



Red Hat OpenStack Platform 16.0

OpenStack Integration Test Suite ガイド

OpenStack Integration Test Suite の概要

Red Hat OpenStack Platform 16.0 OpenStack Integration Test Suite ガイド

OpenStack Integration Test Suite の概要

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/OpenStack_Integration_Test_Suite_Guide.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本ガイドでは、Red Hat OpenStack Platform 環境で OpenStack Integration Test Suite をインストール、設定、および管理する方法について説明します。

目次

前書き	3
第1章 はじめに	4
第2章 OPENSTACK INTEGRATION TEST SUITE のテスト	5
2.1. シナリオテスト	5
2.2. API テスト	5
第3章 OPENSTACK INTEGRATION TEST SUITE のインストール	6
3.1. DIRECTOR の使用	6
3.2. 手動インストールの準備	6
3.3. OPENSTACK INTEGRATION TEST SUITE パッケージのインストール	7
3.3.1. tempest プラグインパッケージの一覧	7
第4章 OPENSTACK INTEGRATION TEST SUITE の設定	9
4.1. ワークスペースの作成	9
4.2. TEMPEST の手動設定	10
4.2.1. Tempest 拡張機能一覧の手動設定	10
4.2.2. heat_plugin の手動設定	10
4.3. TEMPEST 設定の検証	11
4.4. ログ設定の変更	11
4.5. マイクロバージョンテストの設定	11
第5章 TEMPEST の使用	13
5.1. 利用可能なテストの一覧表示	13
5.2. SMOKE テストの実行	13
5.3. ホワイトリストファイルを使用したテストへの合格	13
5.4. ブラックリストファイルを使用したテストのスキップ	13
5.5. 並行または連続してテストの実行	13
5.6. 特定のテストの実行	14
第6章 コンテナ化された TEMPEST の実行	15
6.1. TEMPEST コンテナの準備	15
6.2. コンテナ内でコンテナ化された TEMPEST の実行	16
6.3. コンテナの外でコンテナ化された TEMPEST の実行	17
第7章 TEMPEST リソースのクリーニング	19
7.1. クリーンアップの実行	19
7.2. ドライランの実行	19
7.3. TEMPEST オブジェクトの削除	19

前書き

本ガイドでは、Red Hat OpenStack Platform 環境で OpenStack Integration Test Suite をインストール、設定、および管理する方法について説明します。

第1章 はじめに

OpenStack は多くの異なるプロジェクトで構成されるため、OpenStack クラスター内のプロジェクトの相互運用性をテストすることが重要です。OpenStack Integration Test Suite (tempest) は、Red Hat OpenStack Platform デプロイメントの統合テストを自動化します。テストを実行すると、クラスターが想定どおりに動作していることになり、特にアップグレード後に、潜在的な問題を早期に警告することもできます。

Integration Test Suite には、OpenStack API 検証とシナリオテストのテスト、および自己検証のユニットテストが含まれています。Integration Test Suite は、OpenStack パブリック API を使用し、テストランナーとして tempest を使用してブラックボックステストを実行します。

第2章 OPENSTACK INTEGRATION TEST SUITE のテスト

OpenStack Integration Test Suite には多くのアプリケーションがあります。これは、OpenStack コアプロジェクトへのコミットのゲートとして動作し、クラウドデプロイメントの負荷を生成するためにストレステストを行い、CLI テストを実行してコマンドラインの応答形式を確認できます。ただし、関係のある機能はシナリオテスト および **API テスト** です。これらのテストは、OpenStack クラウドデプロイメントに対して実行されます。以下のセクションでは、これらの各テストの実装方法を説明します。

2.1. シナリオテスト

シナリオテストは、サービス間の統合ポイントをテストする一般的なエンドユーザーアクションワークフローをシミュレートします。テストフレームワークは、設定を実施し、サービス間の統合をテストしてから、自動的に削除されます。テストに関連するサービスでテストにタグを付け、テストが使用するクライアントライブラリーを明確にします。

シナリオは、以下のようなユースケースに基づいています。

- Image サービスへのイメージのアップロード
- イメージからのインスタンスのデプロイ
- インスタンスへのボリュームの接続
- インスタンスのスナップショットの作成
- インスタンスからのボリュームの切断

2.2. API テスト

API テストは、OpenStack API を検証します。テストは、OpenStack API の OpenStack Integration Test Suite 実装を使用します。有効な JSON と無効な JSON の両方を使用すると、エラーの応答が有効であることを確認できます。テストを個別に実行し、以前のテスト状態に依存する必要はありません。

