义资源(CR)中：

```
spec:
  ...
  horizon:
    apiOverride: {}
    enabled: true
    template:
      customServiceConfig: ""
      memcachedInstance: memcached
      override: {}
      preserveJobs: false
      replicas: 2
      resources: {}
      secret: osp-secret
      tls: {}
```

您指定的任何参数都设置为来自服务模板的默认值。

1.3. 将裸机置备服务(IRONIC)添加到 CONTROL PLANE

如果您希望云用户能够启动裸机实例，则必须使用裸机置备服务(ironic)配置 control plane。

流程

1. 在工作站上打开 **OpenStackControlPlane** 自定义资源(CR)文件 **openstack_control_plane.yaml**。

2. 将以下 **cellTemplates** 配置添加到 **nova** 服务配置中：

```
nova:
  apiOverride:
    route: {}
  template:
    ...
    secret: osp-secret
    cellTemplates:
      cell0:
        cellDatabaseAccount: nova-cell0
        hasAPIAccess: true
      cell1:
        cellDatabaseAccount: nova-cell1
        cellDatabaseInstance: openstack-cell1
        cellMessageBusInstance: rabbitmq-cell1
        hasAPIAccess: true
        novaComputeTemplates:
          compute-ironic: (1)
            computeDriver: ironic.IronicDriver
```

(1) Compute 服务的名称。名称限制为 20 个字符，且只能包含小写字母数字字符和 **-** 符号。

3. 创建 **ironic** 服务 pod 附加到的网络，如 **baremetal**。有关如何创建隔离网络的更多信息，*请参阅在 OpenShift 上部署 Red Hat OpenStack Services* 指南中的 为 RHOSO 网络准备 RHOCP。

4. 启用并配置 **ironic** 服务：

```
spec:
  ...
  ironic:
    enabled: true
    template:
      rpcTransport: oslo
      databaseInstance: openstack
      ironicAPI:
        replicas: 1
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
      ironicConductors:
      - replicas: 1
        storageRequest: 10G
        networkAttachments:
        - baremetal (1)
        provisionNetwork: baremetal
        customServiceConfig: |
          [neutron]
```

```
      cleaning_network = provisioning
      provisioning_network = provisioning
      rescuing_network = provisioning
  ironicInspector:
    replicas: 0
    networkAttachments:
    - baremetal
    inspectionNetwork: baremetal
  ironicNeutronAgent:
    replicas: 1
  secret: osp-secret
```

1 **baremetal** 网络的 **NetworkAttachmentDefinition** CR。

5. 更新 control plane：

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

6. 等待 RHOCP 创建与 **OpenStackControlPlane** CR 相关的资源。运行以下命令来检查状态：

```
$ oc get openstackcontrolplane -n openstack
NAME       STATUS  MESSAGE
openstack-control-plane  Unknown  Setup started
```

当状态为 "Setup complete" 时，会创建 **OpenStackControlPlane** 资源。

提示

将 **-w** 选项附加到 **get** 命令的末尾，以跟踪部署进度。

7. 通过查看 **openstack** 命名空间中的 pod 确认 control plane 已部署：

```
$ oc get pods -n openstack
```

当所有 pod 都已完成或运行时，会部署 control plane。

验证

1. 打开与 **OpenStackClient** pod 的远程 shell 连接：

```
$ oc rsh -n openstack openstackclient
```

2. 确认内部服务端点已注册到每个服务：

```
$ openstack endpoint list -c 'Service Name' -c Interface -c URL --service ironic
+--------------+-----------+---------------------------------------------------------------+
| Service Name | Interface | URL                                                           |
+--------------+-----------+---------------------------------------------------------------+
| ironic       | internal  | http://ironic-internal.openstack.svc:9292                     |
| ironic       | public    | http://ironic-public-openstack.apps.ostest.test.metalkube.org |
+--------------+-----------+---------------------------------------------------------------+
```

3. 退出 **openstackclient** pod:

```
$ exit
```

1.4. 在 CONTROL PLANE 中添加计算单元

您可以使用单元将大型部署中的 Compute 节点划分为组。每个单元都有一个专用的消息队列，运行特定于单元的计算服务和数据库的独立副本，并将实例元数据存储在该单元格中专用于实例的数据库中。

默认情况下，control plane 创建两个单元：

- **cell0** ：管理全局组件和服务（如计算调度程序和全局编排器）的控制器单元。此单元还包含专用的数据库，用于存储与无法调度到 Compute 节点的实例相关的信息。您无法将 Compute 节点连接到这个单元。
- **cell1** ：当您不创建和配置其他单元时，计算节点连接到的默认单元。

在创建 control plane 或之后，您可以在 OpenShift (RHOSO)环境的 Red Hat OpenStack Services 中添加单元格。

流程

1. 在工作站上打开 **OpenStackControlPlane** 自定义资源(CR)文件 **openstack_control_plane.yaml**。
2. 为您要添加到 RHOSO 环境的每个新单元创建一个数据库服务器：

