OpenShift Container Platform 4.5

ネットワーク

クラスターネットワークの設定および管理

Last Updated: 2022-11-28

OpenShift Container Platform 4.5 ネットワーク

クラスターネットワークの設定および管理

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/Networking.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

この文書では、DNS、ingress および Pod ネットワークを含む、OpenShift Container Platform のクラスターネットワークを設定し、管理する方法を説明します。

目次

第1章 ネットワークについて 9
1.1. OPENSHIFT CONTAINER PLATFORM DNS 9

第2章 ホストへのアクセス 10
2.1. インストーラーでプロビジョニングされるインフラストラクチャークラスターでの AMAZON WEB SERVICES のホストへのアクセス 10

第3章 OPENSHIFT CONTAINER PLATFORM における CLUSTER NETWORK OPERATOR 11
3.1. CLUSTER NETWORK OPERATOR 11
3.2. クラスターネットワーク設定の表示 11
3.3. CLUSTER NETWORK OPERATOR のステータス表示 12
3.4. CLUSTER NETWORK OPERATOR ログの表示 12
3.5. CLUSTER NETWORK OPERATOR (CNO) の設定 13
3.5.1. OpenShift SDN デフォルト CNI ネットワークプロバイダーの設定パラメーター 14
3.5.2. OVN-Kubernetes デフォルト CNI ネットワークプロバイダーの設定パラメーター 14
3.5.3. Cluster Network Operator の設定例 15

第4章 OPENSHIFT CONTAINER PLATFORM の DNS OPERATOR 16
4.1. DNS OPERATOR 16
4.2. デフォルト DNS の表示 16
4.3. DNS 転送の使用 17
4.4. DNS OPERATOR のステータス 19
4.5. DNS OPERATOR ログ 19

第5章 OPENSHIFT CONTAINER PLATFORM の INGRESS OPERATOR 20
5.1. INGRESS 設定アセット 20
5.2. イメージコントローラー設定パラメーター 20
5.2.1. Ingress コントローラーの TLS プロファイル 26
5.2.2. Ingress コントローラーエンドポイントの公開ストラテジー 27
5.3. デフォルト INGRESS コントローラーの表示 27
5.4. INGRESS OPERATOR ステータスの表示 28
5.5. INGRESS コントローラーログの表示 28
5.6. INGRESS コントローラーステータスの表示 28

第6章 INGRESS コントローラーの設定 29
6.1. カスタムデフォルト証明書の設定 29
6.2. INGRESS コントローラーのスケーリング 30
6.3. INGRESS アクセスロギングの設定 31
6.4. INGRESS コントローラーのシャード化 33
6.4.1. ルートラベルを使用した Ingress コントローラーのシャード化の設定 34
6.4.2. namespace ラベルを使用した Ingress コントローラーのシャード化の設定 34
6.5. 内部ロードバランサーを使用するように INGRESS コントローラーを設定する 35
6.6. クラスターを内部に配置するようにのデフォルト INGRESS コントローラーを設定する 37
6.7. ルートの受付ポリシーの設定 37
6.8. ワイルドカードルートの使用 38
6.9. HTTP/2 INGRESS 接続の有効化 39
6.10. 関連情報 39

第7章 ベアメタルクラスターでの SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の使用 40
7.1. OPENSHIFT CONTAINER PLATFORM での SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) のサポート 40
7.1.1. SCTP プロトコルを使用した設定例 40
7.2. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の有効化 41

7.3. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) が有効になっていることの確認 42

第8章 PTP ハードウェアの設定 45
8.1. PTP ハードウェアについて 45
8.2. PTP ネットワークデバイスの自動検出 45
8.3. PTP OPERATOR のインストール 46
8.3.1. CLI: PTP Operator のインストール 46
8.3.2. Web コンソール: PTP Operator のインストール 47
8.4. LINUXPTP サービスの設定 48

第9章 ネットワークポリシー 52
9.1. ネットワークポリシーについて 52
9.1.1. ネットワークポリシーについて 52
9.1.2. ネットワークポリシーの最適化 55
9.1.3. 次のステップ 55
9.1.4. 関連情報 55
9.2. ネットワークポリシーの作成 55
9.2.1. ネットワークポリシーの作成 55
9.2.2. サンプル NetworkPolicy オブジェクト 57
9.3. ネットワークポリシーの表示 57
9.3.1. ネットワークポリシーの表示 57
9.3.2. サンプル NetworkPolicy オブジェクト 59
9.4. ネットワークポリシーの編集 59
9.4.1. ネットワークポリシーの編集 59
9.4.2. サンプル NetworkPolicy オブジェクト 61
9.4.3. 関連情報 61
9.5. ネットワークポリシーの削除 61
9.5.1. ネットワークポリシーの削除 61
9.6. プロジェクトのデフォルトネットワークポリシーの定義 62
9.6.1. 新規プロジェクトのテンプレートの変更 62
9.6.2. 新規プロジェクトへのネットワークポリシーの追加 63
9.7. ネットワークポリシーを使用したマルチテナント分離の設定 65
9.7.1. ネットワークポリシーを使用したマルチテナント分離の設定 65
9.7.2. 次のステップ 67
9.7.3. 関連情報 67

第10章 複数ネットワーク 68
10.1. 複数ネットワークについて 68
10.1.1. 追加ネットワークの使用シナリオ 68
10.1.2. OpenShift Container Platform の追加ネットワーク 68
10.2. POD の追加のネットワークへの割り当て 69
10.2.1. Pod の追加ネットワークへの追加 69
10.2.1.1. Pod 固有のアドレスおよびルーティングオプションの指定 71
10.3. 追加ネットワークからの POD の削除 74
10.3.1. 追加ネットワークからの Pod の削除 74
10.4. ブリッジネットワークの設定 75
10.4.1. ブリッジ CNI プラグインを使用した追加ネットワーク割り当ての作成 75
10.4.1.1. ブリッジの設定 76
10.4.1.1.1. ブリッジ設定の例 78
10.4.1.2. IPAM CNI プラグインの設定 78
10.4.1.2.1. 静的 IP アドレス割り当ての設定 78
10.4.1.2.2. 動的 IP アドレス割り当ての設定 79
10.4.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定 80
10.4.1.2.4. 静的 IP アドレス割り当ての設定例 81

10.4.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例 81
10.4.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例 81
10.4.2. 次のステップ 81
10.5. ホストデバイスネットワークの設定 82
10.5.1. ホストデバイス CNI プラグインを使用した追加ネットワーク割り当ての作成 82
10.5.1.1. ホストデバイスの設定 83
10.5.1.1.1. ホストデバイス設定例 84
10.5.1.2. IPAM CNI プラグインの設定 85
10.5.1.2.1. 静的 IP アドレス割り当ての設定 85
10.5.1.2.2. 動的 IP アドレス割り当ての設定 86
10.5.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定 87
10.5.1.2.4. 静的 IP アドレス割り当ての設定例 88
10.5.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例 88
10.5.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例 88
10.5.2. 次のステップ 88
10.6. IPVLAN ネットワークの設定 89
10.6.1. IPVLAN CNI プラグインを使用した追加のネットワーク割り当ての作成 89
10.6.1.1. IPVLAN の設定 90
10.6.1.1.1. IPVLAN 設定例 91
10.6.1.2. IPAM CNI プラグインの設定 92
10.6.1.2.1. 静的 IP アドレス割り当ての設定 92
10.6.1.2.2. 動的 IP アドレス割り当ての設定 93
10.6.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定 94
10.6.1.2.4. 静的 IP アドレス割り当ての設定例 94
10.6.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例 94
10.6.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例 95
10.6.2. 次のステップ 95
10.7. 基本的なカスタマイズを使用した MACVLAN ネットワークの設定 95
10.7.1. macvlan CNI プラグインを使用した追加ネットワーク割り当ての作成 95
10.7.1.1. macvlan CNI プラグインの設定 96
10.7.1.1.1. macvlan 設定の例 97
10.7.1.2. IPAM CNI プラグインの設定 97
10.7.1.2.1. 静的 IPAM 設定 YAML 98
10.7.1.2.2. 動的 IPAM 設定 YAML 98
10.7.1.2.3. 静的 IP アドレス割り当ての設定例 99
10.7.1.2.4. 動的 IP アドレス割り当ての設定例 99
10.7.2. 次のステップ 99
10.8. MACVLAN ネットワークの設定 99
10.8.1. macvlan CNI プラグインを使用した追加ネットワーク割り当ての作成 100
10.8.1.1. macvlan CNI プラグインの設定 101
10.8.1.1.1. macvlan 設定の例 102
10.8.1.2. IPAM CNI プラグインの設定 102
10.8.1.2.1. 静的 IP アドレス割り当ての設定 103
10.8.1.2.2. 動的 IP アドレス割り当ての設定 104
10.8.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定 104
10.8.1.2.4. 静的 IP アドレス割り当ての設定例 105
10.8.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例 105
10.8.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例 105
10.8.2. 次のステップ 106
10.9. 追加ネットワークの編集 106
10.9.1. 追加ネットワーク割り当て定義の変更 106
10.10. 追加ネットワークの削除 107
10.10.1. 追加ネットワーク割り当て定義の削除 107

