



Red Hat OpenStack Platform

10

OpenStack Integration Test Suite ガイド

OpenStack Integration Test Suite の概要

OpenStack Team

OpenStack Integration Test Suite の概要

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2017 Red Hat, Inc.

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, 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 Software Collections 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

本ガイドは、Red Hat OpenStack Platform 環境での OpenStack Integration Test Suite のインストール、設定、管理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 のスクリプト	5
第3章 OPENSTACK INTEGRATION TEST SUITE のテスト	6
3.1. シナリオテスト	6
3.2. API テスト	6
3.3. ストレステスト	7
3.4. 単体テスト	8
第4章 OPENSTACK INTEGRATION TEST SUITE のインストール	9
第5章 OPENSTACK INTEGRATION TEST SUITE の設定	10
付録A OPENSTACK INTEGRATION TEST SUITE スクリプトの一覧および説明	11
付録B マイクロバージョンテストの設定および実装	12

前書き

本ガイドは、Red Hat OpenStack Platform 環境での OpenStack Integration Test Suite のインストール、設定、管理について説明します。

第1章 はじめに

OpenStack Integration Test Suite (tempest) は、ライブの OpenStack クラスターに対して実行される統合テストセットです。Integration Test Suite には、一連の OpenStack API の検証テスト、シナリオ、Red Hat OpenStack Platform のデプロイメントに役立つその他の固有テストが含まれます。

OpenStack Integration Test Suite では、**tempest** コードを再利用するためのインターフェースを外部ツールまたはテストスイートに提供します。OpenStack Integration Test Suite のリポジトリの **tempest/lib** にあるパブリックのインターフェースは、安定性のあるパブリックインターフェースとして扱われ、外部でも安全に使用できるはずです。何らかの変更があった場合には、後方互換性が維持されるように、最大限尽力します。ライブラリーは独立しており、ライブラリー以外の OpenStack Integration Test suite の他の機能には依存しません。

Red Hat OpenStack Platform のコンポーネントには、API のマイクロバージョンが実装されています。Integration Test Suite には、API マイクロバージョンのテストをサポートする安定したインターフェースが含まれます。設定オプションや各テストケースの組み合わせで利用できるマイクロバージョンの範囲に基づいて、選択したマイクロバージョンで、API 要求を実行します。

第2章 OPENSTACK INTEGRATION TEST SUITE のスクリプト

OpenStack Integration Test Suite には、インストールおよび設定に利用可能な複数のスクリプトが含まれています。**run_tests.sh** スクリプトを使用すると OpenStack デプロイメントでテストを実行できます。OpenStack Integration Test Suite サービスで利用可能なツールの全一覧は、**/tools** ディレクトリーにあります。これらのスクリプトの概要を以下に示します。オプション **-h** を指定してこれらのスクリプトを実行すると、さらに詳しい説明を表示することができます。

- ※ ***configure-tempest-directory***: Integration Test Suite では、検出するテストが現在の作業ディレクトリーにあることを想定しています。このスクリプトは、現在のディレクトリーにシンボリックリンクおよび他のファイルを追加して、クラウドに対して **tempest** を実行できるようにします。同じマシンから別のクラウドに対して **tempest** を実行するには、このスクリプトを別のディレクトリーで実行してください。
- ※ ***config_tempest.py***: OpenStack Integration Test Suite は設定が難しい場合もあります。さまざまなオプションを設定して、異なるユーザー、テナント、フレーバー、イメージを作成する必要があります。**config_tempest.py** スクリプトにより、設定をより簡単に行うことができます。**--create** オプションを使用すると、必要なリソースと **admin** 認証情報を作成できます。Red Hat は、エンタープライズ用のクラウドに対してこのスクリプトを実行している間には **admin** の認証情報を追加しないことを推奨します。
- ※ ***run-tests.sh***: このスクリプトは、**testr** テストランナーのラッパーで、省略するテストセットを指定したファイルを提供するだけでなく、オプションの **xunit** 出力を生成します。**testr** に指定する long 引数はすべてこのスクリプトで使用することができます。詳しい情報は **testr** ドキュメントを参照してください。

これらのスクリプトに関する詳しい情報は「[OpenStack Integration Test Suite スクリプトの一覧および説明](#)」を参照してください。

