Red Hat OpenStack Services on OpenShift 18.0

Validating and troubleshooting the deployed cloud

Validating and troubleshooting a deployed Red Hat OpenStack Services on OpenShift environment

Last Updated: 2024-08-29

Red Hat OpenStack Services on OpenShift 18.0 Validating and troubleshooting the deployed cloud

Validating and troubleshooting a deployed Red Hat OpenStack Services on OpenShift environment

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Learn how to validate and troubleshoot the deployed Red Hat OpenStack Services on OpenShift cloud.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation for Red Hat OpenStack Services on OpenShift (RHOSO) or earlier releases of Red Hat OpenStack Platform (RHOSP). When you create an issue for RHOSO or RHOSP documents, the issue is recorded in the RHOSO Jira project, where you can track the progress of your feedback.

To complete the Create Issue form, ensure that you are logged in to Jira. If you do not have a Red Hat Jira account, you can create an account at https://issues.redhat.com.

1. Click the following link to open a **Create Issue** page: Create Issue
2. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
3. Click **Create**.

CHAPTER 1. COLLECTING DIAGNOSTIC INFORMATION FOR SUPPORT

Use the Red Hat OpenStack Services on OpenShift (RHOSO) **must-gather** tool to collect diagnostic information about your Red Hat OpenShift Container Platform (RHOCP) cluster, including the RHOSO control plane and the deployed RHOSO services. Use the RHOCP **sosreport** tool to collect diagnostic information about your RHOSO data plane.

1.1. COLLECTING DATA ON THE RHOSO CONTROL PLANE

You can use the Red Hat OpenStack Services on OpenShift (RHOSO) **must-gather** tool to collect the following information about your Red Hat OpenShift Container Platform (RHOCP) cluster to troubleshoot service failures:

- The RHOSO control plane service logs.
- The configuration of RHOSO control plane services, such as the RHOCP **Secrets** and **ConfigMaps**.
- Status of the services that are deployed in the RHOSO control plane.
- The RHOSO generated Custom Resource Definitions (CRDs).
- The RHOSO control plane applied Custom Resources (CRs).
- The **openstack** and **openstack-operators** namespaces.
- RHOCP Events that are related to the RHOSO namespaces.

Prerequisites

- Access to the cluster as a user with **cluster-admin** privileges.

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.
2. Pass one or more images or image streams to the **must-gather** tool to specify the data to collect. For example, the following command gathers both the default cluster data and the information that is specific to the deployed RHOSO control plane:

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

1. The default RHOCP **must-gather** image that is used to gather RHOCP cluster information.
2. The RHOSO **must-gather** image.

This command creates a local directory that stores the logs, services configuration, and the status of the RHOSO control plane services.

1.2. COLLECTING DATA ON THE RHOSO DATA PLANE NODES

The data plane nodes are RHEL nodes where the Compute service (nova) runs, and where the Ceph daemons run in an HCI environment. You can use the **must-gather** tool to collect diagnostic information about your Red Hat OpenStack Services on OpenShift (RHOSO) deployment, such as SOS reports for Red Hat OpenShift Container Platform (RHOCP) and RHOSO data plane and control plane nodes, and specific information related to a particular deployed service. You can provide SOS reports to Red Hat Support to help diagnose and troubleshoot issues in your deployment.

For information on how to use the SOS report tool, see Getting the most from your Support experience .

CHAPTER 2. RUNNING TEMPEST TESTS USING TEST OPERATOR

Use the Red Hat OpenStack Services on OpenShift (RHOSO) **test-operator** when working with Tempest tests.

Prerequisites

- A deployed RHOSO environment.
- Ensure that you are in the OpenStack project:

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

2.1. TEMPEST CUSTOM RESOURCES CONFIGURATION FILE

This **test-v1beta1-tempest.yaml** file is an example of a Tempest custom resource (CR) that you can edit and use to execute Tempest tests with **test-operator**.

```
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. INSTALLING TEST OPERATOR

Install the **test-operator** in the **openstack-operators** project.

Procedure

1. Create a **subscription.yaml** file:

```
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. Apply the **subscription.yaml** file:

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

Verification

- When the **test-operator-controller-manager** pod successfully spawns and the pod is running, you can communicate with the operator using the custom resources (CRs) that the **test-operator** accepts:

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

2.3. RUNNING TEMPEST TESTS

Select and run an image to use for Tempest tests. The following file names are examples and might vary from your environment.

Procedure

1. Edit the Tempest test configuration file, for example with **vim**:

```
$ vim <Tempest_config>
```

   - Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

2. Add the appropriate value for the **containerImage** parameter:

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

   - The **openstack-tempest-all:current-podified** image in this example contains all the default supported plug-ins.

3. Save and close the Tempest test configuration file.

4. Create the pod and run your Tempest tests:

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

   - Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

Verification

- Check that the pod is running:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2.4. FINDING TEMPEST LOGS

You can access the Tempest logs, for example for a test that successfully completed, or to troubleshoot a pod that has failed.

Procedure

1. Get the name and status of the relevant pod:

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

   - Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2. Get the logs:

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

   - Replace **<pod_name>** with the name of the pod that you got in the previous step.

Verification

- View the logs.

2.5. GETTING LOGS FROM INSIDE THE POD

You can access the Tempest logs, for example, for a test that successfully completed, or to troubleshoot a pod that has failed. You can access specific and more detailed Tempest logs from inside the pod.