第11章 ハードウェアネットワーク 108
11.1. SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) ハードウェアネットワークについて 108
11.1.1. SR-IOV ネットワークデバイスを管理するコンポーネント 108
11.1.1.1. サポートされるデバイス 109
11.1.1.2. SR-IOV ネットワークデバイスの自動検出 109
11.1.1.2.1. SriovNetworkNodeState オブジェクトの例 109
11.1.1.3. Pod での Virtual Function (VF) の使用例 110
11.1.2. 次のステップ 112
11.2. SR-IOV NETWORK OPERATOR のインストール 112
11.2.1. SR-IOV Network Operator のインストール 112
11.2.1.1. CLI: SR-IOV Network Operator のインストール 112
11.2.1.2. Web コンソール: SR-IOV Network Operator のインストール 114
11.2.2. 次のステップ 115
11.3. SR-IOV NETWORK OPERATOR の設定 115
11.3.1. SR-IOV Network Operator の設定 115
11.3.1.1. Network Resources Injector について 115
11.3.1.2. SR-IOV Operator Admission Controller Webhook について 116
11.3.1.3. カスタムノードセレクターについて 116
11.3.1.4. Network Resources Injector の無効化または有効化 116
11.3.1.5. SR-IOV Operator Admission Controller Webhook の無効化または有効化 117
11.3.1.6. SRIOV Network Config Daemon のカスタム NodeSelector の設定 117
11.3.2. 次のステップ 118
11.4. SR-IOV ネットワークデバイスの設定 118
11.4.1. SR-IOV ネットワークノード設定オブジェクト 118
11.4.1.1. SR-IOV デバイスの仮想機能 (VF) パーティション設定 120
11.4.2. SR-IOV ネットワークデバイスの設定 121
11.4.3. 次のステップ 122
11.5. SR-IOV イーサネットネットワーク割り当ての設定 122
11.5.1. イーサネットデバイス設定オブジェクト 122
11.5.1.1. IPAM CNI プラグインの設定 123
11.5.1.1.1. 静的 IP アドレス割り当ての設定 124
11.5.1.1.2. 動的 IP アドレス割り当ての設定 125
11.5.1.1.3. Whereabouts を使用した動的 IP アドレス割り当ての設定 125
11.5.1.1.4. 静的 IP アドレス割り当ての設定例 126
11.5.1.1.5. DHCP を使用した動的 IP アドレス割り当ての設定例 126
11.5.1.1.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例 126
11.5.2. SR-IOV の追加ネットワークの設定 127
11.5.3. 次のステップ 128
11.5.4. 関連情報 128
11.6. POD の SR-IOV の追加ネットワークへの追加 128
11.6.1. ネットワーク割り当てのランタイム設定 128
11.6.1.1. イーサネットベースの SR-IOV 割り当てのランタイム設定 128
11.6.2. Pod の追加ネットワークへの追加 129
11.6.3. Non-Uniform Memory Access (NUMA) で配置された SR-IOV Pod の作成 131
11.6.4. 関連情報 132
11.7. 高パフォーマンスのマルチキャストの使用 132
11.7.1. 高パフォーマンスのマルチキャストの設定 132
11.7.2. マルチキャストでの SR-IOV インターフェイスの使用 133
11.8. DPDK および RDMA モードでの仮想機能 (VF) の使用 134
11.8.1. DPDK および RDMA モードでの仮想機能 (VF) の使用例 134
11.8.2. 前提条件 135
11.8.3. Intel NIC を使用した DPDK モードでの仮想機能 (VF) の使用例 135
11.8.4. Mellanox NIC を使用した DPDK モードでの仮想機能 (VF) の使用例 138

11.8.5. Mellanox NIC を使った RDMA モードでの仮想機能 (VF) の例 141

第12章 OPENSHIFT SDN デフォルト CNI ネットワークプロバイダー 144
12.1. OPENSHIFT SDN デフォルト CNI ネットワークプロバイダーについて 144
12.1.1. OpenShift SDN ネットワーク分離モード 144
12.1.2. サポートされるデフォルトの CNI ネットワークプロバイダー機能マトリクス 144
12.2. プロジェクトの EGRESS IP の設定 145
12.2.1. プロジェクトの egress トラフィックについての egress IP アドレスの割り当て 145
12.2.1.1. 自動的に割り当てられた egress IP アドレスを使用する場合の考慮事項 146
12.2.1.2. 手動で割り当てられた egress IP アドレスを使用する場合の考慮事項 146
12.2.2. namespace の自動的に割り当てられた egress IP アドレスの有効化 147
12.2.3. namespace の手動で割り当てられた egress IP アドレスの設定 148
12.3. 外部 IP アドレスへのアクセスを制御するための EGRESS ファイアウォールの設定 149
12.3.1. egress ファイアウォールのプロジェクトでの機能 149
12.3.1.1. egress ファイアウォールの制限 150
12.3.1.2. egress ネットワークポリシールールのマッチング順序 150
12.3.1.3. DNS (Domain Name Server) 解決の仕組み 151
12.3.2. EgressNetworkPolicy カスタムリソース (CR) オブジェクト 151
12.3.2.1. EgressNetworkPolicy ルール 151
12.3.2.2. EgressNetworkPolicy CR オブジェクトの例 152
12.3.3. egress ファイアウォールポリシーオブジェクトの作成 152
12.4. プロジェクトの EGRESS ファイアウォールの編集 153
12.4.1. EgressNetworkPolicy オブジェクトの編集 153
12.4.2. EgressNetworkPolicy カスタムリソース (CR) オブジェクト 154
12.4.2.1. EgressNetworkPolicy ルール 154
12.4.2.2. EgressNetworkPolicy CR オブジェクトの例 155
12.5. プロジェクトからの EGRESS ファイアウォールの削除 155
12.5.1. EgressNetworkPolicy オブジェクトの削除 155
12.6. EGRESS ルーター POD の使用についての考慮事項 156
12.6.1. Egress ルーター Pod について 156
12.6.1.1. Egress ルーターモード 156
12.6.1.2. egress ルーター Pod の実装 157
12.6.1.3. デプロイメントに関する考慮事項 157
12.6.1.4. フェイルオーバー設定 158
12.6.2. 関連情報 158
12.7. リダイレクトモードでの EGRESS ルーター POD のデプロイ 158
12.7.1. リダイレクトモードの egress ルーター Pod 仕様 158
12.7.2. egress 宛先設定形式 160
12.7.3. リダイレクトモードでの egress ルーター Pod のデプロイ 161
12.7.4. 関連情報 161
12.8. HTTP プロキシーモードでの EGRESS ルーター POD のデプロイ 161
12.8.1. HTTP モードの egress ルーター Pod 仕様 161
12.8.2. egress 宛先設定形式 162
12.8.3. HTTP プロキシーモードでの egress ルーター Pod のデプロイ 163
12.8.4. 関連情報 164
12.9. DNS プロキシーモードでの EGRESS ルーター POD のデプロイ 164
12.9.1. DNS モードの egress ルーター Pod 仕様 164
12.9.2. egress 宛先設定形式 165
12.9.3. DNS プロキシーモードでの egress ルーター Pod のデプロイ 166
12.9.4. 関連情報 167
12.10. 設定マップからの EGRESS ルーター POD 宛先一覧の設定 167
12.10.1. 設定マップを使用した egress ルーター宛先マッピングの設定 167
12.10.2. 関連情報 168

12.11. プロジェクトのマルチキャストの有効化 168
12.11.1. マルチキャストについて 168
12.11.2. Pod 間のマルチキャストの有効化 169
12.12. プロジェクトのマルチキャストの無効化 171
12.12.1. Pod 間のマルチキャストの無効化 171
12.13. OPENSHIFT SDN を使用したネットワーク分離の設定 171
12.13.1. 前提条件 172
12.13.2. プロジェクトの結合 172
12.13.3. プロジェクトの分離 172
12.13.4. プロジェクトのネットワーク分離の無効化 173
12.14. KUBE-PROXY の設定 173
12.14.1. iptables ルールの同期について 173
12.14.2. kube-proxy 設定パラメーター 173
12.14.3. kube-proxy 設定の変化 174

第13章 OVN-KUBERNETES デフォルト CNI ネットワークプロバイダー 176
13.1. OVN-KUBERNETES デフォルト CONTAINER NETWORK INTERFACE (CNI) ネットワークプロバイダーについて 176
13.1.1. OVN-Kubernetes の機能 176
13.1.2. サポートされるデフォルトの CNI ネットワークプロバイダー機能マトリクス 176
13.1.3. OVN-Kubernetes の公開メトリクス 177
13.2. OPENSHIFT SDN デフォルト CNI ネットワークプロバイダーからの移行 177
13.2.1. OVN-Kubernetes デフォルト CNI ネットワークプロバイダーへの移行 177
13.3. OPENSHIFT SDN ネットワークプロバイダーへのロールバック 179
13.3.1. デフォルトの CNI ネットワークプロバイダーの OpenShift SDN へのロールバック 179
13.4. プロジェクトのマルチキャストの有効化 181
13.4.1. マルチキャストについて 181
13.4.2. Pod 間のマルチキャストの有効化 181
13.5. プロジェクトのマルチキャストの無効化 183
13.5.1. Pod 間のマルチキャストの無効化 184