第3章 OPENSTACK INTEGRATION TEST SUITE のテスト

OpenStack Integration Test Suite は、多数の異なる環境に役立つように設計されています。たとえば、OpenStack のコアプロジェクトへのコミットをフィルタリングして、OpenStack クラウドの実装が正確であるか、また OpenStack クラウドのツールに問題がないかどうかを検証します。そのため、OpenStack Integration Test Suite テストは多数のフレーバーがあり、それぞれ独自のルールやガイドラインが存在します。テストの種類は以下のとおりです。

- ※ シナリオテスト
- ※ API テスト
- ※ ストレステスト

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が、OpenStack Integration Test Suite では **tempest** の実装の検証のために実行可能な **単体テスト** も提供しています。以下のセクションでは、各テストおよびテストの実装方法を簡単に説明しています。

3.1. シナリオテスト

シナリオテストとは、OpenStack 機能の **スループス** テストです。この種類のテストは通常、複数のサービスを必要とする複雑な状態を設定、実行、終了する一連のステップで構成されます。複数の OpenStack サービスがどのようにに関わりあうかを確認するため、複数の OpenStack サービス間の統合を伴うテストが理想的です。



注記

シナリオテストには、OpenStack の既存の Python クライアントを使用せずに、クライアントの **tempest** 実装を使用するようにしてください。

シナリオテストには、実際のユースケースが含まれているべきです。たとえば、オペレーターが空の環境から開始して、以下の機能を実行するユースケースがあります。

- ※ Image サービスのイメージのアップロード
- ※ そのイメージからのインスタンスのデプロイ
- ※ **ssh** を使用したゲストでのログイン
- ※ インスタンスのスナップショット作成

スコープ: OpenStack プロジェクトの統合テストに使用するシナリオテストは、OpenStack Integration Test Suite の中核的な目的となっています。シナリオテストは常に、OpenStack API の OpenStack Integration Test Suite 実装を使用して、公式なクライアントにバグが潜んでいないことを確認します。テストは、検証するサービスでタグ付けされます。これは、どのクライアントのライブラリーがテストで直接使用されるかにより決定されます。

適切なシナリオテストの例: 複数のサービスの対話をテストする際には、各対話を具体的に表します。「this is my data center」という大規模なスモークテストでは、問題が発生した場合にデバッグが困難です。特に、リソースの設定、変更、デタッチの後に、後ほどそのリソースを再利用する際に対話が繰り返し行われる場合など、冒頭で紹介した Image サービスとコンピュート間の対話のフローが例として適しています。

3.2. API テスト

API テストは、OpenStack API の検証テストです。



注記

API テストは、OpenStack の既存の Python クライアントを使用せずに、クライアントの **tempest** 実装を使用するようにしてください。

これにより、JSON をテストできるようになります。ロークライアントがあるので、無効な JSON を API に渡して結果を確認することができます。これは、ネイティブクライアントではできません。

スコープ: API テストは、OpenStack API の OpenStack Integration Test Suite 実装を常に使用して、公式のクライアントにバグが潜んでいないことを確認する必要があります。これらのテストでは、固有の API 呼び出しをテストし、API 呼び出しに意義を持たせるのに必要な場合には複雑な状況を構築することもできます。良好なデータだけでなく、不正なデータも API に送信して、エラーコードを検索する必要があります。以前のテストで作成された状態だけに依存せず、すべて独立して API テストを実行できる必要があります。

3.3. ストレステスト

ストレステストは、OpenStack 環境にストレスをかけるために設計されており、OpenStack 環境に負荷の高いワークロードを実行してどの部分が壊れてしまうかを確認します。ストレステストのフレームワークは、複数のテストジョブを並列して実行して、OpenStack Integration Test Suite で既存のテストをストレスジョブとして実行できます。

環境: この特定のフレームワークは、稼働中のコンピュートクラスターが **nova** API 2.0 に対応していることを前提としています。ストレスジョブは、クラスターからのログを読み込むことができます。これを有効化するには、**nova-manage** を呼び出すためのホスト名、クラスターに **ssh** 接続するための秘密鍵、ユーザー名を **tempest.conf** の **[stress]** セクションに指定する必要があります。また、ログファイルの場所も指定する必要があります。

```
target_logfiles = "regexp to all log files to be checked for errors"
target_private_key_path
= "private ssh key for controller and log file nodes" target_ssh_user =
"username for
controller and log file nodes" target_controller = "hostname or ip of
controller node (for
nova-manage) log_check_interval = "time between checking logs for
errors (default 60s)"
```

