Red Hat OpenStack Services on OpenShift 18.0

验证部署的云并对其进行故障排除

在 OpenShift 环境中验证部署的 Red Hat OpenStack Services 并进行故障排除

Last Updated: 2024-08-31

Red Hat OpenStack Services on OpenShift 18.0 验证部署的云并对其进行故障排除

在 OpenShift 环境中验证部署的 Red Hat OpenStack Services 并进行故障排除

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

了解如何在 OpenShift 云上验证和故障排除部署的 Red Hat OpenStack 服务。

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 为支持收集诊断信息

使用 OpenShift (RHOSO) **must-gather** 工具上的 Red Hat OpenStack Services 来收集有关 Red Hat OpenShift Container Platform (RHOCP)集群的诊断信息，包括 RHOSO control plane 和部署的 RHOSO 服务。使用 RHOCP **sosreport** 工具收集有关 RHOSO 数据平面的诊断信息。

1.1. 收集 RHOSO CONTROL PLANE 中的数据

您可以使用 OpenShift (RHOSO) **must-gather** 工具上的 Red Hat OpenStack Services 来收集有关 Red Hat OpenShift Container Platform (RHOCP)集群的以下信息，以排除服务故障：

- RHOSO control plane 服务日志。
- RHOSO control plane 服务的配置，如 RHOCP **Secret** 和 **ConfigMap**。
- 在 RHOSO control plane 中部署的服务的状态。
- RHOSO 生成的自定义资源定义(CRD)。
- RHOSO control plane 应用自定义资源(CR)。
- **openstack** 和 **openstack-operators** 命名空间。
- 与 RHOSO 命名空间相关的 RHOCP 事件。

先决条件

- 使用具有 **cluster-admin** 权限的用户访问集群。

流程

1. 进入存储 **must-gather** 数据的目录。
2. 将一个或多个镜像或镜像流传递给 **must-gather** 工具，以指定要收集的数据。例如，使用以下命令可收集默认集群数据和特定于部署的 RHOSO control plane 的信息：

```
$ oc adm must-gather \
 --image-stream=openshift/must-gather \ 1
 --image=registry.redhat.io/rhoso-operators/openstack-must-gather-rhel9:1.0 2
```

1. 用于收集 RHOCP 集群信息的默认 RHOCP **must-gather** 镜像。
2. RHOSO **must-gather** 镜像。

此命令创建一个本地目录，用于存储日志、服务配置和 RHOSO control plane 服务的状态。

1.2. 收集 RHOSO 数据平面节点上的数据

data plane 节点是运行计算服务(nova)的 RHEL 节点，以及 Ceph 守护进程在 HCI 环境中运行的位置。您可以使用 **must-gather** 工具来收集有关 OpenShift (RHOSO)部署上的 Red Hat OpenStack Services 的诊断信息，如 Red Hat OpenShift Container Platform (RHOCP)和 RHOSO 数据平面和 control plane 节点相关的特定信息。您可以为红帽支持提供 SOS 报告，以帮助诊断并排除部署中的问题。

有关如何使用 SOS 报告工具的详情，请参考 从您的支持体验中获得最大限制。

第 2 章 使用 TEST OPERATOR 运行 TEMPEST 测试

使用 Tempest 测试时，请使用 OpenShift (RHOSO) **test-operator** 上的 Red Hat OpenStack Services。

先决条件

- 已部署 RHOSO 环境。
- 确保您位于 OpenStack 项目中：

```
$ oc project openstack
Now using project "openstack" on server "https://api.crc.testing:6443".
```