第14章 ルートの作成 185
14.1. ルート設定 185
14.1.1. ルートのタイムアウトの設定 185
14.1.2. HTTP Strict Transport Security の有効化 185
14.1.3. スループット関連の問題のトラブルシューティング 186
14.1.4. Cookie に使用によるルートのステートフル性の維持 187
14.1.4.1. Cookie を使用したルートのアノテーション 187
14.1.5. パスベースのルート 188
14.1.6. ルート固有のアノテーション 189
14.1.7. ルートの受付ポリシーの設定 192
14.2. セキュリティー保護されたルート 192
14.2.1. カスタム証明書を使用した re-encrypt ルートの作成 193
14.2.2. カスタム証明書を使用した edge ルートの作成 194
14.2.3. passthrough ルートの作成 195

第15章 INGRESS クラスタートラフィックの設定 197
15.1. INGRESS クラスタートラフィックの設定の概要 197
15.2. サービスの EXTERNALIP の設定 197
15.2.1. 前提条件 197
15.2.2. ExternalIP について 197
15.2.2.1. ExternalIP の設定 198
15.2.2.2. 外部 IP アドレスの割り当ての制限 199
15.2.2.3. ポリシーオブジェクトの例 200

15.2.3. ExternalIP アドレスブロックの設定 201
外部 IP 設定の例 202
15.2.4. クラスターの外部 IP アドレスブロックの設定 203
15.2.5. 次のステップ 203
15.3. INGRESS コントローラーを使用した INGRESS クラスターの設定 204
15.3.1. Ingress コントローラーおよびルートの使用 204
15.3.2. 前提条件 204
15.3.3. プロジェクトおよびサービスの作成 205
15.3.4. ルートの作成によるサービスの公開 205
15.3.5. ルートラベルを使用した Ingress コントローラーのシャード化の設定 206
15.3.6. namespace ラベルを使用した Ingress コントローラーのシャード化の設定 207
15.3.7. 関連情報 208
15.4. ロードバランサーを使用した INGRESS クラスターの設定 208
15.4.1. ロードバランサーを使用したトラフィックのクラスターへの送信 208
15.4.2. 前提条件 209
15.4.3. プロジェクトおよびサービスの作成 209
15.4.4. ルートの作成によるサービスの公開 210
15.4.5. ロードバランサーサービスの作成 211
15.5. サービスの外部 IP を使用した INGRESS クラスタートラフィックの設定 212
15.5.1. 前提条件 212
15.5.2. ExternalIP のサービスへの割り当て 213
15.5.3. 関連情報 214
15.6. NODEPORT を使用した INGRESS クラスタートラフィックの設定 214
15.6.1. NodePort を使用したトラフィックのクラスターへの送信 214
15.6.2. 前提条件 214
15.6.3. プロジェクトおよびサービスの作成 215
15.6.4. ルートの作成によるサービスの公開 215

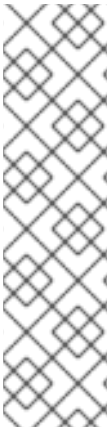
第16章 クラスター全体のプロキシーの設定 217
16.1. 前提条件 217
16.2. クラスター全体のプロキシーの有効化 217
16.3. クラスター全体のプロキシーの削除 219

第17章 カスタム PKI の設定 221
17.1. インストール時のクラスター全体のプロキシーの設定 221
17.2. クラスター全体のプロキシーの有効化 223
17.3. OPERATOR を使用した証明書の挿入 225

第18章 RHOSP での負荷分散 227
18.1. KURYR SDN を使用した OCTAVIA OVN ロードバランサープロバイダードライバーの使用 227
18.2. OCTAVIA を使用したアプリケーショントラフィック用のクラスターのスケーリング 228
18.2.1. Octavia を使用したクラスターのスケーリング 229
18.2.2. Octavia の使用による Kuryr を使用するクラスターのスケーリング 230
18.3. RHOSP OCTAVIA を使用した INGRESS トラフィックのスケーリング 230

第1章 ネットワークについて

Kubernetes は、確実に Pod 間がネットワークで接続されるようにし、内部ネットワークから IP アドレスを各 Pod に割り当てます。こうすることで、Pod 内の全コンテナーが同じホスト上にいるかように動作します。各 Pod に IP アドレスを割り当てると、ポートの割り当て、ネットワーク、名前の指定、サービス検出、負荷分散、アプリケーション設定、移行などの点で、Pod を物理ホストや仮想マシンのように扱うことができます。



注記

一部のクラウドプラットフォームでは、169.254.169.254 IP アドレスでリッスンするメタデータ API があります。これは、IPv4 **169.254.0.0/16** CIDR ブロックのリンクローカル IP アドレスです。

この CIDR ブロックは Pod ネットワークから到達できません。これらの IP アドレスへのアクセスを必要とする Pod には、Pod 仕様の **spec.hostNetwork** フィールドを **true** に設定して、ホストのネットワークアクセスが付与される必要があります。

Pod ホストのネットワークアクセスを許可する場合、Pod に基礎となるネットワークインフラストラクチャーへの特権アクセスを付与します。

1.1. OPENSHIFT CONTAINER PLATFORM DNS

フロントエンドサービスやバックエンドサービスなど、複数のサービスを実行して複数の Pod で使用している場合、フロントエンド Pod がバックエンドサービスと通信できるように、ユーザー名、サービス IP などの環境変数を作成します。サービスが削除され、再作成される場合には、新規の IP アドレスがそのサービスに割り当てられるので、フロントエンド Pod がサービス IP の環境変数の更新された値を取得するには、これを再作成する必要があります。さらに、バックエンドサービスは、フロントエンド Pod を作成する前に作成し、サービス IP が正しく生成され、フロントエンド Pod に環境変数として提供できるようにする必要があります。

そのため、OpenShift Container Platform には DNS が組み込まれており、これにより、サービスは、サービス IP/ポートと共にサービス DNS によって到達可能になります。

第2章 ホストへのアクセス

OpenShift Container Platform インスタンスにアクセスして、セキュアなシェル (SSH) アクセスでマスターノードにアクセスするために bastion ホストを作成する方法を学びます。

2.1. インストーラーでプロビジョニングされるインフラストラクチャークラスターでの AMAZON WEB SERVICES のホストへのアクセス

OpenShift Container Platform インストーラーは、OpenShift Container Platform クラスターにプロビジョニングされる Amazon Elastic Compute Cloud (Amazon EC2) インスタンスのパブリック IP アドレスを作成しません。OpenShift Container Platform ホストに対して SSH を実行できるようにするには、以下の手順を実行する必要があります。

手順

1. **openshift-install** コマンドで作成される仮想プライベートクラウド (VPC) に対する SSH アクセスを可能にするセキュリティーグループを作成します。

2. インストーラーが作成したパブリックサブネットのいずれかに Amazon EC2 インスタンスを作成します。

3. パブリック IP アドレスを、作成した Amazon EC2 インスタンスに関連付けます。OpenShift Container Platform のインストールとは異なり、作成した Amazon EC2 インスタンスを SSH キーペアに関連付ける必要があります。これにはインターネットを OpenShift Container Platform クラスターの VPC にブリッジ接続するための SSH bastion としてのみの単純な機能しかないため、このインスタンスにどのオペレーティングシステムを選択しても問題ありません。どの Amazon Machine Image (AMI) を使用するかについては、注意が必要です。たとえば、Red Hat Enterprise Linux CoreOS (RHCOS) では、インストーラーと同様に、Ignition でキーを指定することができます。

4. Amazon EC2 インスタンスをプロビジョニングし、これに対して SSH を実行した後に、OpenShift Container Platform インストールに関連付けた SSH キーを追加する必要があります。このキーは bastion インスタンスのキーとは異なる場合がありますが、異なるキーにしなければならない訳ではありません。

   注記

   直接の SSH アクセスは、障害復旧を目的とする場合にのみ推奨されます。Kubernetes API が応答する場合、特権付き Pod を代わりに実行します。

5. **oc get nodes** を実行し、出力を検査し、マスターであるノードのいずれかを選択します。ホスト名は **ip-10-0-1-163.ec2.internal** に類似したものになります。

6. Amazon EC2 に手動でデプロイした bastion SSH ホストから、そのマスターホストに対して SSH を実行します。インストール時に指定したものと同じ SSH キーを使用するようにします。

```
$ ssh -i <ssh-key-path> core@<master-hostname>
```

第3章 OPENSHIFT CONTAINER PLATFORM における CLUSTER NETWORK OPERATOR

Cluster Network Operator (CNO) は、インストール時にクラスター用に選択される Container Network Interface (CNI) デフォルトネットワークプロバイダープラグインを含む、OpenShift Container Platform クラスターの各種のクラスターネットワークコンポーネントをデプロイし、これらを管理します。

3.1. CLUSTER NETWORK OPERATOR

Cluster Network Operator は、**operator.openshift.io** API グループから **network** API を実装します。Operator は、デーモンセットを使用して OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグイン、またはクラスターのインストール時に選択したデフォルトネットワークプロバイダープラグインをデプロイします。

手順

Cluster Network Operator は、インストール時に Kubernetes **Deployment** としてデプロイされます。

1. 以下のコマンドを実行して Deployment のステータスを表示します。

```
$ oc get -n openshift-network-operator deployment/network-operator
```

出力例

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
network-operator   1/1     1            1           56m
```

2. 以下のコマンドを実行して、Cluster Network Operator の状態を表示します。

