Red Hat OpenStack Platform 13

Open Virtual Network を使用したネットワーク

OVN を使用した OpenStack のネットワーク

Last Updated: 2023-04-10

Red Hat OpenStack Platform 13 Open Virtual Network を使用したネットワーク

OVN を使用した OpenStack のネットワーク

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

OpenStack のネットワークタスクに OVN を使用するためのガイドです。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、弊社の CTO、Chris Wright のメッセージ を参照してください。

第1章 OPEN VIRTUAL NETWORK (OVN)

Open Virtual Network (OVN) は、インスタンスにネットワークサービスを提供する、Open vSwitch をベースとするソフトウェア定義ネットワーク (SDN) ソリューションです。OVN はプラットフォームに依存しない、OpenStack Networking API の完全なサポートを提供します。OVN により、ゲストインスタンスのグループを L2 または L3 プライベートネットワークにプログラムで接続することができます。OVN は、Red Hat の他のプラットフォームやソリューションを拡張することのできる仮想ネットワークの標準的な方法を採用しています。

Red Hat OpenStack Platform (RHOSP) の本リリースでは、ML2/OVS メカニズムドライバーから ML2/OVN メカニズムドライバーへの移行はサポートされません。RHOSP の本リリースでは、OpenStack コミュニティーの移行ストラテジーはサポートされません。移行サポートは、RHOSP の今後のリリースで予定されています。

注記

OVS の必要な最小バージョンは OVS 2.9 です。

本項では、director を使用した OVN のデプロイに必要なステップを説明します。

注記

OVN は、少なくとも 3 つのコントローラーノードを持ち分散仮想ルーター (DVR) が有効な RHOSP 高可用性 (HA) 環境でのみサポートされます。

1.1. RHOSP OVN アーキテクチャーのコンポーネント一覧

RHOSP OVN アーキテクチャーでは、Networking API をサポートするために OVS Modular Layer 2 (ML2) メカニズムドライバーが OVN ML2 メカニズムドライバーに置き換えられます。OVN は、Red Hat OpenStack Platform のネットワークサービスを提供します。

OVN アーキテクチャーは、以下のコンポーネントとサービスで設定されます。

OVN メカニズムドライバーを使用する ML2 プラグイン

ML2 プラグインは、OpenStack 固有のネットワーク設定を、プラットフォーム非依存の OVN 論理ネットワーク設定に変換します。通常、コントローラーノード上で実行されます。

OVN Northbound (NB) データベース (ovn-nb)

このデータベースは、OVN ML2 プラグインからの論理 OVN ネットワーク設定を保管します。通常コントローラーノードで実行され、TCP ポート **6641** をリッスンします。

OVN Northbound サービス (ovn-northd)

このサービスは OVN NB データベースからの論理ネットワーク設定を論理データパスフローに変換して、それらを OVN Southbound データベースに投入します。通常、コントローラーノード上で実行されます。

OVN Southbound (SB) データベース (ovn-sb)

このデータベースは、変換された論理データパスフローを保管します。通常コントローラーノードで実行され、TCP ポート **6642** をリッスンします。

OVN コントローラー (ovn-controller)

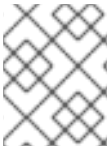
このコントローラーは OVN SB データベースに接続して Open vSwitch コントローラーとして機能し、ネットワークトラフィックの制御とモニターリングを行います。これにより、**OS::Tripleo::Services::OVNController** が定義されているすべてのコンピュートおよびゲートウェイノードで実行されます。

OVN メタデータエージェント (ovn-metadata-agent)

このエージェントは、OVS インターフェイス、ネットワーク名前空間、メタデータ API 要求のプロキシーに使用される HAProxy プロセスを管理するための **haproxy** インスタンスを作成します。このエージェントは、**OS::TripleO::Services::OVNMetadataAgent** が定義されているすべてのコンピュートおよびゲートウェイノードで実行されます。

