Red Hat OpenStack Platform 16.0

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: 2022-09-13

Red Hat OpenStack Platform 16.0 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

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

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。

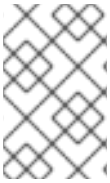
目次

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。

移行サポートの進捗を追跡するには、https://bugzilla.redhat.com/show_bug.cgi?id=1862888 を参照してください。



注記

最低限必要な Open vSwitch (OVS) のバージョンは OVS 2.9 です。

OVN はデフォルトで Python 3.6 パッケージを使用します。

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 HA 環境でのみサポートされます。分散仮想ルーター (DVR) を使用して OVN をデプロイすることを推奨します。

1.1. 簡易ステップ: オーバークラウド上でのコンテナー化された OVN のデプロイ

OVN にすでに精通している場合には、以下の簡易ステップに従って、DVR を使用する OVN を HA 構成でオーバークラウド上にデプロイすることができます。

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

1.2. OVN のアーキテクチャー

OVN アーキテクチャーでは、Networking API をサポートするために OVS ML2 プラグインが OVN Modular Layer 2 (ML2) プラグインに置き換えられます。OVN は、Red Hat OpenStack Platform の頑強な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構成されます。

OVN ML2 プラグイン

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。このプラグインは、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)

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。このデータベースは、通常コントローラーノードで稼働し、TCP ポート **6641** をリッスンします。

OVN Northbound サービス (ovn-northd)

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。このサービスは通常コントローラーノードで実行されます。

OVN Southbound (SB) データベース (ovn-sb)

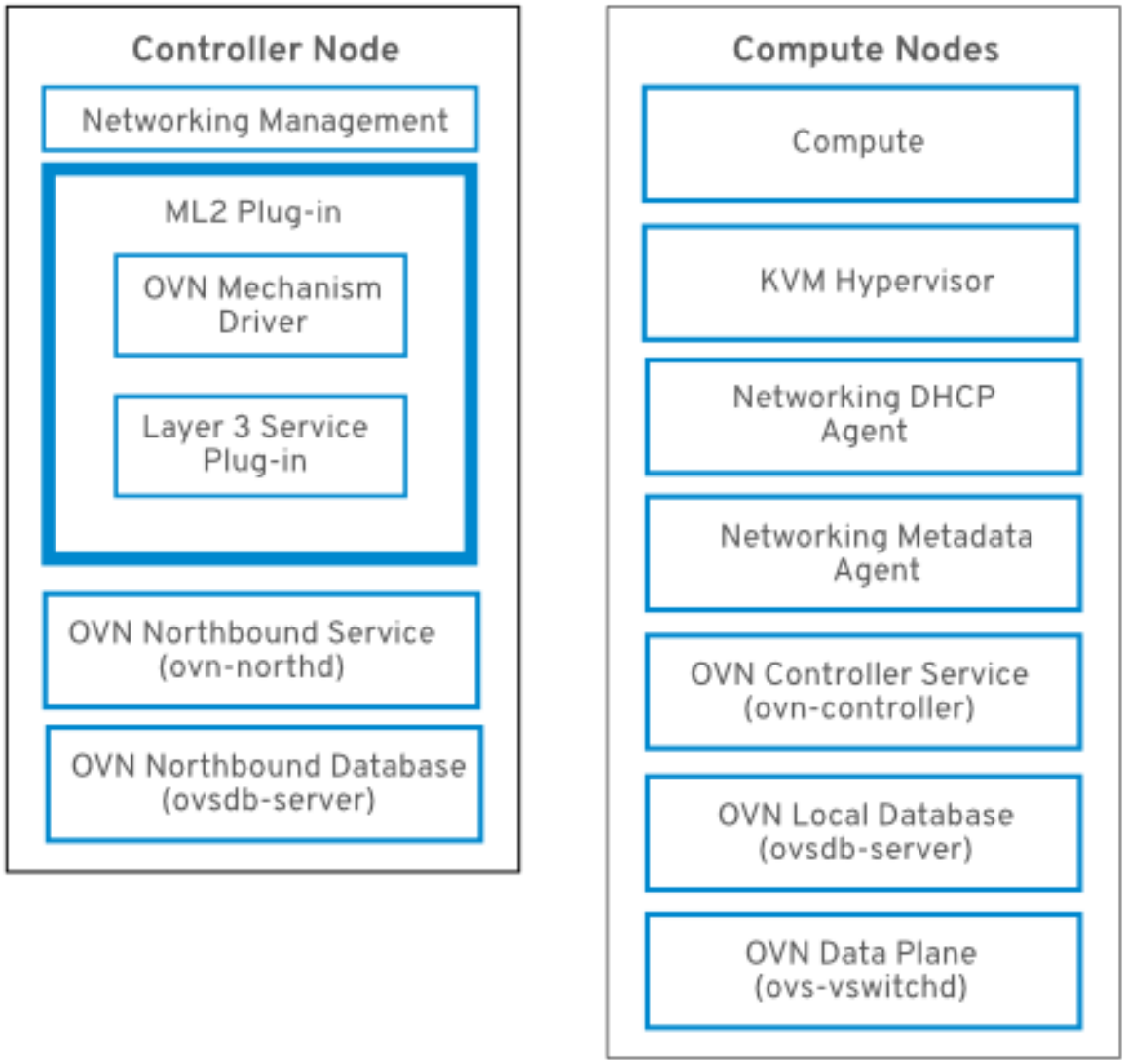
変換された論理データパスフローを保管します。このデータベースは、通常コントローラーノードで実行され、TCP ポート **6642** をリッスンします。