```
$ oc get clusteroperator/network
```

出力例

```
NAME      VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.5.4     True        False         False      50m
```

以下のフィールドは、Operator のステータス (**AVAILABLE**、**PROGRESSING**、および **DEGRADED**) についての情報を提供します。**AVAILABLE** フィールドは、Cluster Network Operator が Available ステータス条件を報告する場合に **True** になります。

3.2. クラスターネットワーク設定の表示

すべての新規 OpenShift Container Platform インストールには、**cluster** という名前の **network.config** オブジェクトがあります。

手順

- **oc describe** コマンドを使用して、クラスターネットワーク設定を表示します。

```
$ oc describe network.config/cluster
```

出力例

```
Name:         cluster
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  config.openshift.io/v1
Kind:         Network
Metadata:
  Self Link:           /apis/config.openshift.io/v1/networks/cluster
Spec: (1)
  Cluster Network:
    Cidr:         10.128.0.0/14
    Host Prefix:  23
  Network Type:   OpenShiftSDN
  Service Network:
    172.30.0.0/16
Status: (2)
  Cluster Network:
    Cidr:               10.128.0.0/14
    Host Prefix:        23
  Cluster Network MTU:  8951
  Network Type:         OpenShiftSDN
  Service Network:
    172.30.0.0/16
Events:  <none>
```

1. **Spec** フィールドは、クラスターネットワークの設定済みの状態を表示します。
2. **Status** フィールドは、クラスターネットワークの現在の状態を表示します。

3.3. CLUSTER NETWORK OPERATOR のステータス表示

oc describe コマンドを使用して、Cluster Network Operator のステータスを検査し、その詳細を表示することができます。

手順

- 以下のコマンドを実行して、Cluster Network Operator のステータスを表示します。

```
$ oc describe clusteroperators/network
```

3.4. CLUSTER NETWORK OPERATOR ログの表示

oc logs コマンドを使用して、Cluster Network Operator ログを表示できます。

手順

- 以下のコマンドを実行して、Cluster Network Operator のログを表示します。

```
$ oc logs --namespace=openshift-network-operator deployment/network-operator
```

重要

Open Virtual Networking (OVN) Kubernetes ネットワークプラグインは、テクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

OVN テクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/articles/4380121 を参照してください。

3.5. CLUSTER NETWORK OPERATOR (CNO) の設定

クラスターネットワークの設定は、Cluster Network Operator (CNO) 設定の一部として指定され、**cluster** という名前の CR オブジェクトに保存されます。CR は **operator.openshift.io** API グループの **Network** API のパラメーターを指定します。

defaultNetwork パラメーターのパラメーター値を CNO CR に設定することにより、OpenShift Container Platform クラスターのクラスターネットワーク設定を指定できます。以下の CR は、CNO のデフォルト設定を表示し、設定可能なパラメーターと有効なパラメーターの値の両方について説明しています。

Cluster Network Operator CR

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork: (1)
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: (2)
  - 172.30.0.0/16
  defaultNetwork: (3)
    ...
  kubeProxyConfig: (4)
    iptablesSyncPeriod: 30s (5)
    proxyArguments:
      iptables-min-sync-period: (6)
      - 0s
```

1. Pod ID アドレスの割り当て、サブネット接頭辞の長さの個別ノードへの割り当てに使用される IP アドレスのブロックを指定する一覧です。
2. サービスの IP アドレスのブロック。OpenShift SDN Container Network Interface (CNI) ネットワークプロバイダーは、サービスネットワークの単一 IP アドレスブロックのみをサポートします。
3. クラスターネットワークのデフォルトの CNI ネットワークプロバイダーを設定します。
4.

このオブジェクトのパラメーターは、Kubernetes ネットワークプロキシー (kube-proxy) 設定を指定します。OVN-Kubernetes デフォルト CNI ネットワークプロバイダーを使用している場合、

5. **iptables** ルールの更新期間。デフォルト値は **30s** です。有効な接尾辞には、**s**、**m**、および **h** などが含まれ、これらについては、Go Package time ドキュメントで説明されています。

> **注記**
>
> OpenShift Container Platform 4.3 以降で強化されたパフォーマンスの向上により、**iptablesSyncPeriod** パラメーターを調整する必要はなくなりました。

6. **iptables** ルールを更新する前の最小期間。このパラメーターにより、更新の頻度が高くなり過ぎないようにできます。有効な接尾辞には、**s**、**m**、および **h** などが含まれ、これらについては、Go Package time で説明されています。

3.5.1. OpenShift SDN デフォルト CNI ネットワークプロバイダーの設定パラメーター

以下の YAML オブジェクトは、OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーの設定パラメーターについて説明しています。

> **注記**
>
> クラスターのインストール時にのみデフォルト CNI ネットワークプロバイダーの設定を変更することができます。

```
defaultNetwork:
  type: OpenShiftSDN (1)
  openshiftSDNConfig: (2)
    mode: NetworkPolicy (3)
    mtu: 1450 (4)
    vxlanPort: 4789 (5)
```

1. 使用されるデフォルト CNI ネットワークプロバイダープラグイン。
2. OpenShift SDN 固有の設定パラメーター。
3. OpenShiftSDN のネットワーク分離モード。
4. VXLAN オーバーレイネットワークの最大転送単位 (MTU)。通常、この値は自動的に設定されます。
5. すべての VXLAN パケットに使用するポート。デフォルト値は **4789** です。

3.5.2. OVN-Kubernetes デフォルト CNI ネットワークプロバイダーの設定パラメーター

以下の YAML オブジェクトは OVN-Kubernetes デフォルト CNI ネットワークプロバイダーの設定パラメーターについて説明しています。

注記

クラスターのインストール時にのみデフォルト CNI ネットワークプロバイダーの設定を変更することができます。

```
defaultNetwork:
  type: OVNKubernetes (1)
  ovnKubernetesConfig: (2)
    mtu: 1400 (3)
    genevePort: 6081 (4)
```

1. 使用されるデフォルト CNI ネットワークプロバイダープラグイン。
2. OVN-Kubernetes 固有の設定パラメーター。
3. Geneve (Generic Network Virtualization Encapsulation) オーバーレイネットワークの MTU。通常、この値は自動的に設定されます。
4. Geneve オーバーレイネットワークの UDP ポート。

3.5.3. Cluster Network Operator の設定例

以下の例のように、CNO の完全な CR オブジェクトが表示されます。

Cluster Network Operator のサンプル CR

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  defaultNetwork:
    type: OpenShiftSDN
    openshiftSDNConfig:
      mode: NetworkPolicy
      mtu: 1450
      vxlanPort: 4789
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period:
      - 0s
```

第4章 OPENSHIFT CONTAINER PLATFORM の DNS OPERATOR

DNS Operator は、Pod に対して名前解決サービスを提供するために CoreDNS をデプロイし、これを管理し、OpenShift 内での DNS ベースの Kubernetes サービス検出を可能にします。

4.1. DNS OPERATOR

DNS Operator は、**operator.openshift.io** API グループから **dns** API を実装します。この Operator は、デーモンセットを使用して CoreDNS をデプロイし、デーモンセットのサービスを作成し、kubelet を Pod に対して名前解決に CoreDNS サービス IP を使用するように指示するように設定します。

手順

DNS Operator は、インストール時に **Deployment** オブジェクトを使用してデプロイされます。

1. **oc get** コマンドを使用してデプロイメントのステータスを表示します。

```
$ oc get -n openshift-dns-operator deployment/dns-operator
```

出力例

```
NAME          READY     UP-TO-DATE   AVAILABLE   AGE
dns-operator   1/1       1            1           23h
```

2. **oc get** コマンドを使用して DNS Operator の状態を表示します。

```
$ oc get clusteroperator/dns
```

出力例

```
NAME      VERSION     AVAILABLE   PROGRESSING   DEGRADED   SINCE
dns       4.1.0-0.11  True        False         False      92m
```

AVAILABLE、 **PROGRESSING** および **DEGRADED** は、Operator のステータスについての情報を提供します。**AVAILABLE** は、CoreDNS デーモンセットからの 1 つ以上の Pod が **Available** ステータス条件を報告する場合は **True** になります。

4.2. デフォルト DNS の表示

すべての新規 OpenShift Container Platform インストールには、**default** という名前の **dns.operator** があります。

手順

1. **oc describe** コマンドを使用してデフォルトの **dns** を表示します。

```
$ oc describe dns.operator/default
```

出力例

■

```
Name:         default
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         DNS
...
Status:
  Cluster Domain:  cluster.local (1)
  Cluster IP:      172.30.0.10 (2)
...
```

1. Cluster Domain フィールドは、完全修飾 Pod およびサービスドメイン名を作成するために使用されるベース DNS ドメインです。
2. クラスター IP は、Pod が名前解決のためにクエリーするアドレスです。IP は、サービス CIDR 範囲の 10 番目のアドレスで定義されます。

2. クラスターのサービス CIDR を見つけるには、**oc get** コマンドを使用します。

```
$ oc get networks.config/cluster -o jsonpath='{$.status.serviceNetwork}'
```

出力例

```
[172.30.0.0/16]
```

4.3. DNS 転送の使用

DNS 転送を使用すると、指定のゾーンにどのネームサーバーを使用するかを指定することで、ゾーンごとに **/etc/resolv.conf** で特定される転送設定をオーバーライドできます。転送されるゾーンが OpenShift Container Platform によって管理される Ingress ドメインである場合、アップストリームネームサーバーがドメインについて認証される必要があります。

手順

1. **default** という名前の DNS Operator オブジェクトを変更します。