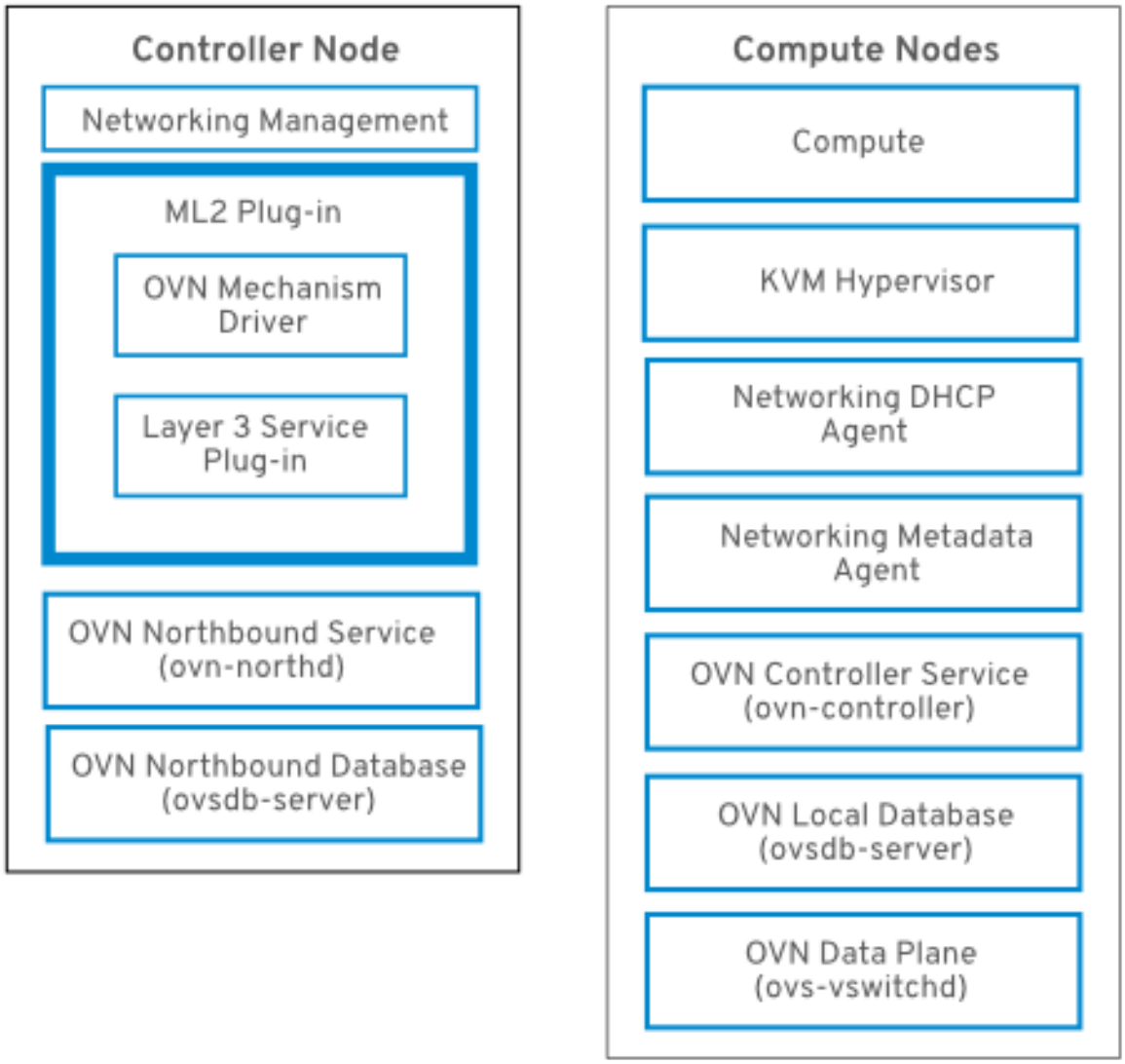
OVS データベースサーバー (OVSDB)

OVN の Northbound および Southbound データベースをホストします。また、**ovs-vswitchd** と連携して OVS データベース **conf.db** をホストします。



注記

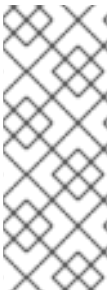
NB データベースのスキーマファイルは **/usr/share/ovn/ovn-nb.ovsschema** にあり、SB データベースのスキーマファイルは **/usr/share/ovn/ovn-sb.ovsschema** にあります。



第2章 OVN デプロイメントのプランニング

OVN は、少なくとも 3 つのコントローラーノードを持つ RHOSP 高可用性 (HA) 環境にのみデプロイします。分散仮想ルーター (DVR) を有効化して OVN をデプロイしてください。

新規の ML2/OVN デプロイメントではデフォルトで DVR が有効化され、新規の ML2/OVS デプロイメントではデフォルトで無効化されています。**neutron-ovn-dvr-ha.yaml** 環境ファイルは、OVN を HA 環境で使用するデプロイメント用の DVR 固有のパラメーターを設定します。



注記

OVN を使用するには、director のデプロイメントで VXLAN ではなく、Generic Network Virtualization Encapsulation (Geneve) を使用する必要があります。Geneve により、OVN は 24 ビットの Virtual Network Identifier (VNI) フィールドと追加の 32 ビットの Type Length Value (TLV) を使用してネットワークを特定し、送信元および宛先の論理ポートの両方を指定できます。MTU 設定を決定する際には、この大きなプロトコルヘッダーについて考慮する必要があります。