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
  namespace: openstack
spec:
  secret: osp-secret
  ...
  galera:
    enabled: true
    templates:
      openstack: 1
        storageRequest: 5G
        secret: cell0-secret
        replicas: 1
      openstack-cell1: 2
        storageRequest: 5G
        secret: cell1-secret
        replicas: 1
      openstack-cell2: 3
        storageRequest: 5G
        secret: cell2-secret
        replicas: 1
```

1 大部分 RHOSO 服务使用的数据库，包括计算服务 **nova-api** 和 **nova-scheduler**，以及 **cell0**。

2 **cell1** 使用的数据库。

3 **cell2** 使用的数据库。

3. 为您要添加到 RHOSO 环境的每个新单元创建一个具有唯一 IP 的消息总线：

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
spec:
  secret: osp-secret
  ...
  rabbitmq:
    templates:
      rabbitmq: 1
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.85
            spec:
              type: LoadBalancer
      rabbitmq-cell1: 2
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.86
            spec:
              type: LoadBalancer
      rabbitmq-cell2: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.87
          spec:
            type: LoadBalancer
```

1 大多数 RHOSO 服务使用的消息总线，包括计算服务 **nova-api** 和 **nova-scheduler**，以及 **cell0**。

2 **cell1** 使用的消息总线。

3 **cell2** 使用的消息总线。

4. 将新单元添加到 **nova** 服务配置中的 **cellTemplates** 配置中：

```
nova:
  apiOverride:
```

```
    route: {}
  template:
    ...
    secret: osp-secret
    apiDatabaseAccount: nova-api
    cellTemplates:
      cell0:
        hasAPIAccess: true
        cellDatabaseAccount: nova-cell0
        cellDatabaseInstance: openstack
        cellMessageBusInstance: rabbitmq
      cell1:
        hasAPIAccess: true
        cellDatabaseAccount: nova-cell1
        cellDatabaseInstance: openstack-cell1
        cellMessageBusInstance: rabbitmq-cell1
      cell2: 1
        hasAPIAccess: true
        cellDatabaseAccount: nova-cell2
        cellDatabaseInstance: openstack-cell2
        cellMessageBusInstance: rabbitmq-cell2
```

1 新的 Compute 单元的名称。名称限制为 20 个字符，且只能包含小写字母数字字符和 **-** 符号。有关您可以为单元配置的属性的更多信息，请查看 **Nova** CRD 的定义：

```
$ oc describe crd nova
```

5. 更新 control plane：

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

6. 等待 RHOCP 创建与 **OpenStackControlPlane** CR 相关的资源。运行以下命令来检查状态：

```
$ oc get openstackcontrolplane -n openstack
NAME      STATUS  MESSAGE
openstack-control-plane  Unknown  Setup started
```

当状态为 "Setup complete" 时，会创建 **OpenStackControlPlane** 资源。

提示

将 **-w** 选项附加到 **get** 命令的末尾，以跟踪部署进度。

7. 可选：通过查看您创建的每个单元的 **openstack** 命名空间中的 pod 来确认 control plane 已被部署：

```
$ oc get pods -n openstack | grep cell2
nova-cell2-conductor-0          1/1     Running     2          5d20h
nova-cell2-novncproxy-0         1/1     Running     2          5d20h
openstack-cell2-galera-0        1/1     Running     2          5d20h
rabbitmq-cell2-server-0         1/1     Running     2          5d20h
```

当所有 pod 都已完成或运行时，会部署 control plane。

8. 可选：确认创建了新的单元：

```
$ oc exec -it nova-cell0-conductor-0 /bin/bash
# nova-manage cell_v2 list_cells
+-------+--------------------------------------+------------------------------------------------------------------
------------------+------------------------------------------------------------+----------+| Name | UUID |
Transport URL | Database Connection | Disabled |+-------+--------------------------------------+--
-----------------------------------------------------------------------------+----------------------------
----------------------------+----------+| cell0 | 00000000-0000-0000-0000-000000000000 |
rabbit: | mysql+pymysql://nova_cell0:****@openstack/nova_cell0 | False || cell1 | c5bf5e35-
6677-40aa-80d0-33a440cac14e | rabbit://default_user_CuUVnXz-
PvgzXvPxypU:****@rabbitmq-cell1.openstack.svc:5672 |
mysql+pymysql://nova_cell1:****@openstack-cell1/nova_cell1 | False || cell2 | c5bf5e35-
6677-40aa-80d0-33a440cac14e | rabbit://default_user_CuUVnXz-
PvgzXvPxypU:****@rabbitmq-cell2.openstack.svc:5672 |
mysql+pymysql://nova_cell2:****@openstack-cell2/nova_cell2| False |+-------+------------------
------------------+-----------------------------------------------------------------------------------------+--------
-------------------------------------------------+----------+
```

1.5. 启用 DASHBOARD 服务(HORIZON)接口

您可以为云用户通过 Web 浏览器访问云启用 Dashboard 服务(horizon)界面。

流程

1. 在工作站上打开 **OpenStackControlPlane** 自定义资源(CR)文件 **openstack_control_plane.yaml**。
2. 启用并配置 **horizon** 服务：

```
spec:
  ...
  horizon:
    apiOverride: {}
    enabled: true
    template:
      customServiceConfig: ""
      memcachedInstance: memcached
      override: {}
      preserveJobs: false
      replicas: 2 (1)
      resources: {}
      secret: osp-secret
      tls: {}
```