2.1. TEMPEST 自定义资源配置文件

此 **test-v1beta1-tempest.yaml** 文件是一个 Tempest 自定义资源(CR)示例，您可以编辑并使用 **test-operator** 执行 Tempest 测试。

```
apiVersion: test.openstack.org/v1beta1
kind: Tempest
metadata:
  name: tempest-tests
  namespace: openstack
spec:
  containerImage: ""
  # storageClass: local-storage
  # parallel: false
  # debug: false

  # configOverwrite
  # ---------------
  # An interface to overwrite default config files like e.g. logging.conf But can also
  # be used to add additional files. Those get added to the service config dir in
  # /etc/test_operator/<file>
  #
  # configOverwrite:
  #   file.txt: |
  #     content of the file

  # SSHKeySecretName
  # ----------------
  # SSHKeySecretName is the name of the k8s secret that contains an ssh key. The key is
  # mounted to ~/.ssh/id_ecdsa in the tempest pod. Note, the test-operator looks for
  # the private key in ssh-privatekey field of the secret.
  #
  # SSHKeySecretName: secret_name
  tempestRun:
    # NOTE: All parameters have default values (use only when you want to override
    #       the default behaviour)
    includeList: | # <-- Use | to preserve \n
      tempest.api.identity.v3.*
    concurrency: 8
    # excludeList: | # <-- Use | to preserve \n
    #   tempest.api.identity.v3.*
```

```
    # workerFile: | # <-- Use | to preserve \n
    #   - worker:
    #     - tempest.api.*
    #     - neutron_tempest_tests
    #   - worker:
    #     - tempest.scenario.*
    # smoke: false
    # serial: false
    # parallel: true
    # externalPlugin:
    #   - repository: "https://opendev.org/openstack/barbican-tempest-plugin.git"
    #   - repository: "https://opendev.org/openstack/neutron-tempest-plugin.git"
    #     changeRepository: "https://review.opendev.org/openstack/neutron-tempest-plugin"
    #     changeRefspec: "refs/changes/97/896397/2"
    # extraImages:
    #   - URL: https://download.cirros-cloud.net/0.6.2/cirros-0.6.2-x86_64-disk.img
    #     name: cirros-0.6.2-test-operator
    #       flavor:
    #         name: cirros-0.6.2-test-operator-flavor
    #         RAM: 512
    #         disk: 20
    #         vcpus: 1

    # extraRPMs:
    # ----------
    # A list of URLs that point to RPMs that should be installed before
    # the execution of tempest. WARNING! This parameter has no efect when used
    # in combination with externalPlugin parameter.
    # extraRPMs:
    #   - https://cbs.centos.org/kojifiles/packages/python-sshtunnel/0.4.0/12.el9s/noarch/python3-
sshtunnel-0.4.0-12.el9s.noarch.rpm
    #   - https://cbs.centos.org/kojifiles/packages/python-whitebox-tests-
tempest/0.0.3/0.1.766ff04git.el9s/noarch/python3-whitebox-tests-tempest-0.0.3-
0.1.766ff04git.el9s.noarch.rpm

  tempestconfRun:
    # NOTE: All parameters have default values (use only when you want to override
    #       the default behaviour)
    # create: true
    # collectTiming: false
    # insecure: false
    # noDefaultDeployer: false
    # debug: false
    # verbose: false
    # nonAdmin: false
    # retryImage: false
    # convertToRaw: false
    # out: ./etc/tempest.conf
    # flavorMinMem: 128
    # flavorMinDisk: 1
    # timeout: 600
    # imageDiskFormat: qcow2
    # image: https://download.cirros-cloud.net/0.5.2/cirros-0.5.2-x86_64-disk.img

    # The following text will be mounted to the tempest pod
    # as /etc/test_operator/deployer_input.yaml
```

```
  # deployerInput: |
  #   [section]
  #   value1 = exmaple_value2
  #   value2 = example_value2

  # The following text will be mounted to the tempest pod
  # as /etc/test_operator/accounts.yaml
  # testAccounts: |
  #  - username: 'multi_role_user'
  #    tenant_name: 'test_tenant_42'
  #    password: 'test_password'
  #     roles:
  #       - 'fun_role'
  #       - 'not_an_admin'
  #       - 'an_admin'

  # The following text will be mounted to the tempest pod
  # as /etc/test_operator/profile.yaml
  # profile: |
  #   collect_timing: false
  #   create: false
  #   create_accounts_file: null

  # createAccountsFile: /path/to/accounts.yaml
  # generateProfile: /path/to/profile.yaml
  # networkID:
  # append: |  # <-- Use | to preserve \n
  #   section1.name1 value1
  #   section1.name1 value2
  # remove: |  # <-- Use | to preserve \n
  #   section1.name1 value1
  #   section1.name1 value2
  # overrides: | # <-- Use | to preserve \n
  #  overrides_section1.name1 value1
  #  overrides_section1.name1 value2

# Workflow
# --------
# Workflow section can be utilized to spawn multiple test pods at the same time.
# The commented out example spawns two test pods that are executed sequentially.
# Each step inherits all configuration that is specified outside of the workflow
# field. For each step you can overwrite values specified in the tempestRun and
# tempestconfRun sections.
#
# workflow:
#   - stepName: firstStep
#     tempestRun:
#       includeList: |
#         tempest.api.*
#   - stepName: secondStep
#     tempestRun:
#       includeList: |
#         neutron_tempest_plugin.*
```

2.2. 安装 TEST OPERATOR

在 **openstack-operators** 项目中安装 **test-operator**。

流程

1. 创建 **subscription.yaml** 文件：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: test-operator
  namespace: openstack-operators
spec:
  name: test-operator
  source: openstack-operator-index
  sourceNamespace: openstack-operators
```

2. 应用 **subscription.yaml** 文件：

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

验证

- 当 **test-operator-controller-manager** pod 成功生成且 pod 运行时，您可以使用 **test-operator** 接受的自定义资源(CR)与 Operator 通信：

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

2.3. 运行 TEMPEST 测试

选择并运行要用于 Tempest 测试的镜像。以下文件名是示例，可能会因您的环境而异。

流程

1. 编辑 Tempest 测试配置文件，例如使用 **vim** ：