2.1. コンピュートノード上の OVN-CONTROLLER サービス

ovn-controller サービスは各コンピュートノードで実行され、OVN Southbound (SB) データベースサーバーに接続して論理フローを取得します。次に **ovn-controller** はその論理フローを OpenFlow の物理フローに変換して、OVS ブリッジ (**br-int**) に追加します。**ovs-vswitchd** と通信して OpenFlow フローをインストールするために、**ovn-controller** は **ovn-controller** の起動時に渡された UNIX ソケットパス (例: **unix:/var/run/openvswitch/db.sock**) を使用して、(**conf.db** をホストする) ローカルの **ovsdb-server** に接続します。

ovn-controller サービスは、**Open_vSwitch** テーブルの **external_ids** コラムに特定のキーと値のペアがあることを想定します。**puppet-ovn** は **puppet-vswitch** を使用して、これらのフィールドにデータを読み込みます。**puppet-vswitch** が **external_ids** コラムに設定するキーと値のペアは以下のとおりです。

```
hostname=<HOST NAME>
ovn-encap-ip=<IP OF THE NODE>
ovn-encap-type=geneve
ovn-remote=tcp:OVN_DBS_VIP:6642
```

2.2. OVN コンポーザブルサービス

通常 Red Hat OpenStack Platform は、事前定義済みロールのノード (Controller ロール、Compute ロール、さまざまなストレージロール種別のノードなど) で設定されます。これらのデフォルトロールには、それぞれコアの heat テンプレートコレクションで定義されるサービスのセットが含まれます。

デフォルトの RHOSP ML2/OVN デプロイメントでは、ML2/OVN コンポーザブルサービスはコントローラーノード上で実行されます。オプションとして、カスタムの Networker ロールを作成し、専用のネットワーカーノードで OVN コンポーザブルサービスを実行することができます。

OVN コンポーザブルサービス **ovn-dbs** は、**ovn-dbs-bundle** というコンテナーにデプロイされます。デフォルトのインストールでは、**ovn-dbs** は Controller ロールに含まれ、コントローラーノードで実行されます。サービスはコンポーザブルなので、Networker ロール等の別のロールに割り当てることができます。

OVN コンポーザブルサービスを別のロールに割り当てる場合には、サービスが Pacemaker サービスと同じノード上に共存し、OVN データベースコンテナーを制御するようにします。

関連情報

- ML2/OVN でのカスタムロールのデプロイ

2.3. ML2/OVN でのカスタムロールのデプロイ

デフォルトの RHOSP ML2/OVN デプロイメントでは、ML2/OVN コンポーザブルサービスはコントローラーノード上で実行されます。オプションとして、専用のネットワーカーノードで OVN コンポーザブルサービスを実行する、Networker 等のサポートされているカスタムロールを使用することができます。

独自のカスタムロールを生成することもできます。

前提条件

- カスタムロールのデプロイ方法を理解している。詳しい情報は、オーバークラウドの高度なカスタマイズ の コンポーザブルサービスとカスタムロール を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインし、source コマンドで **stackrc** ファイルを読み込みます。

```
$ source stackrc
```

2. デプロイメントに適したカスタムロールファイルを選択します。たとえば、Networker ロールの場合は **Networker.yaml** を選択します。そのままでご自分のニーズに適する場合には、直接デプロイコマンドで使用します。あるいは、他のカスタムロールファイルを組み合わせる独自のカスタムロールファイルを生成することもできます。

3. (オプション) これらのカスタムロールファイルの 1 つと他のカスタムロールファイルを組み合わせる新しいカスタムロールデータファイルを生成します。roles_data ファイルの作成 の手順に従います。デプロイメントに応じて、適切なソー出力ルファイルを含めます。

4. (オプション) ロール用の特定のノードを特定するには、特定のハードウェアフレーバーを作成して特定のノードにフレーバーを割り当てることができます。次に、環境ファイルを使用してロールのフレーバーを定義し、ノード数を指定します。詳細については、新規ロールの作成 の例を参照してください。

5. デプロイメントに適した環境ファイルを作成します。たとえば、Networker ロールの場合は、**neutron-ovn-dvr-ha.yaml** という名前のファイルを作成します。

6. デプロイメントに適するように、以下の設定を含めます。たとえば、Networker ロールの場合は、以下の設定を追加します。

```
ControllerParameters:
    OVNCMSOptions: ""
ControllerSriovParameters:
      OVNCMSOptions: ""
NetworkerParameters:
    OVNCMSOptions: "enable-chassis-as-gw"
```