(1) 将 **replicas** 设置为至少 **2** 以实现高可用性。

3. 更新 control plane：

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

4. 等待 RHOCP 创建与 **OpenStackControlPlane** CR 相关的资源。运行以下命令来检查状态：

```
$ oc get openstackcontrolplane -n openstack
NAME       STATUS  MESSAGE
openstack-control-plane  Unknown  Setup started
```

当状态为 "Setup complete" 时，会创建 **OpenStackControlPlane** 资源。

提示

将 **-w** 选项附加到 **get** 命令的末尾，以跟踪部署进度。

5. 通过查看 **openstack** 命名空间中的 pod 确认 control plane 已部署：

```
$ oc get pods -n openstack
```

当所有 pod 都已完成或运行时，会部署 control plane。

6. 检索 Dashboard 服务端点 URL：

```
$ oc get horizons horizon -o jsonpath='{.status.endpoint}'
```

使用此 URL 访问 Horizon 界面。

验证

1. 要以 **admin** 用户身份登录，请从 **osp-secret** secret 中的 **AdminPassword** 参数获取 **admin** 密码：

```
$ oc get secret osp-secret -o jsonpath='{.data.AdminPassword}' | base64 -d
```

2. 打开 Web 浏览器。
3. 输入 Dashboard 端点 URL。
4. 使用您的用户名和密码登录到仪表板。

1.6. 启用编排服务(HEAT)

您可以在 OpenShift (RHOSO)环境中的 Red Hat OpenStack Services 中启用编排服务(heat)。云用户可以使用编排服务来创建和管理云资源，如存储、网络、实例或应用。

流程

1. 在工作站上打开 **OpenStackControlPlane** 自定义资源(CR)文件 **openstack_control_plane.yaml**。
2. 启用并配置 **heat** 服务：

```
spec:
  ...
  heat:
    apiOverride:
      route: {}
    cnfAPIOverride:
```

```
    route: {}
  enabled: true
  template:
    databaseAccount: heat
    databaseInstance: openstack
    heatAPI:
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
      replicas: 1
      resources: {}
      tls:
        api:
          internal: {}
          public: {}
    heatCfnAPI:
      override: {}
      replicas: 1
      resources: {}
      tls:
        api:
          internal: {}
          public: {}
    heatEngine:
      replicas: 1
      resources: {}
    memcachedInstance: memcached
    passwordSelectors:
      authEncryptionKey: HeatAuthEncryptionKey
      service: HeatPassword
    preserveJobs: false
    rabbitMqClusterName: rabbitmq
    secret: osp-secret
    serviceUser: heat
```

3. 更新 control plane：

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

4. 等待 RHOCP 创建与 **OpenStackControlPlane** CR 相关的资源。运行以下命令来检查状态：

```
$ oc get openstackcontrolplane -n openstack
NAME       STATUS  MESSAGE
openstack-control-plane  Unknown  Setup started
```

当状态为 "Setup complete" 时，会创建 **OpenStackControlPlane** 资源。

提示

将 **-w** 选项附加到 **get** 命令的末尾，以跟踪部署进度。

5. 通过查看 **openstack** 命名空间中的 pod 确认 control plane 已部署：

```
$ oc get pods -n openstack
```

当所有 pod 都已完成或运行时，会部署 control plane。

验证

1. 打开与 **OpenStackClient** pod 的远程 shell 连接：

```
$ oc rsh -n openstack openstackclient
```

2. 确认内部服务端点已注册到每个服务：

```
$ openstack endpoint list -c 'Service Name' -c Interface -c URL --service heat
+--------------+-----------+---------------------------------------------------------------+
| Service Name | Interface | URL                                                           |
+--------------+-----------+---------------------------------------------------------------+
| heat         | internal  | http://heat-internal.openstack.svc:9292                       |
| heat         | public    | http://heat-public-openstack.apps.ostest.test.metalkube.org |
+--------------+-----------+---------------------------------------------------------------+
```

3. 退出 **openstackclient** pod:

```
$ exit
```

1.7. 其他资源

- Kubernetes NMState Operator
- Kubernetes NMState 项目
- 使用 MetalLB 进行负载平衡
- MetalLB 文档
- MetalLB 在第 2 层模式中
- 指定 LB IP 可以从中宣布的网络接口
- 多网络
- 在 OpenShift 中使用 Multus CNI
- macvlan plugin
- Whereabouts IPAM CNI 插件 - 扩展配置
- 关于 IP 地址池的广告

- 动态置备

第 2 章 自定义数据平面

OpenShift (RHOSO)数据平面上的 Red Hat OpenStack Services 由 RHEL 9.4 节点组成。您可以使用 **OpenStackDataPlaneNodeSet** 自定义资源定义(CRD)创建定义节点和数据平面布局的自定义资源(CR)。您可以使用预置备节点，或置备裸机节点作为 data plane 创建和部署过程的一部分。