第3章 OPENSTACK INTEGRATION TEST SUITE のインストール

本項には、Director または手動インストールのいずれかを使用した OpenStack Integration Test Suite のインストールに関する情報が含まれています。

3.1. DIRECTOR の使用

stack ユーザーのホームディレクトリーにある **undercloud.conf** ファイルを編集します。**enable_tempest** パラメーターが **true** に設定されていることを確認します。

```
enable_tempest = true
```

アンダークラウドがすでにインストールされている場合には、**undercloud.conf** ファイルを編集してから **openstack undercloud install** コマンドを実行し、アンダークラウドに追加の設定を追加します。

```
$ openstack undercloud upgrade
```

これで、「[OpenStack Integration Test Suite パッケージのインストール](#)」に記載されているように、**tempest** パッケージとプラグインをインストールする準備が整いました。

3.2. 手動インストールの準備

OpenStack Integration Test Suite を実行するには、最初に必要なパッケージをインストールし、さまざまな OpenStack サービスやその他のテスト動作スイッチの場所を Integration Test Suite に通知する構成ファイルを作成する必要があります。

OpenStack Integration Test Suite をインストールするには、以下のネットワークが Red Hat OpenStack Platform 環境内で利用可能である必要があります。

- Floating IP を提供できる外部ネットワーク
- プライベートネットワーク

これらのネットワークはルーター経由で接続されている必要があります。

プライベートネットワークを作成します。ネットワークデプロイメントに従って、以下のオプションを指定します。

```
$ openstack network create <network_name> --share
$ openstack subnet create <subnet_name> --subnet-range <address/prefix> \
--network <network_name>
$ openstack router create <router_name>
$ openstack router add subnet <router_name> <subnet_name>
```

パブリックネットワークを作成します。ネットワークデプロイメントに従って、以下のオプションを指定します。

```
$ openstack network create <network_name> --external \
--provider-network-type flat
$ openstack subnet create <subnet_name> --subnet-range <address/prefix> \
--gateway <default_gateway> --no-dhcp --network <network_name>
$ openstack router set <router_name> --external-gateway <public_network_name>
```

これで、OpenStack Integration Test Suite を **tempest** 仮想マシン内にインストールおよび設定する準備が整いました。詳細は、「[「OpenStack Integration Test Suite パッケージのインストール」](#)」を参照してください。

3.3. OPENSTACK INTEGRATION TEST SUITE パッケージのインストール

1. OpenStack Integration Test Suite に関連するパッケージをインストールします。

```
$ sudo dnf -y install openstack-tempest
```

このコマンドでは、tempest プラグインはインストールされません。OpenStack のインストールに応じて、プラグインを手動でインストールする必要があります。

2. 環境内の各コンポーネントに適切な tempest プラグインをインストールします。たとえば、keystone、horizon、neutron、cinder、および telemetry プラグインをインストールするには、以下のコマンドを実行します。

```
$ sudo dnf install python3-keystone-tests-tempest python3-horizon-tests-tempest python3-neutron-tests-tempest python3-cinder-tests-tempest python3-telemetry-tests-tempest
```

各 OpenStack コンポーネントの tempest プラグインの一覧については、「[tempest プラグインパッケージの一覧](#)」を参照してください。



注記

openstack-tempest-all パッケージをインストールすることもできます。このパッケージには、tempest プラグインがすべて含まれます。

3.3.1. tempest プラグインパッケージの一覧

以下のコマンドを実行して、tempest テストパッケージの一覧を取得します。

```
$ sudo dnf search $(openstack service list -c Name -f value) 2>/dev/null | grep test | awk '{print $1}'
```

コンポーネント	パッケージ名
barbican	python3-barbican-tests-tempest
cinder	python3-cinder-tests-tempest
designate	python3-designate-tests-tempest
ec2-api	python3-ec2api-tests-tempest
heat	python3-heat-tests-tempest
horizon	python3-horizon-tests-tempest
ironic	python3-ironic-tests-tempest

コンポーネント	パッケージ名
keystone	python3-keystone-tests-tempest
kuryr	python3-kuryr-tests-tempest
manila	python3-manila-tests-tempest
mistral	python3-mistral-tests-tempest
networking-bgpvpn	python3-networking-bgpvpn-tests-tempest
networking-l2gw	python3-networking-l2gw-tests-tempest
neutron	python3-neutron-tests-tempest
nova-join	python3-novajoin-tests-tempest
octavia	python3-octavia-tests-tempest
patrole	python3-patrole-tests-tempest
telemetry	python3-telemetry-tests-tempest
tripleo-common	python3-tripleo-common-tests-tempest
zaqar	python3-zaqar-tests-tempest