7. オーバークラウドをデプロイします。**-e** オプションを使用して、環境ファイルをデプロイメントコマンドに追加します。-r オプションを使用して、カスタムロールデータファイルをデプロイメントコマンドに追加します。(例: -r Networker.yaml または -r mycustomrolesfile.yaml)。

検証手順

1. ovn_metadata_agent がコントローラーノードおよびネットワーカーノードで実行されていることを確認します。

```
[heat-admin@controller-0 ~]$ sudo docker ps | grep ovn_metadata
```

以下の例のような出力が表示されるはずです。

```
a65125d9588d  undercloud-0.ctlplane.localdomain:8787/rh-osbs/rhosp13-openstack-neutron-metadata-agent-ovn:13.1_20200813.1  kolla_start           23 hours ago  Up 21 hours ago         ovn_metadata_agent
```

2. OVN サービスが設定されたコントローラーノードまたは専用のネットワーカーノードが OVS のゲートウェイとして設定されていることを確認します。

```
[heat-admin@controller-0 ~]$ sudo ovs-vsctl get Open_Vswitch .
...OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None
```

以下の例のような出力が表示されるはずです。

```
external_ids:ovn-cms-options
    enable-chassis-as-gw
```

関連情報

- オーバークラウドの高度なカスタマイズ の コンポーザブルサービスとカスタムロール

2.4. PACEMAKER を使用した高可用性と DVR

ベースプロファイル ovn-dbs-container と Pacemaker 高可用性 (HA) プロファイル ovn-dbs-container-puppet の 2 つの **ovn-dbs** プロファイルのいずれかを選択できます。

Pacemaker HA プロファイルを有効にすると、**ovsdb-server** は Pacemaker およびリソースエージェントの Open Cluster Framework (OCF) スクリプトにより管理される **マスター/スレーブ** モードで実行されます。OVN データベースサーバーは全コントローラーで起動し、**pacemaker** はその中からマスターロールとして機能するコントローラーを 1 つ選択します**マスター** モードで実行する **ovsdb-server** インスタンスはデータベースに書き込みできますが、その他のスレーブの **ovsdb-server** サービスはすべて **マスター** からローカルにデータベースを複製し、データベースに書き込みできません。

このプロファイル用の YAML ファイルは **tripleo-heat-templates/environments/services-docker/neutron-ovn-dvr-ha.yaml** ファイルです。これを有効化すると、OVN データベースサーバーは Pacemaker によって管理され、**puppet-tripleo** は **ovn:ovndb-servers** という名前の pacemaker OCF リソースを作成します。

OVN データベースサーバーは各コントローラーノードで起動し、仮想 IP アドレス (**OVN_DBS_VIP**) を所有するコントローラーは、OVN DB サーバーを **master** モードで実行します。OVN ML2 メカニズムドライバーと **ovn-controller** は次に **OVN_DBS_VIP** 値を使用してデータベースサーバーに接続します。フェイルオーバーが発生した場合には、Pacemaker がこの仮想 IP アドレス (**OVN_DBS_VIP**) を別のコントローラーに移動し、またそのノードで実行されている OVN データベースサーバーを **master** に昇格します。

2.5. OVN でのレイヤー 3 高可用性

OVN は、特別な設定なしでレイヤー 3 の高可用性 (L3 HA) をサポートします。OVN は、指定した外部ネットワークで L3 ゲートウェイとして機能することが可能なすべての利用可能なゲートウェイノードに対して、ルーターポートを自動的にスケジューリングします。OVN L3 HA は OVN **Logical_Router_Port** テーブルの **gateway_chassis** コラムを使用します。大半の機能は、バンドルされた active_passive の出力を使用する OpenFlow ルールによって管理されます。**ovn-controller** は Address Resolution Protocol (ARP) リスポンダーとルーターの有効化/無効化を処理します。FIP 用の Gratuitous ARP およびルーターの外部アドレスも **ovn-controller** によって定期的に送信されます。

注記

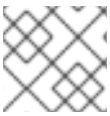
L3HA は OVN を使用してルーターのバランスを取り、元のゲートウェイノードに戻して、ノードがボトルネックとなるのを防ぎます。

BFD モニターリング

OVN は双方向フォワーディング検出 (BFD) プロトコルを使用してゲートウェイノードの可用性をモニターリングします。このプロトコルは、ノード間で確立される Geneve トンネル上でカプセル化されます。

各ゲートウェイノードは、デプロイメント内のスタートポロジーを設定するその他すべてのゲートウェイノードをモニターリングします。ゲートウェイノードは、コンピュートノードもモニターリングして、パケットのルーティングの有効化/無効化および ARP の応答とアナウンスメントを行います。