```
$ oc edit dns.operator/default
```

これにより、**Server** に基づく追加のサーバー設定ブロックを使用して **dns-default** という名前の ConfigMap を作成し、更新できます。クエリーに一致するゾーンを持つサーバーがない場合、名前解決は **/etc/resolv.conf** で指定されたネームサーバーにフォールバックします。

DNS の例

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: foo-server (1)
```

```
    zones: (2)
      - foo.com
    forwardPlugin:
      upstreams: (3)
        - 1.1.1.1
        - 2.2.2.2:5353
  - name: bar-server
    zones:
      - bar.com
      - example.com
    forwardPlugin:
      upstreams:
        - 3.3.3.3
        - 4.4.4.4:5454
```

1. **name** は、**rfc6335** サービス名の構文に準拠する必要があります。
2. **zones** は、**rfc1123** の **subdomain** の定義に準拠する必要があります。クラスタードメインの **cluster.local** は、 **zones** の無効な **subdomain** です。
3. **forwardPlugin** ごとに最大 15 の **upstreams** が許可されます。

注記

servers が定義されていないか、または無効な場合、ConfigMap にはデフォルトサーバーのみが含まれます。

2. ConfigMap を表示します。

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

以前のサンプル DNS に基づく DNS ConfigMap の例

```
apiVersion: v1
data:
  Corefile: |
    foo.com:5353 {
        forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
        forward . 3.3.3.3 4.4.4.4:5454 (1)
    }
    .:5353 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            upstream
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf {
            policy sequential
```

```
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

1. **forwardPlugin** への変更により、CoreDNS デーモンセットのローリング更新がトリガーされます。

関連情報

- DNS 転送の詳細は、CoreDNS forward のドキュメント を参照してください。

4.4. DNS OPERATOR のステータス

oc describe コマンドを使用して、DNS Operator のステータスを検査し、その詳細を表示することができます。

手順

DNS Operator のステータスを表示します。

```
$ oc describe clusteroperators/dns
```

4.5. DNS OPERATOR ログ

oc logs コマンドを使用して、DNS Operator ログを表示できます。

手順

DNS Operator のログを表示します。

```
$ oc logs -n openshift-dns-operator deployment/dns-operator -c dns-operator
```

第5章 OPENSHIFT CONTAINER PLATFORM の INGRESS OPERATOR

Ingress Operator は **ingresscontroller** API を実装し、OpenShift Container Platform クラスターサービスへの外部アクセスを可能にするコンポーネントです。Operator は、1 つ以上の HAProxy ベースの ingress コントローラー をデプロイし、管理してこれを可能にします。OpenShift Container Platform **Route** および Kubernetes **Ingress** リソースを指定して、トラフィックをルーティングするために Ingress Operator を使用します。

5.1. INGRESS 設定アセット

インストールプログラムでは、**config.openshift.io** API グループの **Ingress** リソースでアセットを生成します (**cluster-ingress-02-config.yml**)。

Ingress リソースの YAML 定義

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.openshiftdemos.com
```

インストールプログラムは、このアセットを **manifests/** ディレクトリーの **cluster-ingress-02-config.yml** ファイルに保存します。この **Ingress** リソースは、Ingress のクラスター全体の設定を定義します。この Ingress 設定は、以下のように使用されます。

- Ingress Operator は、クラスター Ingress 設定のドメインを、デフォルト Ingress コントローラーのドメインとして使用します。
- OpenShift API Server Operator は、クラスター Ingress 設定からのドメインを使用します。このドメインは、明示的なホストを指定しない **Route** リソースのデフォルトホストを生成する際にも使用されます。

5.2. イメージコントローラー設定パラメーター

ingresscontrollers.operator.openshift.io リソースは以下の設定パラメーターを提供します。

パラメーター	説明

パラメーター	説明
domain	**domain** は Ingress コントローラーによって提供される DNS 名で、複数の機能を設定するために使用されます。 ● **LoadBalancerService** エンドポイント公開ストラテジーの場合、**domain** は DNS レコードを設定するために使用されます。**endpointPublishingStrategy** を参照してください。 ● 生成されるデフォルト証明書を使用する場合、証明書は **domain** およびその **subdomains** で有効です。**defaultCertificate** を参照してください。 ● この値は個別の Route ステータスに公開され、ユーザーは外部 DNS レコードのターゲット先を認識できるようにします。 **domain** 値はすべての Ingress コントローラーの中でも固有の値であり、更新できません。 空の場合、デフォルト値は **ingress.config.openshift.io/cluster .spec.domain** です。
replicas	**replicas** は Ingress コントローラーレプリカの必要な数です。設定されていない場合、デフォルト値は **2** になります。
endpointPublishingStrategy	**endpointPublishingStrategy** は Ingress コントローラーエンドポイントを他のネットワークに公開し、ロードバランサーの統合を有効にし、他のシステムへのアクセスを提供するために使用されます。 設定されていない場合、デフォルト値は **infrastructure.config.openshift.io/cluster .status.platform** をベースとします。 ● AWS: **LoadBalancerService** (外部スコープあり) ● Azure: **LoadBalancerService** (外部スコープあり) ● GCP: **LoadBalancerService** (外部スコープあり) ● Bare metal: **NodePortService** ● その他: **HostNetwork** **endpointPublishingStrategy** 値は更新できません。

パラメーター	説明
defaultCertificate	**defaultCertificate** 値は、Ingress コントローラーによって提供されるデフォルト証明書が含まれるシークレットの参照です。ルートが独自の証明書を指定しない場合、**defaultCertificate** が使用されます。 シークレットには以下のキーおよびデータが含まれる必要があります: * **tls.crt**: 証明書ファイルコンテンツ * **tls.key**: キーファイルコンテンツ 設定されていない場合、ワイルドカード証明書は自動的に生成され、使用されます。証明書は Ingress コントローラーの **domain** および **subdomains** で有効であり、生成された証明書 CA はクラスターの信頼ストアに自動的に統合されます。 使用中の証明書 (生成されるか、ユーザー指定の場合かを問わない) は OpenShift Container Platform のビルトイン OAuth サーバーに自動的に統合されます。
namespaceSelector	**namespaceSelector** は、Ingress コントローラーによってサービスされる namespace セットをフィルターするために使用されます。これはシャードの実装に役立ちます。
routeSelector	**routeSelector** は、Ingress コントローラーによって提供される Routes のセットをフィルターするために使用されます。これはシャードの実装に役立ちます。

nodePlacement	**nodePlacement** は、Ingress コントローラーのスケジュールに対する明示的な制御を有効にします。
	
	設定されていない場合は、デフォルト値が使用されます。
	
	注記
	
	nodePlacement パラメーターには、**nodeSelector** と **tolerations** の 2 つの部分が含まれます。以下に例を示します。
	
	`nodePlacement:`
	`  nodeSelector:`
	`    matchLabels:`
	`      beta.kubernetes.io/os: linux`
	`  tolerations:`
	`  - effect: NoSchedule`
	`    operator: Exists`

パラメーター	説明
tlsSecurityProfile	**tlsSecurityProfile** は、Ingress コントローラーの TLS 接続の設定を指定します。 これが設定されていない場合、デフォルト値は **apiservers.config.openshift.io/cluster** リソースをベースとして設定されます。 **Old**、**Intermediate**、および **Modern** のプロファイルタイプを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース **X.Y.Z** にデプロイされた **Intermediate** プロファイルを使用する仕様がある場合、リリース **X.Y.Z+1** へのアップグレードにより、新規のプロファイル設定が Ingress コントローラーに適用され、ロールアウトが生じる可能性があります。 Ingress コントローラーの最小の TLS バージョンは **1.1** で、最大の TLS バージョンは **1.2** です。 **重要** HAProxy Ingress コントローラーイメージは TLS **1.3** をサポートしません。**Modern** プロファイルには TLS**1.3** が必要であることから、これはサポートされません。Ingress Operator は **Modern** プロファイルを **Intermediate** に変換します。 また、Ingress Operator は TLS **1.0** の **Old** または **Custom** プロファイルを **1.1** に変換し、TLS**1.3** の **Custom** プロファイルを **1.2** に変換します。 **注記** 設定されたセキュリティープロファイルの暗号および最小 TLS バージョンが **TLSProfile** ステータスに反映されます。

パラメーター	説明
routeAdmission	**routeAdmission** は、複数の namespace での要求の許可または拒否など、新規ルート要求を処理するためのポリシーを定義します。 **namespaceOwnership** は、namespace 間でホスト名の要求を処理する方法を記述します。デフォルトは **Strict** です。 ● **Strict**: ルートが複数の namespace 間で同じホスト名を要求することを許可しません。 ● **InterNamespaceAllowed**: ルートが複数の namespace 間で同じホスト名の異なるパスを要求することを許可します。 **wildcardPolicy** は、ワイルドカードポリシーを使用するルートが Ingress コントローラーによって処理される方法を記述します。 ● **WildcardsAllowed**: ワイルドカードポリシーと共にルートが Ingress コントローラーによって許可されていることを示します。 ● **WildcardsDisallowed**: ワイルドカードポリシーの **None** を持つルートのみが Ingress コントローラーによって許可されることを示します。**wildcardPolicy** を **WildcardsAllowed** から **WildcardsDisallowed** に更新すると、ワイルドカードポリシーの **Subdomain** を持つ許可されたルートが機能を停止します。これらのルートは、Ingress コントローラーによって許可されるように **None** のワイルドカードポリシーに対して再作成される必要があります。**WildcardsDisallowed** はデフォルト設定です。