注記

python3-telemetry-tests-tempest パッケージには、aodh、panko、gnocchi、および ceilometer テスト用のプラグインが含まれます。**python3-ironic-tests-tempest** パッケージには、ironic および ironic-inspector のプラグインが含まれます。

第4章 OPENSTACK INTEGRATION TEST SUITE の設定

4.1. ワークスペースの作成

1. ターゲットデプロイメントの認証情報を読み込みます。

- ターゲットがアンダークラウドにある場合は、source コマンドでアンダークラウドの認証情報を読み込みます。

```
# source stackrc
```

- ターゲットがオーバークラウドにある場合、source コマンドでオーバークラウドの認証情報を読み込みます。

```
# source overcloudrc
```

2. **tempest**を初期化します。

```
# tempest init mytempest
# cd mytempest
```

このコマンドは、**mytempest** という名前の tempest ワークスペースを作成します。

以下のコマンドを実行して、既存のワークスペースの一覧を表示します。

```
# tempest workspace list
```

3. **etc/tempest.conf** ファイルを生成します。

```
# discover-tempest-config --deployer-input ~/tempest-deployer-input.conf \
--debug --create --network-id <UUID>
```

UUID を外部ネットワークの UUID に置き換えます。

discover-tempest-config は以前は **config_tempest.py** と呼ばれ、同じパラメーターを取得していました。これは、**openstack-tempest** の依存関係としてインストールされる **python-tempestconf** で提供されます。



注記

アンダークラウドの **etc/tempest.conf** ファイルを生成するには、**tempest-deployer-input.conf** ファイルのリージョン名がアンダークラウドデプロイメントの名前と同じであることを確認します。これらの名前が一致しない場合は、**tempest-deployer-input.conf** ファイルのリージョン名を更新して、アンダークラウドのリージョン名と一致するように更新します。

アンダークラウドのリージョン名を検証するには、以下のコマンドを実行します。

```
$ source stackrc
$ openstack region list
```

オーバークラウドのリージョン名を検証するには、以下のコマンドを実行します。

```
$ source overcloudrc
$ openstack region list
```

4.2. TEMPEST の手動設定

discover-tempest-config コマンドは、**tempest.conf** ファイルを自動的に生成します。ただし、**tempest.conf** ファイルが環境の設定に対応していることを確認する必要があります。

4.2.1. Tempest 拡張機能一覧の手動設定

デフォルトの **tempest.conf** ファイルには、各コンポーネントの拡張機能が含まれます。**tempest.conf** ファイルの各コンポーネントの **api_extensions** 属性を検査し、拡張機能の一覧がデプロイメントに対応することを確認します。

デプロイメントで利用可能な拡張機能が **tempest.conf** ファイルの **api_extensions** 属性の拡張機能の一覧と一致しない場合、コンポーネントは tempest テストに失敗します。この失敗を回避するには、デプロイメントで利用可能な拡張機能を特定し、**api_extensions** パラメーターに含める必要があります。デプロイメント内の Network、Compute、Volume、または Identity 拡張機能の一覧を取得するには、以下のコマンドを実行します。

```
$ openstack extension list [--network] [--compute] [--volume] [--identity]
```

4.2.2. heat_plugin の手動設定

デプロイメント設定に従って、手動で heat プラグインを設定します。以下の例は、**heat_plugin** の最小 **tempest.conf** 設定を示しています。

```
[service_available]
heat = True

[heat_plugin]
username = demo
password = ***
project_name = demo
admin_username = admin
admin_password = ****
admin_project_name = admin
auth_url = http://10.0.0.110:5000//v3
auth_version = 3
user_domain_id = default
project_domain_id = default
user_domain_name = Default
project_domain_name = Default
region = regionOne
fixed_network_name = demo_project_network
network_for_ssh = public
floating_network_name = nova
instance_type = m1.nano
minimal_instance_type = m1.micro
image_ref = 7faed41e-a56c-4971-bf48-24e4e23e69a5
minimal_image_ref = 7faed41e-a56c-4971-bf48-24e4e23e69a5
```

`openstack network list` コマンドを使用して、`fixed_network_name`、`network_for_ssh`、および `floating_network_name` のネットワークを特定します。