```
$ vim <Tempest_config>
```

   - 将 **<Tempest_config** > 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

2. 为 **containerImage** 参数添加适当的值：

```
registry.redhat.io/rhoso/openstack-tempest-all-rhel9:18.0
```

   - 本例中的 **openstack-tempest-all:current-podified** 镜像包含所有默认支持的插件。

3. 保存并关闭 Tempest 测试配置文件。

4. 创建 pod 并运行 Tempest 测试：

```
$ oc apply -f <Tempest_config>
```

- 将 **<Tempest_config>** 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

验证

- 检查 pod 是否正在运行：

```
$ oc get pods | grep -i <pod_name>
```

- 将 **<pod_name>** 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2.4. 查找 TEMPEST 日志

您可以访问 Tempest 日志，例如测试成功完成，或者对失败的 pod 进行故障排除。

流程

1. 获取相关 pod 的名称和状态：

```
$ oc get pods | grep -i <pod_name>
```

   - 将 **<pod_name>** 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2. 获取日志：

```
$ oc logs <pod_name>
```

   - 将 **<pod_name>** 替换为您在上一步中获取的 pod 的名称。

验证

- 查看日志。

2.5. 从 POD 内部获取日志

例如，您可以访问 Tempest 日志，如成功完成测试，或者对失败的 pod 进行故障排除。您可以从 pod 内部访问特定及更详细的 Tempest 日志。

流程

1. 获取相关 pod 的名称和状态：

```
$ oc get pods | grep -i <pod_name>
```

   - 将 **<pod_name>** 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2. 访问 pod：

```
$ oc debug <pod_name>
```

- 将 < **pod_name** > 替换为您在上一步中获取的 pod 的名称。

3. 查看 pod 中的可用日志文件：

```
sh-5.1$ ls -lah /var/lib/tempest/external_files
```

4. 查看所需目录中的可用日志文件：

```
sh-5.1$ ls -lah /var/lib/tempest/external_files/<tempest-tests>
```

- 将 < **tempest-tests** > 替换为您要查看日志的相关目录的名称，如 **tempest-tests**。

验证

- 查看日志。

2.6. 重新运行 TEMPEST 测试

修改 Tempest 配置文件并重新运行 Tempest 测试。

流程

1. 获取相关 pod 的名称和状态：

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

注意

如果 pod 仍然活跃，您可以等待测试完成，然后继续下一步。

2. 获取 Tempest 自定义资源(CR)的名称：

```
$ oc get tempest
```

3. 删除 Tempest CR：

```
$ oc delete tempest <tempest_cr>
```

- 将 < **tempest_cr** > 替换为您在上一步中获取的 Tempest CR 的名称。

4. 验证您是否删除了 pod：

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

5. 编辑 Tempest 测试配置文件，例如使用 **vim** ：

```
$ vim <Tempest_config>
```

- 将 **<Tempest_config** > 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

6. 对 Tempest 测试配置文件进行所需的编辑，例如您可以修改 **excludeList:** 参数：

```
excludeList: | # <-- Use | to preserve \n
<excludeList_value>
```

- 使用您要测试的 **excludeList** 值替换 **<excludeList_value>**，如 **tempest.api.identity.v3 Remediation**。

7. 保存并关闭 Tempest 测试配置文件。

8. 为 Tempest 测试创建 pod：

```
$ oc apply -f <Tempest_config>
```

- 将 **<Tempest_config** > 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

验证

- 获取您在上一步中创建的 pod 的名称：

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

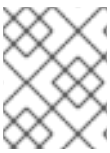
- 在日志中检查预期的更改：

```
$ oc logs <pod_name> | grep  <excludeList_value> --context=4
```

- 将 < **pod_name** > 替换为 **<excludeList_value>** 您在上一步中获取的相关 pod 的名称，并使用添加到 Tempest 测试配置文件中的 **excludeList** 值替换，如 **tempest.api.identity.v3 aws**。

2.7. 安装外部插件

您可以安装外部插件，如 **barbican-tempest-plugin**。



注意

barbican-tempest-plugin 包含在镜像 **registry.redhat.io/rhoso/openstack-tempest-all-rhel9:18.0** 中，作为示例所示。如果您使用不支持的外部插件，请确保谨慎操作。

流程

1. 编辑 Tempest 测试配置文件，例如使用 **vim** ：

```
$ vim <Tempest_config>
```

- 将 **<Tempest_config** > 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

2. 添加 **externalPlugin** 选项，或者取消 Tempest 测试配置文件中的相关行的注释：

```
externalPlugin:
  - repository: "https://opendev.org/openstack/barbican-tempest-plugin.git"
```

3. 保存并关闭 Tempest 测试配置文件。
4. 为 Tempest 测试创建新 pod：