パラメーター	説明
IngressControllerLogging	**logging** はログに記録される内容および場所のパラメーターを定義します。このフィールドが空の場合、操作ログは有効になりますが、アクセスログは無効になります。 ● **access** は、クライアント要求をログに記録する方法を記述します。このフィールドが空の場合、アクセスロギングは無効になります。 　○ **destination** はログメッセージの宛先を記述します。 　　■ **type** はログの宛先のタイプです。 　　● **Container** は、ログがサイドカーコンテナーに移動することを指定します。Ingress Operator は Ingress コントローラー Pod で **logs** という名前のコンテナーを設定し、Ingress コントローラーがログをコンテナーに書き込むように設定します。管理者がこのコンテナーからログを読み取るカスタムロギングソリューションを設定することが予想されます。コンテナーログを使用すると、ログの割合がコンテナーランタイムの容量やカスタムロギングソリューションの容量を超えるとログがドロップされることがあります。 　　● **Syslog** は、ログが Syslog エンドポイントに送信されることを指定します。管理者は、Syslog メッセージを受信できるエンドポイントを指定する必要があります。管理者がカスタム Syslog インスタンスを設定していることが予想されます。 　　■ **container** は **Container** ロギング宛先タイプのパラメーターを記述します。現在、コンテナーロギングのパラメーターはないため、このフィールドは空である必要があります。 　　■ **syslog** は、**Syslog** ロギング宛先タイプのパラメーターを記述します。 　　● **address** は、ログメッセージを受信する syslog エンドポイントの IP アドレスです。 　　● **port** は、ログメッセージを受信する syslog エンドポイントの UDP ポート番号です。 　　● **facility** はログメッセージの syslog ファシリティーを指定します。このフィールドが空の場合、ファシリティーは **local1** になります。それ以外の場合、有効な syslog ファシリティー (**kern**、**user**、**mail**、**daemon**、**auth**、**syslog**、**lpr**、**news**、**uucp**、**cron**、 **auth2**、**ftp**、**ntp**、**audit**、 **alert**、**cron2**、 **local0**、**local1**、**local2**、**local3**) を指定する必要があります。**local4**、**local5**、 **local6**、または **local7**。 　○ **httpLogFormat** は、HTTP 要求のログメッセージの形式を指定します。このフィールドが空の場合、ログメッセージは実装のデフォルト HTTP ログ形式を使用します。HAProxy のデフォルトの HTTP ログ形式については、HAProxy ドキュメント を参照してください。

注記

すべてのパラメーターはオプションです。

5.2.1. Ingress コントローラーの TLS プロファイル

tlsSecurityProfile パラメーターは、TLS セキュリティープロファイルのスキーマを定義します。このオブジェクトは、TLS セキュリティー設定をオペランドに適用するために Operator によって使用されます。

TLS セキュリティープロファイルには、以下の 4 つのタイプがあります。

- **Old**
- **Intermediate**
- **Modern**
- **Custom**

Old、**Intermediate**、および **Modern** プロファイルは 推奨される設定 をベースとします。**Custom** プロファイルは、個別の TLS セキュリティープロファイルパラメーターを指定する機能を提供します。

Old プロファイル設定のサンプル

```
spec:
  tlsSecurityProfile:
    type: Old
```

Intermediate プロファイル設定のサンプル

```
spec:
  tlsSecurityProfile:
    type: Intermediate
```

Modern プロファイル設定のサンプル

```
spec:
  tlsSecurityProfile:
    type: Modern
```

Custom プロファイルは、ユーザーが定義する TLS セキュリティープロファイルです。

警告

無効な設定により問題が発生する可能性があるため、 **Custom** プロファイルを使用する際には注意してください。

Custom プロファイルのサンプル

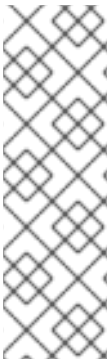
```
spec:
  tlsSecurityProfile:
    type: Custom
    custom:
      ciphers:
        - ECDHE-ECDSA-AES128-GCM-SHA256
        - ECDHE-RSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
```

5.2.2. Ingress コントローラーエンドポイントの公開ストラテジー

NodePortService エンドポイントの公開ストラテジー

NodePortService エンドポイントの公開ストラテジーは、Kubernetes NodePort サービスを使用して Ingress コントローラーを公開します。

この設定では、Ingress コントローラーのデプロイメントはコンテナーのネットワークを使用します。**NodePortService** はデプロイメントを公開するために作成されます。特定のノードポートは OpenShift Container Platform によって動的に割り当てられますが、静的ポートの割り当てをサポートするために、管理される **NodePortService** のノードポートフィールドへの変更が保持されます。



注記

Ingress Operator は、サービスの **.spec.ports[].nodePort** フィールドへの更新を無視します。

デフォルトで、ポートは自動的に割り当てられ、各種の統合用のポート割り当てにアクセスできます。ただし、既存のインフラストラクチャーと統合するために静的ポートの割り当てが必要になることがありますが、これは動的ポートに対応して簡単に再設定できない場合があります。静的ノードポートとの統合を実行するには、管理対象のサービスリソースを直接更新できます。

詳細は、**NodePort** についての Kubernetes サービスについてのドキュメント を参照してください。

HostNetwork エンドポイントの公開ストラテジー

HostNetwork エンドポイントの公開ストラテジーは、Ingress コントローラーがデプロイされるノードポートで Ingress コントローラーを公開します。

HostNetwork エンドポイント公開ストラテジーを持つ Ingress コントローラーには、ノードごとに単一の Pod レプリカのみを設定できます。**n** のレプリカを使用する場合、それらのレプリカをスケジュールできる **n** 以上のノードを使用する必要があります。各 Pod はスケジュールされるノードホストでポート **80** および **443** を要求するので、同じノードで別の Pod がそれらのポートを使用している場合、レプリカをノードにスケジュールすることはできません。

5.3. デフォルト INGRESS コントローラーの表示

Ingress Operator は、OpenShift Container Platform の中核となる機能であり、追加の設定なしに有効にできます。

すべての新規 OpenShift Container Platform インストールには、**ingresscontroller** の名前付きのデフォルトがあります。これは、追加の Ingress コントローラーで補足できます。デフォルトの **ingresscontroller** が削除される場合、Ingress Operator は 1 分以内にこれを自動的に再作成します。

手順

手順

- デフォルト Ingress コントローラーを表示します。

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/default
```

5.4. INGRESS OPERATOR ステータスの表示

Ingress Operator のステータスを表示し、検査することができます。

手順

- Ingress Operator ステータスを表示します。

```
$ oc describe clusteroperators/ingress
```

5.5. INGRESS コントローラーログの表示

Ingress コントローラーログを表示できます。

手順

- Ingress コントローラーログを表示します。

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator
```

5.6. INGRESS コントローラーステータスの表示

特定の Ingress コントローラーのステータスを表示できます。

手順

- Ingress コントローラーのステータスを表示します。

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/<name>
```

第6章 INGRESS コントローラーの設定

6.1. カスタムデフォルト証明書の設定

管理者として、 Secret リソースを作成し、**IngressController** カスタムリソース (CR) を編集して Ingress コントローラーがカスタム証明書を使用するように設定できます。

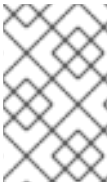
前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。ここで、証明書は信頼される認証局またはカスタム PKI で設定されたプライベートの信頼される認証局で署名されます。
- 証明書が以下の要件を満たしている必要があります。
  - 証明書が Ingress ドメインに対して有効化されている必要があります。
  - 証明書は拡張を使用して、**subjectAltName** 拡張を使用して、***.apps.ocp4.example.com** などのワイルドカードドメインを指定します。
- **IngressController** CR がなければなりません。デフォルトの CR を使用できます。

```
$ oc --namespace openshift-ingress-operator get ingresscontrollers
```

出力例

```
NAME      AGE
default   10m
```



注記

Intermediate 証明書がある場合、それらはカスタムデフォルト証明書が含まれるシークレットの **tls.crt** ファイルに組み込まれる必要があります。証明書を指定する際の順序は重要になります。サーバー証明書の後に Intermediate 証明書を一覧表示します。

手順

以下では、カスタム証明書とキーのペアが、現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提とします。**tls.crt** および **tls.key** を実際のパス名に置き換えます。さらに、 Secret リソースを作成し、これを IngressController CR で参照する際に、**custom-certs-default** を別の名前に置き換えます。

注記

このアクションにより、Ingress コントローラーはデプロイメントストラテジーを使用して再デプロイされます。

1. **tls.crt** および **tls.key** ファイルを使用して、カスタム証明書を含む Secret リソースを **openshift-ingress** namespace に作成します。

```
$ oc --namespace openshift-ingress create secret tls custom-certs-default --cert=tls.crt --
key=tls.key
```

2. IngressController CR を、新規証明書シークレットを参照するように更新します。

```
$ oc patch --type=merge --namespace openshift-ingress-operator ingresscontrollers/default \
  --patch '{"spec":{"defaultCertificate":{"name":"custom-certs-default"}}}'
```

3. 更新が正常に行われていることを確認します。