コンソールへのログインを有効にするには、**tempest.conf** で **use_stderr** を有効にするか、デフォルトの **logging.conf.sample** ファイルを使用するようにしてください。

デフォルトのストレステストセットの実行: ストレステストのフレームワークでは、OpenStack Integration Test Suite 内のテストを自動的に検出することができます。**@stresstest** のデコレーターが付いたテストフラグはすべて実行されます。この検出機能を使用するには、OpenStack Integration Test Suite CLI をインストールして、**tempest root** ディレクトリーに移動し、以下を実行してください。

```
# tempest run-stress -a -d 30
```

サンプルテストを実行してインストールをテストします。

```
# tempest run-stress -t tempest/stress/etc/server-create-destroy-  
test.json -d 30
```

このサンプルテストは、複数の仮想マシンを作成して、複数の仮想マシンを中断するように試みます。

テストが終了しない場合や、問題が発生することがあります。そのような場合には、**nova** クラスタを消去してください。tools のサブディレクトリーにスクリプトが提供されています。以下のスクリプトを使用して、キーペア、Floating IP アドレス、サーバーを破棄してください。

```
# tempest/stress/tools/cleanup.py
```

3.4. 単体テスト

単体テストは、OpenStack Integration Test Suite の自己テストで、**tempest** の内部コンポーネントの機能を検証したり、再発したバグがないかを確認したりします。このテストは、OpenStack Integration Test Suite の個別のコンポーネントが想定通りに機能していることを検証する目的のみで使用すべきでは、以下のようにテスト検出パスを指定して実行することができます。

```
# OS_TEST_PATH=./tempest/tests testr run --parallel
```

OS_TEST_PATH を **./tempest/tests** に設定することで、テスト検出が単体テストのディレクトリーでしか実行できないと指定します。**OS_TEST_PATH** のデフォルト値は

OS_TEST_PATH=./tempest/test_discover で、OpenStack Integration Test Suite でのみテスト検出が実行されるようになっています。

または、**venv** を作成して、単体テストを実行する **run_tests.sh** スクリプトを使用します。適切なバージョンの Python で単体テストを実行する **py27** および **py34** の tox ジョブもあります。

スコープ: 単体テストは、OpenStack Integration Test Suite で使用されるメカニズムが有効で機能していることを確認するためのものです。単体テストは、実行に外部サービスや追加の設定は必要ありません。テストに必要な状態がテスト用に作りだされるか、一時的なテストディレクトリーに作成されます (一時テストディレクトリーの使用例は、**test_wrappers.py** を参照してください)。

第4章 OPENSTACK INTEGRATION TEST SUITE のインストール

OpenStack Integration Test Suite は、Red Hat OpenStack Platform director のアンダークラウドで **enable_tempest** を有効にして、director でインストールすることができます。**enable_tempest** パラメーターは、検証ツールをインストールするかどうかを定義します。デフォルトでは、この値は **false** に設定されています。

OpenStack Integration Test Suite を手動でインストールする手順に関する情報は、[「OpenStack Integration Test Suite のインストール」](#)を参照してください。

第5章 OPENSTACK INTEGRATION TEST SUITE の設定

OpenStack Integration Test Suite は、作成する必要があるユーザー、テナント、フレーバー、イメージなどのさまざまなオプションを設定するための `config_tempest.py` スクリプトを提供します。詳しい情報は [config_tempest.py](#) を参照してください。

OpenStack Integration Test Suite を手動で設定してクラウドデプロイメントと連携させる方法に関しては、「[OpenStack Integration Test Suite の設定](#)」を参照してください。

付録A OPENSTACK INTEGRATION TEST SUITE スクリプト の一覧および説明

以下のセクションは、利用可能な **tempest** スクリプトとそのスクリプトの github へのリンクを一覧表示しています。

✳ [configure-tempest-directory](#)

✳ [config_tempest.py](#)

✳ [run_tempest.sh](#)

✳ [run_tests.sh](#)

付録B マイクロバージョンテストの設定および実装

OpenStack Integration Test Suite は、API マイクロバージョンをテストする安定性のあるインターフェースを提供します。以下のセクションは、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の実装方法を説明します。