各コンピュートノードは BFD を使用して、各ゲートウェイノードをモニターリングし、特定のルーターのアクティブなゲートウェイノードを介して送信元および宛先のネットワークアドレス変換 (SNAT および DNAT) などの外部のトラフィックを自動的に誘導します。コンピュートノードは他のコンピュートノードをモニターリングする必要はありません。

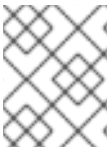


注記

ML2-OVS 設定で検出されるような外部ネットワークのエラーは検出されません。

OVN 向けの L3 HA では、以下の障害モードがサポートされています。

- ゲートウェイノードがネットワーク (トンネリングインターフェイス) から切断された場合。
- **ovs-vswitchd** が停止した場合 (**ovs-switchd** が BFD のシグナリングを行うロールを果たします)。
- **ovn-controller** が停止した場合 (**ovn-controller** は登録済みノードとして、それ自身を削除します)。



注記

この BFD モニターリングメカニズムは、リンクのエラーのみで機能し、ルーティングのエラーには機能しません。

第3章 DIRECTOR を使用した OVN のデプロイ

以下のイベントは、Red Hat OpenStack Platform 上に OVN をデプロイするとトリガーされます。

1. OVN ML2 プラグインを有効化して、必要な設定オプションを生成します。
2. OVN データベースと **ovn-northd** サービスをコントローラーノードにデプロイします。
3. 各コンピュートノードに **ovn-controller** をデプロイします。
4. 各コンピュートノードに **neutron-ovn-metadata-agent** をデプロイします。

3.1. DVR を使用する ML2/OVN のデプロイ

ML2/OVN デプロイメントにおいて分散仮想ルーター (DVR) をデプロイおよび管理するには、heat テンプレートおよび環境ファイルで設定を行います。

注記

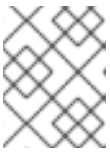
本ガイドの以下の手順では、デフォルトの DVR を使用する OVN を HA 環境でデプロイします。

デフォルト設定はガイドラインとしてのみ提供されます。ネットワークの分離、専用の NIC、またはその他の変動要因のためにカスタマイズが必要となる実稼働環境またはテスト環境で機能することは想定されていません。

以下の手順の例は、典型的なデフォルト値を使用して ML2/OVN、HA、DVR の概念実証用のデプロイメントを設定する方法を示しています。

手順

1. **environments/services/neutron-ovn-dvr-ha.yaml** ファイルの **OS::TripleO::Compute::Net::SoftwareConfig** の値が、使用中の **OS::TripleO::Controller::Net::SoftwareConfig** の値と同じであることを確認します。これは通常、**environments/net-multiple-nics.yaml** ファイルなど、オーバークラウドのデプロイ時に使用するネットワーク環境ファイルで確認することができます。これにより、コンピュートノード上に適切な外部のネットワークブリッジが作成されます。



注記

コンピュートノードのネットワーク設定をカスタマイズする場合には、代わりにカスタムファイルに適切な設定を追加しなければならない場合があります。

2. オーバークラウドのデプロイ時に **environments/services/neutron-ovn-dvr-ha.yaml** を環境ファイルとして含めます。以下に例を示します。

```
$ openstack overcloud deploy \
   --templates /usr/share/openstack-tripleo-heat-templates \
   ...
 -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-dvr-ha.yaml
```

3. **roles_data.yaml** の Compute ロールおよび Controller ロールには、タグ **external_bridge** が含まれ、外部ネットワークエントリーがコンピュートノードに追加されるようにします。以下に例を示します。

```
- name: Compute
  description: |
    Basic Compute Node role
  CountDefault: 1
  # Create external Neutron bridge (unset if using ML2/OVS without DVR)
  tags:
    - external_bridge
  networks:
    External:
      subnet: external_subnet
...
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
    - external_bridge
```

3.2. コンピュートノードでの OVN メタデータエージェントのデプロイ

OVN メタデータエージェントは **tripleo-heat-templates/docker/services/ovn-metadata.yaml** ファイルで設定され、**OS::TripleO::Services::OVNMetadataAgent** でデフォルトのコンピュートロールに含まれます。そのため、デフォルトのパラメーターを使用する OVN メタデータエージェントは、OVN のデプロイメントの一環としてデプロイされます。*3章director を使用した OVN のデプロイ* を参照してください。