注記

`tempest.conf` ファイルの `[service_available]` セクションで `heat` を `True` に設定する必要があります。また、`[heat_plugin]` セクションの `username` 属性にあるユーザーは、`member` である必要があります。たとえば、以下のコマンドを実行して `member` ロールを `demo` ユーザーに追加します。

```
$ openstack role add --user demo --project demo member
```

4.3. TEMPEST 設定の検証

現在の `tempest` 設定を検証します。

```
# tempest verify-config -o <output>
```

`output` は、Tempest が更新された設定を書き込む出力ファイルです。これは、元の設定ファイルとは異なります。

4.4. ログ設定の変更

ログファイルのデフォルトの場所は、`tempest` ワークスペース内の `logs` ディレクトリーです。

このディレクトリーを変更するには、`[DEFAULT]` セクションの `tempest.conf` で、`log_dir` を必要なディレクトリーに設定します。

```
[DEFAULT]
log_dir = <directory>
```

`tempest.conf` に独自のロギング構成ファイルを使用している場合は、使用しているファイルの `[DEFAULT]` セクションの下に `log_config_append` を設定します。

```
[DEFAULT]
log_config_append = <file>
```

`log_config_append` 属性を設定すると、Tempest は `log_dir` 属性を含む、`tempest.conf` 内の他のロギング設定をすべて無視します。

4.5. マイクロバージョンテストの設定

OpenStack Integration Test Suite は、API マイクロバージョンをテストする安定したインターフェースを提供します。このセクションは、これらのインターフェースを使用してマイクロバージョンテストを実装する方法を説明します。

まず、ターゲットマイクロバージョンを指定するには、`tempest.conf` 設定ファイルでオプションを設定する必要があります。サポートされているマイクロバージョンが OpenStack クラウドで使用されているマイクロバージョンに対応するように、これらのオプションを設定します。単一の Integration Test Suite 操作で複数のマイクロバージョンテストを実行するターゲットマイクロバージョンの範囲を指定できます。

たとえば、設定ファイルの **[compute]** セクションで、**compute** サービスのマイクロバージョンの範囲を制限するには、**min_microversion** および **max_microversion** パラメーターに値を割り当てます。

```
[compute]
```

```
min_microversion = 2.14
```

```
max_microversion = latest
```


第5章 TEMPEST の使用

本セクションのコマンドを入力して、さまざまなテスト操作を実行します。1つの **tempest run** コマンドで、複数のオプションを組み合わせることもできます。

5.1. 利用可能なテストの一覧表示

--list-tests または **-l** オプションのいずれかで **tempest-run** コマンドを実行し、利用可能な一時テストの一覧を取得します。

```
# tempest run -l
```

5.2. SMOKE テストの実行

smoke テストは、最も重要な機能のみを対象とした予備的なテストの種類です。これらのテストは包括的ではありませんが、smoke テストの実行で問題が特定できれば時間を節約できます。

```
# tempest run --smoke
```

5.3. ホワイトリストファイルを使用したテストへの合格

ホワイトリストファイルは、追加するテストを選択する正規表現が含まれるファイルです。正規表現は改行で区切られます。

ホワイトリストファイルを使用するには、**--whitelist-file** または **-w** オプションのいずれかで **tempest run** コマンドを実行します。

```
# tempest run -w <whitelist_file>
```

5.4. ブラックリストファイルを使用したテストのスキップ

ブラックリストファイルは、除外するテストを選択する正規表現が含まれるファイルです。正規表現は改行で区切られます。

ブラックリストファイルを使用するには、**--blacklist-file** または **-b** のいずれかのオプションを指定して **tempest run** コマンドを実行します。

```
# tempest run -b <blacklist_file>
```

5.5. 並行または連続してテストの実行

テストを順次実行します。

```
# tempest run --serial
```

テストを並行して実行します。並列テストがデフォルトです。

```
# tempest run --parallel
```

--concurrency または **-c** オプションを使用して、テストを並行して実行するときに使用するワーカーの数を指定します。

```
# tempest run --concurrency <workers>
```

デフォルトでは、Integration Test Suite は利用可能な CPU ごとに1つのワーカーを使用します。

5.6. 特定のテストの実行

--regex 正規表現オプションで特定のテストを実行します。正規表現は Python 正規表現である必要があります。

```
# tempest run --regex <regex>
```

たとえば、以下のコマンド例を使用して、**tempest.scenario** で始まる名前を持つすべてのテストを実行します。

```
# tempest run --regex ^tempest.scenario
```