OVN コントローラー (ovn-controller)

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リングを行います。**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)

OVS インターフェース、ネットワーク名前空間、メタデーター 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** インスタンスを起動します。**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。


Controller Node
Networking Management
ML2 Plug-in
OVN Mechanism Driver
Layer 3 Service Plug-in
OVN Northbound Service (ovn-northd)
OVN Northbound Database (ovsdb-server)
Compute Nodes
Compute
KVM Hypervisor
Networking DHCP Agent
Networking Metadata Agent
OVN Controller Service (ovn-controller)
OVN Local Database (ovsdb-server)
OVN Data Plane (ovs-vswitchd)

第2章 OVN デプロイメントのプランニング

OVN は HA デプロイメントでのみデプロイします。分散仮想ルーター (DVR) を有効化してデプロイすることを推奨します。

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。

OVN を使用した DVR HA

DVR を使用する OVN を HA 環境でデプロイします。OVN は HA 環境でのみサポートされます。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。

2.1. コンピュートノード上の OVN-CONTROLLER

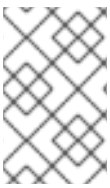
ovn-controller サービスは各コンピュートノードで実行され、OVN SB データベースサーバーに接続して論理フローを取得します。次に **ovn-controller** はその論理フローを OpenFlow の物理フローに変換して、OVS ブリッジ (**br-int**) に追加します。**ovs-vswitchd** と通信して OpenFlow フローをインストールするために、**ovn-controller** は **ovn-controller** の起動時に渡された UNIX ソケットパス (例: **unix:/var/run/openvswitch/db.sock**)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

director には、**ovn-dbs** という名前の OVN 用コンポーザブルサービスがあり、これには、ベースプロファイルと Pacemaker HA プロファイルの 2 つのプロファイルがあります。OVN の Northbound および Southbound データベースは、**ovsdb-server** サービスによりホストされます。同様に、**ovsdb-server** プロセスは、**ovs-vswitchd** と並行して実行され、OVS データベース (**conf.db**) をホスティングします。



注記

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

2.3. PACEMAKER を使用した高可用性と DVR

必要な HA プロファイルを使用するのに加えて、OVN を DVR を使用してデプロイして、ネットワークサービスの可用性を確保します。HA プロファイルが有効化されると、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 master 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。

ovsdb-server サービスは現在 **active-active** モードをサポートしていません。**master-slave** モードでの HA はサポートしています。これは、Pacemaker によって、リソースエージェントの Open Cluster Framework (OCF) スクリプトを使用して管理されます。**ovsdb-server** を **master** モードで実行すると、データベースへの書き込みアクセスが許可されますが、その他のスレーブの **ovsdb-server** サービスはすべて **master** からローカルにデータベースを複製し、書き込みアクセスは許可しません。

このプロファイル用の YAML ファイルは **tripleo-heat-templates/environments/services/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。

2.4. 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

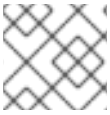
L3HA は OVN を使用してルーターのバランスを取り、元のゲートウェイノードに戻して、ノードがボトルネックとなるのを防ぎます。

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構成するその他すべてのゲートウェイノードをモニタリングします。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リングする必要はありません。



注記

ML2-OVS 構成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ース) 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役割を果たします)。
- **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

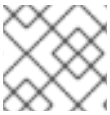
この BFD モニタリングメカニズムは、リンクのエラーのみで機能し、ルーティングのエラーには機能しません。

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 を使用した OVN のデプロイ



注記

本ガイドでは、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。

DVR を使用する OVN を HA 環境でデプロイするには、以下の手順を実行します。

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した場合には、これらのファイルに適切な設定を追加する必要がある場合があります。

2. **OS::TripleO::Compute::Ports::ExternalPort** を **OS::TripleO::Compute::Ports::ExternalPort: ../network/ports/external.yaml** などの適切な値に変更して、外部ネットワークにあるコンピュートノードのネットワークポートを設定します。

3. 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
```

実稼働環境 (または、ネットワークの分離や専用の NIC など、特別にカスタマイズする必要のあるテスト環境) の場合には、環境の例を参考にすることができます。ブリッジマッピングタイプのパラメーター (例: OVS) や外部向けのブリッジへの参照には、最新の注意を払う必要があります。

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/deployment/ovn/ovn-metadata-container-**

puppet.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

OVN を使用して内部 DNS をデプロイするには、以下を実行します。

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/neutron-ovn-dvr-ha.yaml
```

第4章 OVN のモニタリング

OVN 論理フローのモニタ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 の論理フローのモニタリング

OVN は論理フローを使用します。これは、優先度、マッチング、アクションで構成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。

4.2. OPENFLOWS のモニタリング

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
```