OpenStack のゲストインスタンスは、169.254.169.254 のリンクローカル IP アドレスで利用可能なネットワークのメタデータサービスにアクセスします。**neutron-ovn-metadata-agent** は、コンピュートのメタデータ API があるホストネットワークへのアクセスが可能です。各 HAProxy は、適切なホストネットワークに到達できないネットワーク名前空間内にあります。HaProxy は、メタデータ API の要求に必要なヘッダーを追加してから、UNIX ドメインソケット上でその要求を **neutron-ovn-metadata-agent** に転送します。

OVN のネットワークサービスは、メタデータサービスを有効化する各仮想ネットワークに独自のネットワーク名前空間を作成します。コンピュートノード上のインスタンスがアクセスする各ネットワークには、対応するメタデータ名前空間があります (ovnmeta-<net_uuid>)。

3.2.1. メタデータに関する問題のトラブルシューティング

メタデータ名前空間を使用して、コンピュートノード上のローカルインスタンスへのアクセス問題のトラブルシューティングを行うことができます。メタデータ名前空間の問題をトラブルシューティングするには、コンピュートノードで以下のコマンドを root として実行します。

```
# ip netns exec ovnmeta-fd706b96-a591-409e-83be-33caea824114 ssh
USER@INSTANCE_IP_ADDRESS
```

USER@INSTANCE_IP_ADDRESS は、トラブルシューティングするローカルインスタンスのユーザー名と IP アドレスに置き換えます。

3.3. OVN を使用した内部 DNS のデプロイ

East-West トラフィックにローカルネットワークの IP アドレスの代わりにドメイン名を使用する場合は、内部ドメイン名サービス (DNS) を使用します。内部 DNS では、ovn-controller は DNS クエリーにコンピュートノード上でローカルに応答します。内部 DNS は、インスタンスの /etc/resolv.conf ファイルで指定されたカスタム DNS サーバーに優先する点に注意してください。内部 DNS がデプロイされると、インスタンスの DNS クエリーはカスタム DNS サーバーではなく ovn-controller によって処理されます。

手順

1. **NeutronPluginExtensions** パラメーターを使用して DNS を有効にします。

```
parameter_defaults:
  NeutronPluginExtensions: "dns"
```

2. オーバークラウドをデプロイする前に DNS ドメインを設定します。

```
  NeutronDnsDomain: "mydns-example.org"
```

3. オーバークラウドをデプロイします。

```
$ openstack overcloud deploy \
    --templates /usr/share/openstack-tripleo-heat-templates \
    ...
 -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-ovn-dvr-ha.yaml
```

第4章 OVN のモニターリング

OVN 論理フローのモニターリングとトラブルシューティングには、**ovn-trace** コマンドを使用できます。また、OpenFlows のモニターリングとトラブルシューティングには、**ovs-ofctl dump-flows** コマンドを使用できます。

4.1. OVN トラブルシューティングコマンドのエイリアスの作成

OVN データベースコマンド (ovn-nbctl show 等) は ovn_controller コンテナーで実行されます。コンテナーはコントローラーノードとコンピュートノードで実行されます。コマンドへのアクセスを簡素化するには、エイリアスを定義するスクリプトを作成し source コマンドでスクリプトファイルを読み込みます。

前提条件

- メカニズムドライバーとして ML2/OVN を使用する Red Hat OpenStack Platform 13 のデプロイメント

OVN データベースコマンドのエイリアスの作成および使用

1. ovn コマンドを実行するオーバークラウドノードの適切なディレクトリーにシェルスクリプトファイルを作成します。たとえば、heat-admin としてコントローラーノードにログインし、heat-admin ユーザーの ~/bin ディレクトリーに ovn-alias.sh ファイルを作成します。

2. 以下のコマンドをスクリプトファイルに保存します。

```
EXTERNAL_ID=\
$(sudo ovs-vsctl get open . external_ids:ovn-remote | awk -F: '{print $2}')
export NBDB=tcp:${EXTERNAL_ID}:6641
export SBDB=tcp:${EXTERNAL_ID}:6642

alias ovn-sbctl="sudo docker exec ovn_controller ovn-sbctl --db=$SBDB"
alias ovn-nbctl="sudo docker exec ovn_controller ovn-nbctl --db=$NBDB"
alias ovn-trace="sudo docker exec ovn_controller ovn-trace --db=$SBDB"
```

3. source コマンドでスクリプトファイルを読み込みます。たとえば、heat-admin としてコントローラーノードにログインし、以下のコマンドを実行します。

```
# source ovn-alias.sh
```