Procedure

1. Get the name and status of the relevant pod:

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

-

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2. Access the pod:

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

- Replace **<pod_name>** with the name of the pod that you got in the previous step.

3. View available log files inside the pod:

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

4. View available log files in the required directory:

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

- Replace **<tempest-tests>** with the name of the relevant directory that you want to view logs in, for example **tempest-tests**.

Verification

- View the logs.

2.6. RE-RUNNING TEMPEST TESTS

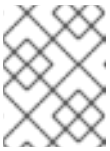
Modify the Tempest configuration file and re-run the Tempest tests.

Procedure

1. Get the name and status of the relevant pod:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.



NOTE

If the pod is still active, you can wait for the test to complete before proceeding to the following step.

2. Get the name of the Tempest custom resource (CR):

```
$ oc get tempest
```

3. Delete the Tempest CR:

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

- Replace **<tempest_cr>** with the name of the Tempest CR that you got in the previous step.

4. Verify that you deleted the pod:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

5. Edit the Tempest test configuration file, for example with **vim**:

```
$ vim <Tempest_config>
```

- Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

6. Make the required edits to the Tempest test configuration file, for example you can modify the **excludeList:** parameter:

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

- Replace **<excludeList_value>** with the **excludeList** value that you want to test, such as **tempest.api.identity.v3.***.

7. Save and close the Tempest test configuration file.

8. Create the pod for your Tempest tests:

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

- Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

Verification

- Get the name of the pod that you created in the previous step:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

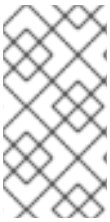
- Check in the logs for the expected changes:

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

- Replace **<pod_name>** with the name of the relevant pod that you got in the previous step and replace **<excludeList_value>** with the **excludeList** value that you added to the Tempest test configuration file, such as **tempest.api.identity.v3.***.

2.7. INSTALLING EXTERNAL PLUG-INS

You can install external plug-ins, such as **barbican-tempest-plugin**.



NOTE

The **barbican-tempest-plugin** is included with the image **registry.redhat.io/rhoso/openstack-tempest-all-rhel9:18.0** and is shown in the following procedure as an example. If you are using external plug-ins that are unsupported, ensure that you proceed with caution.

Procedure

1. Edit the Tempest test configuration file, for example with **vim**:

```
$ vim <Tempest_config>
```

   - Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

2. Add **externalPlugin** option, or uncomment the relevant lines in your Tempest test configuration file:

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

3. Save and close the Tempest test configuration file.

4. Create the new pod for the Tempest tests:

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

   - Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

Verification

- Get the name and status of the pod that you created in the previous step:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2.8. FIXING POD IN PENDING STATE

You can use the following procedure to fix a pod that is in a **Pending** state caused by a lack of available persistent volumes.

Procedure

1. Get the name of the relevant pod and verify it has a status of **Pending**:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2. Confirm that the **Pending** status is caused by a lack of available persistent volumes:

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

- Replace **<pod_name>** with the name of the pod that you got in the previous step.

3. List all persistent volumes that are associated with Tempest:

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

4. Edit one of the persistent volumes to change the claim reference value to **null**:

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

- Replace **<name_of_volume>** with the name of one of the Tempest volume that you got from the previous step.

Verification

- Confirm that the volume that you edited has changed from **Released** to **Bound**:

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

- Confirm that status of the pod has changed from **Pending**:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2.9. USING DEBUG MODE

With debug mode, you can keep the pod running if the test finishes or in case of a failure, and use a remote shell to get more information and detail.

Procedure

1. Edit the Tempest test configuration file, for example with **vim**:

```
$ vim <Tempest_config>
```

- Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

2. Change the value of **debug:** parameter to **true**, or add the line **debug: true** to the configuration file:

```
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. Save and close the Tempest test configuration file.
4. Create the new pod for the Tempest tests:

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

- Replace **<Tempest_config>** with the name of your Tempest test configuration file, such as **test_v1beta1_tempest.yaml**.

Verification

1. Get the name of the pod that you created in the previous step:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2. Access the pod remotely:

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

- Replace **<pod_name>** with the name of the pod that you got in the previous step.

3. Make changes or check errors in the running pod:

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

2.10. USING PUDB TO DEBUG TEMPEST TESTS

You can use **pudb** to create and customize breakpoints that you can use to debug your Tempest tests.

Prerequisites

- You have configured debug mode. For more information about debug mode, see Section 2.9, “Using debug mode”.

Procedure

1. Get the name of the pod that you want to use **pudb** with:

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

- Replace **<pod_name>** with the name that you specified in your Tempest Custom Resources configuration file, for example **tempest-tests**, or you can just use **$ oc get pods** and search for the relevant pod.

2. Access the pod remotely:

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

- Replace **<pod_name>** with the name of the pod that you got in the previous step.

3. Navigate to the correct directory:

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

4. Create a Python3 lightweight virtual environment:

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

5. Activate the Python3 lightweight virtual environment:

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

6. Download and install **pudb** in the Python3 lightweight virtual environment:

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

7. Find the path to the file that you want to debug, for example **test_networks.py**:

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

8. Open your chosen file for editing:

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

9. Insert the line **import pudb; pu.db** into the file where you want to create a **pudb** breakpoint, and save and close the file.

10. Change the ownership:

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

11. Run the test with the **pudb** breakpoint:

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

Verification

- The **pudb** interface opens. You can interact with the **pudb** interface before the test completes.