您可以使用在 *OpenShift 上部署 Red Hat OpenStack Services* 指南中的创建 数据平面 中的步骤在数据平面中添加额外的节点集。

您还可以修改现有 **OpenStackDataPlaneNodeSet** CR，将 Compute 单元添加到数据平面中，并通过创建自定义服务来自定义数据平面。

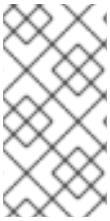
2.1. 先决条件

- RHOSO 环境部署在 Red Hat OpenShift Container Platform (RHOCP)集群中。如需更多信息，请参阅在 OpenShift 上部署 Red Hat OpenStack Services 。
- 以具有 **cluster-admin** 权限的用户身份登录到可访问 RHOCP 集群的工作站。

2.2. MODIFYING AN OPENSTACKDATAPLANENODESET CR

您可以修改现有的 **OpenStackDataPlaneNodeSet** 自定义资源(CR)，以添加新节点或更新节点配置。每个节点只能包含在一个 **OpenStackDataPlaneNodeSet** CR 中。每个节点集只能连接到一个 Compute 单元。默认情况下，节点集连接到 **cell1**。如果您的 control plane 包含额外的 Compute 单元，您必须指定节点集连接的单元。

要将 **OpenStackDataPlaneNodeSet** CR 修改应用到数据平面，您可以创建一个 **OpenStackDataPlaneDeployment** CR 来部署修改后的 **OpenStackDataPlaneNodeSet** CR。



注意

当 **OpenStackDataPlaneDeployment** 成功执行后，它不会自动执行 Ansible，即使 **OpenStackDataPlaneDeployment** 或相关的 **OpenStackDataPlaneNodeSet** 资源已更改。要启动另一个 Ansible 执行，您必须创建另一个 **OpenStackDataPlaneDeployment** CR。

流程

1. 为您要更新的节点集打开 **OpenStackDataPlaneNodeSet** CR 定义文件，如 **openstack_data_plane.yaml**。
2. 更新或添加您需要的配置。有关可用于配置通用节点属性的属性的信息，请参阅 *Deploying Red Hat OpenStack Services on OpenShift 指南中的**OpenStackDataPlaneNodeSet** CR spec 属性*。
3. 保存 **OpenStackDataPlaneNodeSet** CR 定义文件。
4. 应用更新的 **OpenStackDataPlaneNodeSet** CR 配置：

```
$ oc apply -f openstack_data_plane.yaml
```

5. 通过确认状态为 **SetupReady** 来验证 data plane 资源是否已更新：

```
$ oc wait openstackdataplanenodeset openstack-data-plane --for condition=SetupReady --timeout=10m
```

当状态为 **SetupReady** 时，命令会返回一个 **condition met** 信息，否则会返回超时错误。

如需有关 data plane 条件和状态的信息，请参阅 在 *OpenShift 上部署 Red Hat OpenStack Services 中的 Data plane 条件 和状态*。

6. 在工作站上创建一个文件来定义 **OpenStackDataPlaneDeployment** CR：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: <node_set_deployment_name>
```

提示

为定义文件和 **OpenStackDataPlaneDeployment** CR 指定一个唯一的描述性名称，指示修改的节点集的目的。

7. 添加您修改的 **OpenStackDataPlaneNodeSet** CR：

```
spec:
  nodeSets:
    - <nodeSet_name>
```

8. 保存 **OpenStackDataPlaneDeployment** CR 部署文件。
9. 部署修改后的 **OpenStackDataPlaneNodeSet** CR：

```
$ oc create -f openstack_data_plane_deploy.yaml -n openstack
```

您可以在部署执行时查看 Ansible 日志：

```
$ oc get pod -l app=openstackansibleee -w
$ oc logs -l app=openstackansibleee -f --max-log-requests 10
```

如果 **oc logs** 命令返回类似以下错误的错误，请提高 **--max-log-requests** 值：

```
error: you are attempting to follow 19 log streams, but maximum allowed concurrency is 10,
use --max-log-requests to increase the limit
```

10. 验证修改后的 **OpenStackDataPlaneNodeSet** CR 是否已部署：

```
$ oc get openstackdataplanedeployment -n openstack
NAME                  STATUS   MESSAGE
openstack-data-plane   True     Setup Complete


$ oc get openstackdataplanenodeset -n openstack
NAME                  STATUS   MESSAGE
openstack-data-plane   True     NodeSet Ready
```

有关返回状态的含义的信息，请参阅 在 *OpenShift 上部署 Red Hat OpenStack Services* 指南中的 Data plane 条件和状态。

如果状态表示 data plane 尚未部署，则对部署进行故障排除。如需更多信息，*请参阅在 OpenShift 上部署 Red Hat OpenStack Services* 指南中的对 data plane 创建和部署进行故障排除。

11. 如果您将新节点添加到节点集中，请将节点映射到它所连接的 Compute 单元：

```
$ oc rsh nova-cell0-conductor-0 nova-manage cell_v2 discover_hosts --verbose
```

如果您没有创建额外的单元，这个命令会将 Compute 节点映射到 **cell1**。

访问 **openstackclient** pod 的远程 shell，并验证部署的 Compute 节点是否在 control plane 上可见：