4. エイリアスを検証します。たとえば、ノースバウンドデータベースを表示します。

```
ovn-nbctl show
```

出力例

```
switch 26ce22db-1795-41bd-b561-9827cbd81778 (neutron-f8e79863-6c58-43d0-8f7d-
8ec4a423e13b) (aka internal_network)
 port 1913c3ae-8475-4b60-a479-df7bcce8d9c8
     addresses: ["fa:16:3e:33:c1:fc 192.168.254.76"]
 port 1aabaee3-b944-4da2-bf0a-573215d3f3d9
     addresses: ["fa:16:3e:16:cb:ce 192.168.254.74"]
```

```
    port 7e000980-59f9-4a0f-b76a-4fdf4e86f27b
        type: localport
        addresses: ["fa:16:3e:c9:30:ed 192.168.254.2"]
```

4.2. OVN の論理フローのモニターリング

OVN は論理フローを使用します。これは、優先度、マッチング、アクションで設定されるフローのテーブルです。これらの論理フローは、各コンピュートノード上で実行される **ovn-controller** に分散されます。以下の例に示したように、コントローラーノード上で **ovn-sbctl lflow-list** コマンドを使用すると、論理フローの完全なセットを表示することができます。

```
$ ovn-sbctl --db=tcp:172.17.1.10:6642 lflow-list
    Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10)  Pipeline: ingress
      table=0 (ls_in_port_sec_l2  ), priority=100  , match=(eth.src[40]), action=(drop;)
      table=0 (ls_in_port_sec_l2  ), priority=100  , match=(vlan.present), action=(drop;)
      table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "sw0-port1" && eth.src ==
{00:00:00:00:00:01}), action=(next;)
      table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport == "sw0-port2" && eth.src ==
{00:00:00:00:00:02}), action=(next;)
      table=1 (ls_in_port_sec_ip  ), priority=0    , match=(1), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port1" && eth.src ==
00:00:00:00:00:01 && arp.sha == 00:00:00:00:00:01), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port1" && eth.src ==
00:00:00:00:00:01 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll == 00:00:00:00:00:01) ||
((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:01)))), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && arp.sha == 00:00:00:00:00:02), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=90   , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll == 00:00:00:00:00:02) ||
((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:02)))), action=(next;)
      table=2 (ls_in_port_sec_nd  ), priority=80   , match=(inport == "sw0-port1" && (arp || nd)), action=
(drop;)
      table=2 (ls_in_port_sec_nd  ), priority=80   , match=(inport == "sw0-port2" && (arp || nd)), action=
(drop;)
      table=2 (ls_in_port_sec_nd  ), priority=0    , match=(1), action=(next;)
      table=3 (ls_in_pre_acl      ), priority=0    , match=(1), action=(next;)
      table=4 (ls_in_pre_lb       ), priority=0    , match=(1), action=(next;)
      table=5 (ls_in_pre_stateful ), priority=100  , match=(reg0[0] == 1), action=(ct_next;)
      table=5 (ls_in_pre_stateful ), priority=0    , match=(1), action=(next;)
      table=6 (ls_in_acl          ), priority=0    , match=(1), action=(next;)
      table=7 (ls_in_qos_mark     ), priority=0    , match=(1), action=(next;)
      table=8 (ls_in_lb           ), priority=0    , match=(1), action=(next;)
      table=9 (ls_in_stateful     ), priority=100  , match=(reg0[1] == 1), action=(ct_commit(ct_label=0/1);
next;)
      table=9 (ls_in_stateful     ), priority=100  , match=(reg0[2] == 1), action=(ct_lb;)
      table=9 (ls_in_stateful     ), priority=0    , match=(1), action=(next;)
      table=10(ls_in_arp_rsp      ), priority=0    , match=(1), action=(next;)
      table=11(ls_in_dhcp_options ), priority=0    , match=(1), action=(next;)
      table=12(ls_in_dhcp_response), priority=0    , match=(1), action=(next;)
      table=13(ls_in_l2_lkup      ), priority=100  , match=(eth.mcast), action=(outport = "_MC_flood";
output;)
      table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:01), action=(outport
= "sw0-port1"; output;)
      table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:02), action=(outport
= "sw0-port2"; output;)
```

```
    Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10)  Pipeline: egress
      table=0 (ls_out_pre_lb      ), priority=0    , match=(1), action=(next;)
      table=1 (ls_out_pre_acl     ), priority=0    , match=(1), action=(next;)
      table=2 (ls_out_pre_stateful), priority=100  , match=(reg0[0] == 1), action=(ct_next;)
      table=2 (ls_out_pre_stateful), priority=0    , match=(1), action=(next;)
      table=3 (ls_out_lb          ), priority=0    , match=(1), action=(next;)
      table=4 (ls_out_acl         ), priority=0    , match=(1), action=(next;)
      table=5 (ls_out_qos_mark    ), priority=0    , match=(1), action=(next;)
      table=6 (ls_out_stateful    ), priority=100  , match=(reg0[1] == 1), action=(ct_commit(ct_label=0/1);
next;)
      table=6 (ls_out_stateful    ), priority=100  , match=(reg0[2] == 1), action=(ct_lb;)
      table=6 (ls_out_stateful    ), priority=0    , match=(1), action=(next;)
      table=7 (ls_out_port_sec_ip ), priority=0    , match=(1), action=(next;)
      table=8 (ls_out_port_sec_l2 ), priority=100  , match=(eth.mcast), action=(output;)
      table=8 (ls_out_port_sec_l2 ), priority=50   , match=(outport == "sw0-port1" && eth.dst ==
{00:00:00:00:00:01}), action=(output;)
      table=8 (ls_out_port_sec_l2 ), priority=50   , match=(outport == "sw0-port2" && eth.dst ==
{00:00:00:00:00:02}), action=(output;)
```