作業を開始する前に、サポートされるマイクロバージョンが OpenStack クラウドで使用するものと同じになるように、**tempest.conf** の設定ファイルでオプションを設定してテスト対象のマイクロバージョンを指定する必要があります。テスト対象のマイクロバージョンの範囲を指定することで、単一の Integration Test Suite オペレーションで複数のマイクロバージョンテストを操作することができます。

設定ファイルの **[compute]** セクションで、以下のように **min_microversion** および **max_microversion** パラメーターの値を追加します。

```
[compute]
min_microversion = None
max_microversion = latest
```

マイクロバージョンテストの実装

1. テストクラスや設定のマイクロバージョン範囲をベースにテストを省略するロジックを追加します。設定範囲に含まれないテストを自動的にスキップするには、以下のように **api_version_utils.check_skip_with_microversion** 関数を使用してください。

```
class BaseTestCase1(api_version_utils.BaseMicroversionTest):

    [...]
    @classmethod
    def skip_checks(cls):
        super(BaseTestCase1, cls).skip_checks()

        api_version_utils.check_skip_with_microversion(cls.min_microversion,
        cls.max_microversion,
        CONF.compute.min_microversion,
        CONF.compute.max_microversion)
```

Skip ロジックをテストベースクラスに追加してください。追加しないと、固有のテストクラスがテストクラスの構造に依存します。

2. API 要求と合わせて送信する際に使用する適切なマイクロバージョンを選択してください。 **api_version_utils.select_request_microversion** 関数を使用して、適切なマイクロバージョンを選択できます。このマイクロバージョンは API 要求に使用されます。以下に例を示します。

```
@classmethod
def resource_setup(cls):
    super(BaseTestCase1, cls).resource_setup()
    cls.request_microversion = (
        api_version_utils.select_request_microversion(
            cls.min_microversion,
```



```
CONF.compute.min_microversion))
```

3. 選択したマイクロバージョンで API を要求できるように、以前のステップでテストクラスが選択したマイクロバージョンを、サービスクライアントに設定する必要があります。

マイクロバージョンは、サービスクライアント上でグローバル変数として定義することができます。これは、フィクスチャーを使用して設定できます。マイクロバージョンヘッダー名は、サービスクライアントで定義する必要があります。ヘッダー名は、API コントラクトのようにプロジェクト別に変更すべきではないので、一貫性を持たせる必要があります。以下に例を示します。

```
COMPUTE_MICROVERSION = None

class BaseClient1(rest_client.RestClient):
    api_microversion_header_name = 'X-OpenStack-Nova-API-Version'
```

新規のテストクラスは、テストが完了したらリセットできるようにするフィクスチャーを使用して、必要なサービスクライアントに対して、選択したマイクロバージョンを設定できるようになりました。

```
def setUp(self):
    super(BaseTestCase1, self).setUp()

    self.useFixture(api_microversion_fixture.APIMicroversionFixture(
        self.request_microversion))
```

サービスクライアントは、API 要求ヘッダーのマイクロバージョンを追加する必要があります。これは、以下のように、rest_client の get_headers() メソッドを上書きすることで追加できます。

```
COMPUTE_MICROVERSION = None

class BaseClient1(rest_client.RestClient):
    api_microversion_header_name = 'X-OpenStack-Nova-API-Version'

    def get_headers(self):
        headers = super(BaseClient1, self).get_headers()
        if COMPUTE_MICROVERSION:
            headers[self.api_microversion_header_name] =
COMPUTE_MICROVERSION
        return headers
```

4. マイクロバージョンごとにテストクラスを分離します。

マイクロバージョンテストの場合には、別のテストクラスを実装する必要があります。さらに、各テストクラスは min_microversion および max_microversion などのクラス変数で、そのマイクロバージョンを定義します。テストは、定義した範囲で有効です。その範囲が設定したマイクロバージョンの範囲外の場合は、テストは省略されます。



注記

マイクロバージョンテストでは個別のテストケースレベルではなく、テストクラスレベルでのテストがサポートされています。

以下のテストは、2.10 から 2.9 のマイクロバージョンに適用可能です。

```
class BaseTestCase1(api_version_utils.BaseMicroversionTest,
                    tempest.test.BaseTestCase):

    [...]

class Test1(BaseTestCase1):
    min_microversion = '2.2'
    max_microversion = '2.9'

    [...]
```

以下のテストは、2.10 から最新のマイクロバージョンに適用可能です。

```
class Test2(BaseTestCase1):
    min_microversion = '2.10'
    max_microversion = 'latest'

    [...]
```