```
$ oc get --namespace openshift-ingress-operator ingresscontrollers/default \
  --output jsonpath='{.spec.defaultCertificate}'
```

出力例

```
map[name:custom-certs-default]
```

証明書シークレットの名前は、CR を更新するために使用された値に一致する必要があります。

IngressController CR が変更された後に、Ingress Operator はカスタム証明書を使用できるように Ingress コントローラーのデプロイメントを更新します。

6.2. INGRESS コントローラーのスケーリング

Ingress コントローラーは、スループットを増大させるための要件を含む、ルーティングのパフォーマンスや可用性に関する各種要件に対応するために手動でスケーリングできます。**oc** コマンドは、**IngressController** リソースのスケーリングに使用されます。以下の手順では、デフォルトの **IngressController** をスケールアップする例を示します。

手順

1. デフォルト **IngressController** の現在の利用可能なレプリカ数を表示します。

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

出力例

```
2
```

2. **oc patch** コマンドを使用して、デフォルトの **IngressController** を必要なレプリカ数にスケーリングします。以下の例では、デフォルトの **IngressController** を 3 つのレプリカにスケーリングしています。

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":{"replicas":
3}}' --type=merge
```

出力例

```
ingresscontroller.operator.openshift.io/default patched
```

3. デフォルトの **IngressController** が指定したレプリカ数にスケーリングされていることを確認します。

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

出力例

```
3
```

注記

スケーリングは、必要な数のレプリカを作成するのに時間がかかるため、すぐに実行できるアクションではありません。

6.3. INGRESS アクセスロギングの設定

アクセスログを有効にするように Ingress コントローラーを設定できます。大量のトラフィックを受信しないクラスターがある場合、サイドカーにログインできます。クラスターのトラフィックが多い場合、ロギングスタックの容量を超えないようにしたり、OpenShift Container Platform 外のロギングインフラストラクチャーと統合したりするために、ログをカスタム syslog エンドポイントに転送することができます。アクセスログの形式を指定することもできます。

コンテナーロギングは、既存の Syslog ロギングインフラストラクチャーがない場合や、Ingress コントローラーで問題を診断する際に短期間使用する場合に、低トラフィックのクラスターのアクセスログを有効にするのに役立ちます。

アクセスログがクラスターロギングスタックの容量を超える可能性があるトラフィックの多いクラスターや、ロギングソリューションが既存の Syslog ロギングインフラストラクチャーと統合する必要のある環境では、syslog が必要です。Syslog のユースケースは重複する可能性があります。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

サイドカーへの Ingress アクセスロギングを設定します。

- Ingress アクセスロギングを設定するには、**spec.logging.access.destination** を使用して宛先を指定する必要があります。サイドカーコンテナーへのロギングを指定するには、**Container spec.logging.access.destination.type** を指定する必要があります。以下の例は、コンテナー **Container** の宛先に対してログ記録する Ingress コントローラー定義です。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  endpointPublishingStrategy:
    type: NodePortService (1)
  logging:
    access:
      destination:
        type: Container
```

1. サイドカーへの Ingress アクセスロギングの設定では、**NodePortService** は必要ありません。Ingress ロギングは、すべての **endpointPublishingStrategy** と互換性があります。

- Ingress コントローラーをサイドカーに対してログを記録するように設定すると、Operator は Ingress コントローラー Pod 内に **logs** という名前のコンテナーを作成します。

```
$ oc -n openshift-ingress logs deployment.apps/router-default -c logs
```

出力例

```
2020-05-11T19:11:50.135710+00:00 router-default-57dfc6cd95-bpmk6 router-default-
57dfc6cd95-bpmk6 haproxy[108]: 174.19.21.82:39654 [11/May/2020:19:11:50.133] public
be_http:hello-openshift:hello-openshift/pod:hello-openshift:hello-openshift:10.128.2.12:8080
0/0/1/0/1 200 142 - - --NI 1/1/0/0/0 0/0 "GET / HTTP/1.1"
```

Syslog エンドポイントへの Ingress アクセスロギングを設定します。

- Ingress アクセスロギングを設定するには、**spec.logging.access.destination** を使用して宛先を指定する必要があります。Syslog エンドポイント宛先へのロギングを指定するには、**spec.logging.access.destination.type** に **Syslog** を指定する必要があります。宛先タイプが **Syslog** の場合、**spec.logging.access.destination.syslog.endpoint** を使用して宛先エンドポイントも指定する必要があります。また、**spec.logging.access.destination.syslog.facility** を使用してファシリティーを指定できます。以下の例は、**Syslog** 宛先に対してログを記録する Ingress コントローラーの定義です。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  endpointPublishingStrategy:
    type: NodePortService
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
```

注記

syslog 宛先ポートは UDP である必要があります。

特定のログ形式で Ingress アクセスロギングを設定します。

- **spec.logging.access.httpLogFormat** を指定して、ログ形式をカスタマイズできます。以下の例は、IP アドレスが 1.2.3.4 およびポート 10514 の **syslog** エンドポイントに対してログを記録する Ingress コントローラーの定義です。

```
apiVersion: operator.openshift.io/v1
```

```
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  endpointPublishingStrategy:
    type: NodePortService
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
      httpLogFormat: '%ci:%cp [%t] %ft %b/%s %B %bq %HM %HU %HV'
```

Ingress アクセスロギングを無効にします。

- Ingress アクセスロギングを無効にするには、**spec.logging** または **spec.logging.access** を空のままにします。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  endpointPublishingStrategy:
    type: NodePortService
  logging:
    access: null
```

6.4. INGRESS コントローラーのシャード化

トラフィックがクラスターに送信される主要なメカニズムとして、Ingress コントローラーまたはルーターへの要求が大きくなる可能性があります。クラスター管理者は、以下を実行するためにルートをシャード化できます。

- Ingress コントローラーまたはルーターを複数のルートに分散し、変更に対する応答を加速します。
- 特定のルートを他のルートとは異なる信頼性の保証を持つように割り当てます。
- 特定の Ingress コントローラーに異なるポリシーを定義することを許可します。
- 特定のルートのみが追加機能を使用することを許可します。
- たとえば、異なるアドレスで異なるルートを公開し、内部ユーザーおよび外部ユーザーが異なるルートを認識できるようにします。

Ingress コントローラーは、ルートラベルまたは namespace ラベルのいずれかをシャード化の方法として使用できます。

6.4.1. ルートラベルを使用した Ingress コントローラーのシャード化の設定

ルートラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーがルートセレクターによって選択される任意 namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

```
# cat router-internal.yaml
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    routeSelector:
      matchLabels:
        type: sharded
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace のルートを選択します。

6.4.2. namespace ラベルを使用した Ingress コントローラーのシャード化の設定

namespace ラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーが namespace セレクターによって選択される任意の namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

```
# cat router-internal.yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    namespaceSelector:
      matchLabels:
        type: sharded
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace セレクターによって選択される namespace のルートを選択します。

6.5. 内部ロードバランサーを使用するように INGRESS コントローラーを設定する

クラウドプラットフォームで Ingress コントローラーを作成する場合、Ingress コントローラーはデフォルトでパブリッククラウドロードバランサーによって公開されます。管理者は、内部クラウドロードバランサーを使用する Ingress コントローラーを作成できます。

警告

クラウドプロバイダーが Microsoft Azure の場合、ノードを参照するパブリックロードバランサーが少なくとも 1 つ必要です。これがない場合、すべてのノードがインターネットへの egress 接続を失います。

重要

IngressController オブジェクトの **スコープ** を変更する必要がある場合、**IngressController** オブジェクトを削除してから、これを再作成する必要があります。カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更することはできません。

実装の詳細については、Kubernetes サービスドキュメント を参照してください。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の例のように、**<name>-ingress-controller.yaml** という名前のファイルに **IngressController** カスタムリソース (CR) を作成します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> (1)
spec:
  domain: <domain> (2)
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal (3)
```

(1) **<name>** を **IngressController** オブジェクトの名前に置き換えます。

(2) コントローラーによって公開されるアプリケーションの **ドメイン** を指定します。

(3) 内部ロードバランサーを使用するために **Internal** の値を指定します。

2. 以下のコマンドを実行して、直前の手順で定義された Ingress コントローラーを作成します。

```
$ oc create -f <name>-ingress-controller.yaml (1)
```

(1) **<name>** を **IngressController** オブジェクトの名前に置き換えます。

3. オプション: 以下のコマンドを実行して Ingress コントローラーが作成されていることを確認します。

```
$ oc --all-namespaces=true get ingresscontrollers
```

6.6. クラスターを内部に配置するようにのデフォルト INGRESS コントローラーを設定する

削除や再作成を実行して、クラスターを内部に配置するように **default** Ingress コントローラーを設定できます。

> **警告**
>
> クラウドプロバイダーが Microsoft Azure の場合、ノードを参照するパブリックロードバランサーが少なくとも 1 つ必要です。これがない場合、すべてのノードがインターネットへの egress 接続を失います。

> **重要**
>
> **IngressController** オブジェクトの **スコープ** を変更する必要がある場合、**IngressController** オブジェクトを削除してから、これを再作成する必要があります。カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更することはできません。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 削除や再作成を実行して、クラスターを内部に配置するように **default** Ingress コントローラーを設定します。