```
$ oc rsh -n openstack openstackclient
$ openstack hypervisor list
```

2.3. 数据平面服务

data plane 服务是一个 Ansible 执行，用于管理在 data plane 节点上部署的软件部署的安装、配置和执行。每个服务都是 **OpenStackDataPlaneService** 自定义资源定义(CRD)的资源实例，它将组合了来自 **ConfigMap** 和 **Secret** CR 的 Ansible 内容和配置数据。您可以使用 Ansible play 内容指定服务的 Ansible 执行，这可以是来自 **edpm-ansible** 或任何 Ansible play 内容的 Ansible playbook。 **ConfigMap** 和 **Secret** CR 可以包含需要由 Ansible 内容消耗的任何配置数据。

OpenStack Operator 提供了在 data plane 节点上默认部署的核心服务。如果省略 **OpenStackDataPlaneNodeSet** 规格中的 **services** 字段，则按以下顺序应用以下服务：

```
services:
  - bootstrap
  - download-cache
  - configure-network
  - validate-network
  - install-os
  - configure-os
  - ssh-known-hosts
  - run-os
  - reboot-os
  - install-certs
  - ovn
  - neutron-metadata
  - libvirt
  - nova
  - telemetry
```

OpenStack Operator 还包括以下服务，这些服务没有默认启用：

service	描述

service	描述

ceph-client	包括此服务，将 data plane 节点配置为 Red Hat Ceph Storage 服务器的客户端。包括 **install-os** 和 **configure-os** 服务。**OpenStackDataPlaneNodeSet** CR 必须包括以下配置来访问 Red Hat Ceph Storage secret：
	`apiVersion: dataplane.openstack.org/v1beta1`
	`kind: OpenStackDataPlaneNodeSet`
	`spec:`
	`  ...`
	`  nodeTemplate:`
	`  extraMounts:`
	`  - extraVolType: Ceph`
	`  volumes:`
	`  - name: ceph`
	`  secret:`
	`  secretName: ceph-conf-files`
	`  mounts:`
	`  - name: ceph`
	`  mountPath: "/etc/ceph"`
	`  readOnly: true`

ceph-hci-pre	包含此服务，以准备数据平面节点，以便在 HCI 配置中托管 Red Hat Ceph Storage。如需更多信息，请参阅 *部署超融合基础架构环境*。
neutron-dhcp	包括此服务以在 data plane 节点上运行 Neutron DHCP 代理。
neutron-metadata	包括此服务，以便在 data plane 节点上运行 Neutron OVN Metadata 代理。需要此代理来为 Compute 节点提供元数据服务。
neutron-ovn	包括此服务以在 data plane 节点上运行 Neutron OVN 代理。需要此代理来为 Compute 节点上的硬件卸载端口提供 QoS。
neutron-sriov	包括此服务以在 data plane 节点上运行 Neutron SR-IOV NIC 代理。

有关可用默认服务的详情，请参考 https://github.com/openstack-k8s-operators/openstack-operator/tree/main/config/services。

您可以为 **OpenStackDataPlaneNodeSet** 资源启用和禁用服务。

注意

不要更改默认服务部署的顺序。

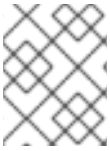
您可以使用 **OpenStackDataPlaneService** CRD 创建可在 data plane 节点上部署的自定义服务。您可以将自定义服务添加到必须执行该服务的默认服务列表中。如需更多信息，请参阅创建和启用自定义服务。

您可以通过查看资源的 YAML 表示来查看服务的详情：

```
$ oc get openstackdataplaneservice configure-network -o yaml -n openstack
```

2.3.1. 创建并启用自定义服务

您可以使用 **OpenStackDataPlaneService** CRD 创建自定义服务，以便在 data plane 节点上部署。



注意

不要创建名称与默认服务相同的自定义服务。如果自定义服务名称与默认服务名称匹配，则默认服务值会覆盖 **OpenStackDataPlaneNodeSet** 协调期间的自定义服务值。

您可以使用 Ansible playbook 或通过直接在服务的 **playbookContents** 部分中包含自由格式 playbook 内容来指定服务的 Ansible 执行。

注意

您不能在同一服务中包含 Ansible **playbook** 和 **playbookContents**。

流程

1. 创建 **OpenStackDataPlaneService** CR，并将它保存到工作站上的 YAML 文件中，如 **custom-service.yaml** ：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
```

2. 通过引用 Ansible playbook 或将 Ansible play 包含在 **playbookContents** 字段中来指定用于创建自定义服务的 Ansible 命令：

   - 指定要使用的 Ansible playbook：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  playbook: osp.edpm.configure_os
```

   - 将 **playbookContents** 字段中的 Ansible play 指定为使用 Ansible playbook 语法的字符串：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  playbookContents: |
    - hosts: all
      tasks:
```

```
    - name: Hello World!
      shell: "echo Hello World!"
      register: output
    - name: Show output
      debug:
        msg: "{{ output.stdout }}"
    - name: Hello World role
      import_role: hello_world
```