第6章 コンテナ化された TEMPEST の実行

本項では、アンダークラウド上のコンテナからの tempest な実行について説明します。オーバークラウドまたはアンダークラウドに対して tempest を実行できます。コンテナ化された tempest には、コンテナ化されていない tempest と同じリソースが必要です。

6.1. TEMPEST コンテナの準備

tempest コンテナをダウンロードし、設定するには、以下の手順を実行します。

1. `/home/stack` ディレクトリーに移動します。

```
$ cd /home/stack
```

2. tempest コンテナをダウンロードします。

```
$ podman pull registry.redhat.io/rhosp-rhel8/openstack-tempest:16.0
```

このコンテナには、すべての tempest プラグインが含まれます。このコンテナでグローバルに tempest テストを実行するには、プラグインのテストが含まれます。たとえば、**tempest run --regex '(*)'** コマンドを実行すると、tempest はすべてのプラグインテストを実行します。デプロイメントにすべてのプラグインの設定が含まれていない場合には、これらの tempest テストに失敗します。**tempest list-plugins** コマンドを実行して、インストールされているすべてのプラグインを表示します。テストを除外するには、ブラックリストファイルに除外するテストを含める必要があります。詳細は、「[5章 Tempest の使用](#)」を参照してください。

3. ホストマシンとコンテナ間でデータを交換するために使用するディレクトリーを作成します。

```
$ mkdir container_tempest tempest_workspace
```

4. 必要なファイルを **container_tempest** ディレクトリーにコピーします。このディレクトリーはコンテナのファイルソースです。

```
$ cp stackrc overcloudrc tempest-deployer-input.conf container_tempest
```

5. 利用可能なコンテナイメージを一覧表示します。

```
$ podman images
REPOSITORY                                TAG      IMAGE ID      CREATED
SIZE
registry.redhat.io/rhosp-rhel8/openstack-tempest latest   881f7ac24d8f  10 days ago
641 MB
```

6. コマンドエントリーを容易にするエイリアスを作成します。ディレクトリーをマウントする際に、絶対パスを使用していることを確認します。

```
$ alias podman-tempest="podman run -i --privileged=true\
-v "$(pwd)"/container_tempest:/home/stack/container_tempest:z \
-v "$(pwd)"/tempest_workspace:/home/stack/tempest_workspace:z \
registry.redhat.io/rhosp-rhel8/openstack-tempest:16.0 \
/bin/bash"
```

7. コンテナで利用可能な tempest プラグインの一覧を取得するには、以下のコマンドを実行します。

```
$ podman-tempest -c "rpm -qa | grep tempest"
```

6.2. コンテナ内でコンテナ化された TEMPEST の実行

1. コンテナ内で実行できる tempest スクリプトを作成して **tempest.conf** ファイルを生成し、tempest テストを実行します。スクリプトは以下のアクションを実行します。
 - コマンド **set -e** の終了ステータスを設定します。
 - オーバークラウドに対して tempest を実行する場合は、source コマンドで **overcloudrc** ファイルを読み込みます。アンダークラウドに対して tempest を実行する場合は **stackrc** ファイルを読み込みます。
 - **tempest init** を実行して、tempest ワークスペースを作成します。共有ディレクトリーを使用して、ホストからもファイルにアクセスできるようにします。
 - ディレクトリーを **tempest_workspace** に変更します。
 - 後で簡単に使用できるように、TEMPESTCONF 環境変数をエクスポートします。
 - **discover-tempest-config** を実行して **tempest.conf** ファイルを生成します。**discover-tempest-config** コマンドで追加するオプションの詳細は、**discover-tempest-config --help** を実行します。
 - **--out** を **home/stack/tempest_workspace/tempest.conf** に設定して、**tempest.conf** ファイルがホストマシンからアクセスできるようにします。
 - **--deployer-input** を、共有ディレクトリーの **tempest-deployer-input.conf** ファイルをポイントするように設定します。
 - tempest テストを実行します。このサンプルスクリプトは、smoke テスト **tempest run --smoke** を実行します。

```
$ cat <<'EOF'>> /home/stack/container_tempest/tempest_script.sh
set -e
source /home/stack/container_tempest/overcloudrc
tempest init /home/stack/tempest_workspace
pushd /home/stack/tempest_workspace

export TEMPESTCONF="/usr/bin/discover-tempest-config"

$TEMPESTCONF \
  --out /home/stack/tempest_workspace/etc/tempest.conf \
  --deployer-input /home/stack/container_tempest/tempest-deployer-input.conf \
  --debug \
  --create \
  object-storage.reseller_admin ResellerAdmin

tempest run --smoke

EOF
```

tempest.conf ファイルがすでにあり、tempest テストのみを実行する場合は、スクリプトから **TEMPESTCONF** を省略し、これを **container_tempest** ディレクトリーから **tempest_workspace/etc** ディレクトリーにコピーするコマンドに置き換えます。

```
$ cp /home/stack/container_tempest/tempest.conf
/home/stack/tempest_workspace/etc/tempest.conf
```