OVN と OpenFlow には、主に以下のような相違点があります。

- OVN ポートは、ネットワーク内にある論理エンティティーで、単一のスイッチ上にある物理ポートではありません。
- OVN により、パイプライン内の各テーブルには番号に加えて名前が付けられます。名前は、パイプライン内のそのステージの目的を示します。
- OVN の match 構文は、複雑なブール表現をサポートしています。
- OVN の論理フローでは、OpenFlow よりも幅広いアクションをサポートしています。OVN の論理フローの構文で DHCP などの高度な機能を実装することができます。

ovn-trace

ovn-trace コマンドを使用して、パケットが OVN の論理フローをどのように通過するかシミュレーションしたり、パケットがドロップする原因を特定するのに役立てたりすることができます。**ovn-trace** コマンドには、以下のパラメーターを指定して実行してください。

DATAPATH
: シミュレーションされるパケットの送信が開始される場所の論理スイッチまたは論理ルーター。

MICROFLOW
: シミュレーションされるパケット。**ovn-sb** データベースで使用される構文で指定します。

この例では、シミュレーションされるパケットに **--minimal** の出力オプションが示されており、そのパケットが宛先に到達したことを表しています。

```
$ ovn-trace --minimal sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 && eth.dst ==
00:00:00:00:00:02'
   # reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type=0x0000
   output("sw0-port2");
```

さらに詳しい情報を表示するには、シミュレーションされる同じパケットの **--summary** 出力に完全な実行パイプラインが表示されます。

```
$ ovn-trace --summary sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 && eth.dst ==
```

```
00:00:00:00:00:02'
# reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type=0x0000
ingress(dp="sw0", inport="sw0-port1") {
    outport = "sw0-port2";
    output;
    egress(dp="sw0", inport="sw0-port1", outport="sw0-port2") {
        output;
        /* output to "sw0-port2", type "" */;
    };
};
```

この出力例には、以下の内容が示されています。

- パケットは **sw0-port1** ポートから **sw0** ネットワークに入り、受信のパイプラインを通過します。
- **outport** 変数が **sw0-port2** に設定されているのは、このパケットの宛先が **sw0-port2** に指定されていることを意味します。
- パケットは受信のパイプラインから出力されます。このパイプラインは、**outport** 変数が **sw0-port2** に設定された **sw0** の送信パイプラインにパケットを送ります。
- 出力のアクションは、送信のパイプラインで実行されます。このパイプラインでは、パケットが **outport** 変数の現在の値である **sw0-port2** に出力されます。

詳しい情報は、**ovn-trace** の man ページを参照してください。

4.3. OPENFLOWS のモニターリング

ovs-ofctl dump-flows コマンドを使用して、ネットワーク内の論理スイッチ上の OpenFlow のフローをモニターリングすることができます。

```
$ ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=72.132s, table=0, n_packets=0, n_bytes=0, idle_age=72,
priority=10,in_port=1,dl_src=00:00:00:00:00:01 actions=resubmit(,1)
 cookie=0x0, duration=60.565s, table=0, n_packets=0, n_bytes=0, idle_age=60,
priority=10,in_port=2,dl_src=00:00:00:00:00:02 actions=resubmit(,1)
 cookie=0x0, duration=28.127s, table=0, n_packets=0, n_bytes=0, idle_age=28, priority=0
actions=drop
 cookie=0x0, duration=13.887s, table=1, n_packets=0, n_bytes=0, idle_age=13, priority=0,in_port=1
actions=output:2
 cookie=0x0, duration=4.023s, table=1, n_packets=0, n_bytes=0, idle_age=4, priority=0,in_port=2
actions=output:1
```