有关如何创建 Ansible playbook 的详情，请参考 创建 playbook。

3. 可选： 要使用自定义服务的额外 Ansible 内容覆盖 **ansible-runner** 执行环境使用的默认容器镜像，构建并包含自定义 **ansible-runner** 镜像。如需更多信息，请参阅 构建自定义 **ansible-runner** 镜像。

4. 可选：指定要用于将 secret 或配置传递给 **OpenStackAnsibleEE** 作业的 **Secret** 或 **ConfigMap** 资源的名称：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  ...
  playbookContents: |
    ...
  dataSources:
    - configMapRef:
        name: hello-world-cm-0
    - secretRef:
        name: hello-world-secret-0
    - secretRef:
        name: hello-world-secret-1
        optional: true (1)
```

(1) 可选：将 **可选字段** 设置为 "true"，将资源标记为可选，以便在不存在时不会抛出错误。

为 **OpenStackAnsibleEE** pod 中的每个 **Secret** 和 **ConfigMap** CR 创建一个挂载，其文件名与 resource 值匹配。挂载在 **/var/lib/openstack/configs/<service name>** 下创建。然后，您可以使用 Ansible 内容来访问配置或机密数据。

5. 可选：如果服务必须在 **OpenStackDataPlaneDeployment** CR 中的所有节点集中运行，则将 **deployOnAllNodeSets** 字段设置为 true，即使该服务没有在部署中设置的每个节点中被列为服务：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:

  playbookContents: |
  ...
  deployOnAllNodeSets: true
```

6. 可选：指定服务的 **edpmServiceType** 字段。不同的自定义服务可能会使用相同的 Ansible 内容来管理相同的数据平面服务，如 **ovn** 或 **nova**。您必须在同一位置挂载 DataSources、TLS 证书和 CA 证书，以便 Ansible 内容可在使用自定义服务时找到它们。您可以使用 **edpmServiceType** 字段来创建此关联。该值是默认服务的名称，它使用与自定义服务相同的 Ansible 内容。例如，使用来自 **edpm-ansible** 的 **edpm_ovn** Ansible 内容的自定义服务会将 **edpmServiceType** 设置为 **ovn**，它将与 OpenStack Operator 提供的默认 **ovn** 服务名称匹配。

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  ...
  edpmServiceType: ovn
```

注意

fieldnames 中使用的 acroynm **edpm** 代表 "External Data Plane Management"。

7. 创建自定义服务：

```
$ oc apply -f custom-service.yaml -n openstack
```

8. 验证是否已创建自定义服务：

```
$ oc get openstackdataplaneservice <custom_service_name> -o yaml -n openstack
```

9. 将自定义服务添加到节点设置的定义文件中的 **services** 字段中。按照相对于其他服务执行的顺序添加服务名称。如果 **deployAllNodeSets** 字段设为 **true**，则您只需要将该服务添加到部署中的一个节点集中。

注意

将自定义服务添加到节点设置定义中的服务列表中时，必须包含所有所需的服务，包括默认服务。如果您在服务列表中仅包含您的自定义服务，则这是部署的唯一服务。

2.3.2. 构建自定义 ansible-runner 镜像

当自定义服务需要额外的 Ansible 内容时，您可以使用自己的自定义镜像覆盖 **ansible-runner** 执行环境使用的默认容器镜像。

流程

1. 创建一个将自定义内容添加到默认镜像的 **Containerfile** ：

```
FROM quay.io/openstack-k8s-operators/openstack-ansibleee-runner:latest
COPY my_custom_role /usr/share/ansible/roles/my_custom_role
```

2. 构建镜像并将其推送到容器 registry：

```
$ podman build -t quay.io/example_user/my_custom_image:latest .
$ podman push quay.io/example_user/my_custom_role:latest
```

3. 将新容器镜像指定为 **ansible-runner** 执行环境必须用来添加自定义服务所需的额外 Ansible 内容的镜像，如 Ansible 角色或模块：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  label: dataplane-deployment-custom-service
  openStackAnsibleEERunnerImage: quay.io/openstack-k8s-operators/openstack-ansibleee-runner:latest (1)
  playbookContents: |
```

(1) **ansible-runner** 执行环境用来执行 Ansible 的容器镜像。

2.4. 为功能或工作负载配置节点集

您可以为特定功能或工作负载指定节点集。要为功能或工作负载指定并配置节点集，请完成以下任务：

1. 创建 **ConfigMap** 自定义资源(CR)来为该功能配置节点。
2. 为运行 playbook 的节点集合创建自定义服务。
3. 在自定义服务中包含 **ConfigMap** CR。

注意

Compute 服务(nova)提供名为 **nova-extra-config** 的默认 **ConfigMap** CR，您可以在其中添加适用于使用默认 **nova** 服务的所有节点集的通用配置。如果您使用此默认 **nova-extra-config ConfigMap** 添加要应用到所有节点集的通用配置，则不需要创建自定义服务。