```
$ oc apply -f <Tempest_config>
```

- 将 **<Tempest_config** > 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

验证

- 获取您在上一步中创建的 pod 的名称和状态：

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2.8. 修复处于待处理状态的 POD

您可以使用以下步骤修复因为缺少可用持久性卷导致的处于 **Pending** 状态的 pod。

流程

1. 获取相关 pod 的名称，并验证其状态是否为 **Pending** ：

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2. 确认缺少可用持久性卷导致 **Pending** 状态：

```
$ oc describe pod <pod_name>
```

- 将 < **pod_name** > 替换为您在上一步中获取的 pod 的名称。

3. 列出与 Tempest 关联的所有持久性卷：

```
$ oc get pv | grep -i tempest
```

4. 编辑其中一个持久性卷，将 claim 引用值改为 **null** ：

```
$ oc patch pv <name_of_volume> -p '{"spec":{"claimRef":null}}'
```

- 将 < **name_of_volume** > 替换为您在上一步中获取的 Tempest 卷的名称。

验证

- 确认您编辑的卷已从 **Released** 改为 **Bound** ：

```
$ oc get pv | grep -i tempest
```

- 确认 pod 的状态已从 **Pending**:

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2.9. 使用 DEBUG 模式

使用 debug 模式，如果测试完成或出现故障时，您可以保持 pod 运行，并使用远程 shell 获取更多信息和详细信息。

流程

1. 编辑 Tempest 测试配置文件，例如使用 **vim** ：

```
$ vim <Tempest_config>
```

   - 将 **<Tempest_config** > 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

2. 将 **debug:** 参数的值改为 **true**，或者在配置文件中添加 **debug: true** 行：

```
apiVersion: test.openstack.org/v1beta1
kind: Tempest
metadata:
  name: tempest-tests
  namespace: openstack
spec:
  containerImage: registry.redhat.io/rhoso/openstack-tempest-all-rhel9:18.0
  debug: true
```

3. 保存并关闭 Tempest 测试配置文件。

4. 为 Tempest 测试创建新 pod：

```
$ oc apply -f <Tempest_config>
```

   - 将 **<Tempest_config** > 替换为 Tempest 测试配置文件的名称，如 **test_v1beta1_tempest.yaml**。

验证

1. 获取您在上一步中创建的 pod 的名称：

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2. 远程访问 pod：

```
$ oc rsh <pod_name>
```

- 将 < **pod_name** > 替换为您在上一步中获取的 pod 的名称。

3. 更改或检查正在运行的 pod 中的错误：

```
$ sh-5.1$ ls –lah /var/lib/tempest
```

2.10. 使用 PUDB 调试 TEMPEST 测试

您可以使用 **pudb** 创建和自定义断点，供您用于调试 Tempest 测试。

先决条件

- 您已配置了 debug 模式。有关 debug 模式的更多信息，请参阅 第 2.9 节 “使用 debug 模式”。

流程

1. 获取您要在其中使用 **pudb** 的 pod 的名称：

```
$ oc get pods | grep -i <pod_name>
```

- 将 < **pod_name** > 替换为您在 Tempest 自定义资源配置文件中指定的名称，如 **tempest-tests**，或者只能使用 **$ oc get pods** 并搜索相关的 pod。

2. 远程访问 pod：

```
$ oc rsh <pod_name>
```

- 将 < **pod_name** > 替换为您在上一步中获取的 pod 的名称。

3. 进入正确的目录：

```
sh-5.1$ cd /var/lib/tempest/openshift
```

4. 创建 Python3 轻量级虚拟环境：

```
sh-5.1$ python3 -m venv --system-site-packages .venv
```

5. 激活 Python3 轻量级虚拟环境：

```
sh-5.1$ . .venv/bin/activate
```

6. 在 Python3 轻量级虚拟环境中下载并安装 **pudb** ：

```
(.venv) sh-5.1$ pip install pudb
```

7. 找到您要调试的文件的路径，如 **test_networks.py** ：

```
(.venv) sh-5.1$ find / -name test_networks.py 2> /dev/null
```

8. 打开您选择的文件进行编辑：

```
(.venv) sh-5.1$ sudo vi /usr/lib/python3.9/site-
packages/tempest/api/network/test_networks.py
```

9. 将行 **导入 pudb; pu.db** 插入到要创建 **pudb** 断点的文件，然后保存并关闭该文件。

10. 更改所有权：

```
(.venv) sh-5.1 $ sudo chown -R tempest:tempest /var/lib/tempest/.config
```

11. 使用 **pudb** 断点运行测试：

```
(.venv) sh-5.1 $ python -m testtools.run
tempest.api.network.test_networks.NetworksTest.test_list_networks
```

验证

- **pudb** 接口将打开。在测试完成前，您可以与 **pudb** 接口交互。