2. **tempest_script.sh** スクリプトに実行可能権限を設定します。

```
$ chmod +x container_tempest/tempest_script.sh
```

3. 前のステップで作成したエイリアスを使用して、コンテナから tempest スクリプトを実行します。

```
$ podman-tempest -c 'set -e; /home/stack/container_tempest/tempest_script.sh'
```

4. **.stestr** ディレクトリーでテスト結果に関する情報を検査します。

5. tempest テストを再実行する場合は、最初に tempest ワークスペースを削除し、再作成する必要があります。

```
$ sudo rm -rf /home/stack/tempest_workspace
$ mkdir /home/stack/tempest_workspace
```

6.3. コンテナの外部でコンテナ化された TEMPEST の実行

コンテナは **tempest.conf** ファイルを生成または取得して、テストを実行します。これらの操作は、コンテナ外部から実行できます。

1. オーバークラウドに対して tempest を実行する場合は、source コマンドで **overcloudrc** ファイルを読み込みます。アンダークラウドに対して tempest を実行する場合は、**stackrc** ファイルを読み込みます。

```
# source /home/stack/container_tempest/overcloudrc
```

2. **tempest init** を実行して、tempest ワークスペースを作成します。共有ディレクトリーを使用して、ホストからもファイルにアクセスできるようにします。

```
# tempest init /home/stack/tempest_workspace
```

3. **tempest.conf** ファイルを生成します。

```
# discover-tempest-config \
--out /home/stack/tempest_workspace/tempest.conf \
--deployer-input /home/stack/container_tempest/tempest-deployer-input-conf \
--debug \
--create \
object-storage.reseller_admin ResellerAdmin
```

discover-tempest-config コマンドで追加するオプションの詳細は、**discover-tempest-config --help** を実行します。

4. tempest テストを実行します。たとえば、以下のコマンドを実行して、直前の手順で作成したエイリアスを使用して tempest smoke テストを実行します。

```
# podman-tempest -c "tempest run --smoke"
```

5. **.stestr** ディレクトリーでテスト結果に関する情報を検査します。
6. tempest テストを再実行する場合は、最初に tempest ワークスペースを削除し、再作成する必要があります。

```
$ sudo rm -rf /home/stack/tempest_workspace  
$ mkdir /home/stack/tempest_workspace
```

第7章 TEMPEST リソースのクリーニング

tempest を実行した後は、テストプロセスで作成されたファイル、ユーザー、およびプロジェクトが削除する必要のあることとなります。self-clean は、**tempest** の設計原則の1つです。

7.1. クリーンアップの実行

最初に、保存された状態を初期化する必要があります。これにより **saved_state.json** ファイルが作成されます。これにより、クリーンアップは保存する必要のあるオブジェクトを削除できなくなります。通常、**tempest** の実行前に **--init-saved-state** でクリーンアップを実行します。そうでない場合は、クリーンアップを削除するオブジェクトを削除するには、**saved_state.json** を編集する必要があります。

```
# tempest cleanup --init-saved-state
```

クリーンアップを実行します。

```
# tempest cleanup
```

tempest cleanup コマンドは **tempest** リソースを削除しますが、プロジェクトや **tempest** の管理者アカウントは削除しません。

7.2. ドライランの実行

実際のクリーンアップを実行する前にドライランを実行することが推奨されます。ドライランでは、クリーンアップによって削除されるファイルが一覧表示されますが、ファイルは削除されません。ファイルは **dry_run.json** ファイルに一覧表示されます。**dry_run.json** ファイルをチェックして、クリーンアップが環境に必要なファイルを削除していないことを確認します。

```
# tempest cleanup --dry-run
```

7.3. TEMPEST オブジェクトの削除

次のコマンドを実行して、プロジェクトを含むすべての **tempest** リソースを削除しますが、**tempest** 管理者アカウントは削除しません。

```
# tempest cleanup --delete-tempest-conf-objects
```