流程

1. 创建为该功能定义新配置文件的 **ConfigMap** CR：

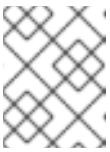
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: feature-configmap
  namespace: openstack
data:
  <integer>-<feature>.conf: |
    <[config_grouping]>
    <config_option> = <value>
    <config_option> = <value>
```

- 将 **<integer>** 替换为指示何时应用配置的数字。control plane 服务以字典顺序在服务目录中应用每个文件 **/etc/<service>/<service>.conf.d/**。因此，之后文件中定义的配置会覆盖之前文件中定义的相同配置。每个服务 Operator 都会生成默认配置文件，其名称为 **01-<service>.conf**。例如，**nova-operator** 的默认配置文件是 **01-nova.conf**。

注意

以下 25 个数字是为 OpenStack 服务和 Ansible 配置文件保留的。

- 将 **<feature** > 替换为表示要配置功能的字符串。



注意

不要使用默认配置文件的名称，因为它将覆盖基础架构配置，如 **transport_url**。

- 将 **<[config_grouping**]> 替换为服务配置文件中的配置选项的名称。例如，**[compute]** 或 **数据库**。
- 将 **<config_option** > 替换为您要配置的选项，如 **cpu_shared_set**。
- 将 **&lt**;value> 替换为配置选项的值，如 **2,6**。
  部署服务时，它会将配置添加到服务容器中的 **etc/<service>/<service>.conf.d/** 目录中。例如，对于计算功能，配置文件将添加到 **nova_compute** 容器中的 **etc/nova/nova.conf.d/** 中。

  有关创建 **ConfigMap** 对象的更多信息，请参阅 RHOCP *节点* 指南中的 创建和使用配置映射。

提示

如果配置包含敏感信息，如认证所需的密码或证书，则可以使用 **Secret** 来创建自定义配置。

2. 为节点集合创建自定义服务。有关如何创建自定义服务的详情，请参考 创建并启用自定义服务。
3. 将 **ConfigMap** CR 添加到自定义服务中：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: <nodeset>-service
spec:
  ...
  dataSources:
    - configMapRef:
        name: feature-configmap
```

4. 为运行此服务的节点集的单元指定 **Secret** CR：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: <nodeset>-service
spec:
  ...
  dataSources:
    - configMapRef:
        name: feature-configmap
    - secretRef:
```

```
      name: nova-migration-ssh-key
  - secretRef:
      name: nova-cell1-compute-config
```

2.5. 将 OPENSTACKDATAPLANENODESET CR 连接到计算单元

每个节点集只能连接到一个 Compute 单元。默认情况下，节点集连接到 **cell1**。如果您在 control plane 中添加了额外的 Compute 单元，您必须指定节点集连接的单元。

流程

1. 创建一个自定义 **nova** 服务，其中包含要连接的单元的 **Secret** CR：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-cell-custom
  spec:
    playbook: osp.edpm.nova
    ...
    dataSources:
    - secretRef:
        name: nova-cell2-compute-config (1)
    edpmServiceType: nova (2)
```

(1) 由 control plane 为单元生成的 **Secret** CR。

(2) 将 **OpenStackDataPlaneService** CR 与 **nova** 服务关联。

有关如何创建自定义服务的详情，请参考 创建并启用自定义服务。

2. 将 **OpenStackDataPlaneNodeSet** CR 中的 **nova** 服务替换为您的自定义 **nova** 服务：

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-cell2
spec:
  services:
    - download-cache
    - bootstrap
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - ssh-known-hosts
    - run-os
    - ovn
    - libvirt
    - *nova-cell-custom*
    - telemetry
```

注意

不要更改默认服务的顺序。

3. 完成 **OpenStackDataPlaneNodeSet** CR 的配置。如需更多信息，请参阅创建数据平面。
4. 保存 **OpenStackDataPlaneNodeSet** CR 定义文件。
5. 创建 data plane 资源：

```
$ oc create -f openstack_data_plane.yaml
```

6. 通过确认状态为 **SetupReady** 来验证 data plane 资源是否已创建：

```
$ oc wait openstackdataplanenodeset openstack-cell2 --for condition=SetupReady --
timeout=10m
```

当状态为 **SetupReady** 时，命令会返回一个 **condition met** 信息，否则会返回超时错误。

如需有关 data plane 条件和状态的信息，请参阅 在 *OpenShift 上部署 Red Hat OpenStack Services 中的 Data plane 条件 和状态*。

7. 验证是否为节点集合创建了 **Secret** 资源：

```
$ oc get secret | grep openstack-cell2
dataplanenodeset-openstack-cell2 Opaque 1 3m50s
```

8. 验证是否已创建服务：

```
$ oc get openstackdataplaneservice -n openstack | grep nova-cell-custom
```

9. 创建 **OpenStackDataPlaneDeployment** CR 以部署 **OpenStackDataPlaneNodeSet** CR。如需更多信息，*请参阅在 OpenShift 上部署 Red Hat OpenStack Services 指南中的部署数据平面*。

第 3 章 在 OPENSHIFT OBSERVABILITY 中自定义 RED HAT OPENSTACK SERVICES

使用 OpenShift 上的 Red Hat OpenStack Services (RHOSO)的可观察性，深入了解您的部署的指标、日志和警报。

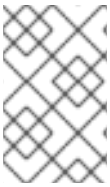
RHOSO 中的可观察性架构由 OpenShift 中的服务组成，以及在您的 Compute 节点上公开指标、日志和警报的服务。您可以使用 OpenShift observability 生态系统深入了解 RHOSO 环境。另外，您还可以访问日志基础架构，以便收集、存储和搜索日志。RHOSO 服务（如 ceilometer 和 sg-core）从计算节点发出指标，以及相关的虚拟基础架构可供 OpenShift Observability 框架使用。

3.1. 在 OPENSHIFT OBSERVABILITY 中配置 RED HAT OPENSTACK SERVICES

在 OpenShift (RHOSO)部署的 Red Hat OpenStack Services (RHOSO)部署中默认启用 Telemetry 服务(ceilometer、prometheus)。您可以通过编辑 **openstack_control_plane.yaml** CR 文件来配置可观察性。

先决条件

- 可选：如果计划启用日志记录，Cluster Logging Operator 会从 OperatorHub 安装。
  - LokiStack 实例必须正在运行。如需更多信息，请参阅配置 LokiStack 日志存储。
  - ClusterLogging 实例必须正在运行。如需更多信息，请参阅 配置日志记录收集器。
  - 必须启用 syslog 接收器。如需更多信息，请参阅使用 syslog 协议 转发日志。



注意

您不需要这些 Operator 以 Prometheus 格式公开和查询 OpenStack 指标。如果不禁用 ceilometer，则会通过以下 URL 从集群内部创建并公开 Prometheus 指标导出器 ：http://ceilometer-internal.openstack.svc:3000/metrics

流程

1. 在工作站上打开 **OpenStackControlPlane** CR 定义文件 **openstack_control_plane.yaml**。
2. 根据您的环境需求更新 **telemetry** 部分：

```
telemetry:
  enabled: true
  template:
    metricStorage:
      enabled: true
      dashboardsEnabled: true
      monitoringStack:
        alertingEnabled: true
        scrapeInterval: 30s (1)
        storage:
          strategy: persistent
          retention: 24h (2)
          persistent:
```

```
        pvcStorageRequest: 20G 3
    autoscaling:
      enabled: false
      aodh:
        databaseAccount: aodh
        databaseInstance: openstack
        secret: osp-secret
      heatInstance: heat
    ceilometer:
      enabled: true
      secret: osp-secret
    logging:
      enabled: false
      ipaddr: 172.17.0.80 4
      annotations:
        metallb.universe.tf/address-pool: internalapi
        metallb.universe.tf/allow-shared-ip: internalapi
        metallb.universe.tf/loadBalancerIPs: 172.17.0.80
```

1. 使用 **scrapeInterval** 字段控制收集新指标前经过的时间长度。更改此参数可能会影响性能。
2. 使用 **retention** 字段调整遥测指标存储的时间长度。此字段会影响所需的存储量。
3. 使用 **pvcStorageRequest** 字段更改要为 Prometheus 时间序列数据库分配的存储量。
4. 使用 **ipaddr** 字段设置 **rsyslog** 发送消息的 IP 地址。确保可以从 Compute 节点访问 IP 地址。默认 IP 地址是 **internalapi** 的 vIP，即 172.17.0.80。确保 **ipaddr** 和 **loadBalancerIP** 具有相同的 IP 地址，以便客户端和服务器可以进行通信。

3. 更新 control plane：

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

4. 等待 RHOCP 创建与 **OpenStackControlPlane** CR 相关的资源。运行以下命令来检查状态：

```
$ oc get openstackcontrolplane -n openstack
NAME       STATUS  MESSAGE
openstack-control-plane  Unknown  Setup started
```

当状态为 "Setup complete" 时，会创建 OpenStackControlPlane 资源。

提示

将 **-w** 选项附加到 **get** 命令的末尾，以跟踪部署进度。

5. 可选：通过查看 **openstack** 命名空间中的 pod 来查看每个单元，确认部署了 control plane：

```
$ oc get pods -n openstack
```

当所有 pod 都已完成或运行时，会部署 control plane。

验证

1. 从您的工作站访问 **OpenStackClient** pod 的远程 shell：

```
$ oc rsh -n openstack openstackclient
```

2. 确认您可以查询 **prometheus**，并且提取端点处于活跃状态：

```
$ openstack metric query up --disable-rbac -c container -c instance  -c value
```

输出示例：

```
+----------------+----------------------+-------+
| container      | instance             | value |
+----------------+----------------------+-------+
| alertmanager   | 10.217.1.112:9093    | 1     |
| prometheus     | 10.217.1.63:9090     | 0     |
| proxy-httpd    | 10.217.1.52:3000     | 1     |
|                | 192.168.122.100:9100 | 1     |
|                | 192.168.122.101:9100 | 1     |
+----------------+----------------------+-------+
```

注意

当集群中调度活跃工作负载时，value 字段中的每个条目都应该为 "1"，但 **prometheus** 容器除外。因为 TLS，**prometheus** 容器会报告一个 "0" 的值，它会被默认启用。

3. 您可以通过点 RHOCP 控制台中的 **Observe** 及 **Dashboards** 来查找 RHOSO 遥测仪表板。

其他资源

- 安装日志记录