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF
```

6.7. ルートの受付ポリシーの設定

管理者およびアプリケーション開発者は、同じドメイン名を持つ複数の namespace でアプリケーションを実行できます。これは、複数のチームが同じホスト名で公開されるマイクロサービスを開発する組織を対象としています。

警告

複数の namespace での要求の許可は、namespace 間の信頼のあるクラスターに対してのみ有効にする必要があります。有効にしないと、悪意のあるユーザーがホスト名を乗っ取る可能性があります。このため、デフォルトの受付ポリシーは複数の namespace 間でのホスト名の要求を許可しません。

前提条件

- クラスター管理者の権限。

手順

- 以下のコマンドを使用して、**ingresscontroller** リソース変数の **.spec.routeAdmission** フィールドを編集します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec":
{"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --type=merge
```

イメージコントローラー設定例

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
```

6.8. ワイルドカードルートの使用

HAProxy Ingress コントローラーにはワイルドカードルートのサポートがあります。Ingress Operator は **wildcardPolicy** を使用して、Ingress コントローラーの **ROUTER_ALLOW_WILDCARD_ROUTES** 環境変数を設定します。

Ingress コントローラーのデフォルトの動作では、ワイルドカードポリシーの **None** (既存の **IngressController** リソースとの後方互換性がある) を持つルートを許可します。

手順

1. ワイルドカードポリシーを設定します。

   a. 以下のコマンドを使用して **IngressController** リソースを編集します。

```
$ oc edit IngressController
```

   b. **spec** の下で、**wildcardPolicy** フィールドを **WildcardsDisallowed** または **WildcardsAllowed** に設定します。

```
spec:
  routeAdmission:
    wildcardPolicy: WildcardsDisallowed # or WildcardsAllowed
```

6.9. HTTP/2 INGRESS 接続の有効化

HAProxy で透過的なエンドツーエンド HTTP/2 接続を有効にすることができます。これにより、アプリケーションの所有者は、単一接続、ヘッダー圧縮、バイナリーストリームなど、HTTP/2 プロトコル機能を使用できます。

個別の Ingress コントローラーまたはクラスター全体について、HTTP/2 接続を有効にすることができます。

クライアントから HAProxy への接続について HTTP/2 の使用を有効にするために、ルートはカスタム証明書を指定する必要があります。デフォルトの証明書を使用するルートは HTTP/2 を使用することができません。この制限は、クライアントが同じ証明書を使用する複数の異なるルートに接続を再使用するなどの、接続の結合 (coalescing) の問題を回避するために必要です。

HAProxy からアプリケーション Pod への接続は、re-encrypt ルートのみに HTTP/2 を使用でき、edge termination ルートまたは非セキュアなルートには使用しません。この制限は、HAProxy が TLS 拡張である Application-Level Protocol Negotiation (ALPN) を使用してバックエンドで HTTP/2 の使用をネゴシエートするためにあります。そのため、エンドツーエンドの HTTP/2 はパススルーおよび re-encrypt 使用できますが、非セキュアなルートまたは edge termination ルートでは使用できません。

重要

パススルー以外のルートの場合、Ingress コントローラーはクライアントからの接続とは独立してアプリケーションへの接続をネゴシエートします。つまり、クライアントが Ingress コントローラーに接続して HTTP/1.1 をネゴシエートし、Ingress コントローラーは次にアプリケーションに接続して HTTP/2 をネゴシエートし、アプリケーションへの HTTP/2 接続を使用してクライアント HTTP/1.1 接続からの要求の転送を実行できます。Ingress コントローラーは WebSocket を HTTP/2 に転送できず、その HTTP/2 接続を WebSocket に対してアップグレードできないため、クライアントが後に HTTP/1.1 から WebSocket プロトコルに接続をアップグレードしようとすると問題が発生します。そのため、WebSocket 接続を受け入れることが意図されたアプリケーションがある場合、これは HTTP/2 プロトコルのネゴシエートを許可できないようにする必要があります。そうしないと、クライアントは WebSocket プロトコルへのアップグレードに失敗します。

手順

単一 Ingress コントローラーで HTTP/2 を有効にします。

- Ingress コントローラーで HTTP/2 を有効にするには、**oc annotate** コマンドを入力します。

```
$ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name>
ingress.operator.openshift.io/default-enable-http2=true
```

  <ingresscontroller_name> をアノテーションを付ける Ingress コントローラーの名前に置き換えます。

クラスター全体で HTTP/2 を有効にします。

- クラスター全体で HTTP/2 を有効にするには、**oc annotate** コマンドを入力します。

```
$ oc annotate ingresses.config/cluster ingress.operator.openshift.io/default-enable-http2=true
```

6.10. 関連情報

- カスタム PKI の設定

第7章 ベアメタルクラスターでの SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の使用

クラスター管理者は、クラスターで SCTP (Stream Control Transmission Protocol) を使用できます。

7.1. OPENSHIFT CONTAINER PLATFORM での SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) のサポート

クラスター管理者は、クラスターのホストで SCTP を有効にできます。Red Hat Enterprise Linux CoreOS (RHCOS) で、SCTP モジュールはデフォルトで無効にされています。

SCTP は、IP ネットワークの上部で実行される信頼できるメッセージベースのプロトコルです。

これを有効にすると、SCTP を Pod、サービス、およびネットワークポリシーでプロトコルとして使用できます。**Service** オブジェクトは、**type** パラメーターを **ClusterIP** または **NodePort** のいずれかの値に設定して定義する必要があります。

7.1.1. SCTP プロトコルを使用した設定例

protocol パラメーターを Pod またはサービスリソース定義の **SCTP** 値に設定して、Pod またはサービスを SCTP を使用するように設定できます。

以下の例では、Pod は SCTP を使用するように設定されています。

```
apiVersion: v1
kind: Pod
metadata:
  namespace: project1
  name: example-pod
spec:
  containers:
    - name: example-pod
...
      ports:
        - containerPort: 30100
          name: sctpserver
          protocol: SCTP
```

以下の例では、サービスは SCTP を使用するように設定されています。

```
apiVersion: v1
kind: Service
metadata:
  namespace: project1
  name: sctpserver
spec:
...
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30100
      targetPort: 30100
  type: ClusterIP
```

以下の例では、**NetworkPolicy** オブジェクトは、特定のラベルの付いた Pod からポート **80** の SCTP ネットワークトラフィックに適用するように設定されます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-sctp-on-http
spec:
  podSelector:
    matchLabels:
      role: web
  ingress:
  - ports:
    - protocol: SCTP
      port: 80
```

7.2. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の有効化

クラスター管理者は、クラスターのワーカーノードでブラックリストに指定した SCTP カーネルモジュールを読み込み、有効にできます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 以下の YAML 定義が含まれる **load-sctp-module.yaml** という名前のファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: load-sctp-module
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
          filesystem: root
          mode: 420
          path: /etc/modules-load.d/sctp-load.conf
```

2. **MachineConfig** オブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc create -f load-sctp-module.yaml
```

3. オプション: MachineConfig Operator が設定変更を適用している間にノードのステータスを確認するには、以下のコマンドを入力します。ノードのステータスが **Ready** に移行すると、設定の更新が適用されます。

```
$ oc get nodes
```

7.3. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) が有効になっていることの確認

SCTP がクラスターで機能することを確認するには、SCTP トラフィックをリッスンするアプリケーションで Pod を作成し、これをサービスに関連付け、公開されたサービスに接続します。

前提条件

- クラスターからインターネットにアクセスし、**nc** パッケージをインストールすること。
- OpenShift CLI (**oc**) をインストールします。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. SCTP リスナーを起動する Pod を作成します。

   a. 以下の YAML で Pod を定義する **sctp-server.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpserver
  labels:
    app: sctpserver
spec:
  containers:
    - name: sctpserver
      image: registry.access.redhat.com/ubi8/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
      ports:
        - containerPort: 30102
          name: sctpserver
          protocol: SCTP
```

   b. 以下のコマンドを入力して Pod を作成します。

```
$ oc create -f sctp-server.yaml
```

2. SCTP リスナー Pod のサービスを作成します。

a. 以下の YAML でサービスを定義する **sctp-service.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: sctpservice
  labels:
    app: sctpserver
spec:
  type: NodePort
  selector:
    app: sctpserver
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30102
      targetPort: 30102
```

b. サービスを作成するには、以下のコマンドを入力します。

```
$ oc create -f sctp-service.yaml
```

3. SCTP クライアントの Pod を作成します。

a. 以下の YAML で **sctp-client.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpclient
  labels:
    app: sctpclient
spec:
  containers:
    - name: sctpclient
      image: registry.access.redhat.com/ubi8/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
```

b. **Pod** オブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc apply -f sctp-client.yaml
```

4. サーバーで SCTP リスナーを実行します。

a. サーバー Pod に接続するには、以下のコマンドを入力します。

```
$ oc rsh sctpserver
```

b. SCTP リスナーを起動するには、以下のコマンドを入力します。

```
$ nc -l 30102 --sctp
```

5. サーバーの SCTP リスナーに接続します。

   a. ターミナルプログラムで新規のターミナルウィンドウまたはタブを開きます。

   b. **sctpservice** サービスの IP アドレスを取得します。以下のコマンドを入力します。

   ```
   $ oc get services sctpservice -o go-template='{{.spec.clusterIP}}{{"\n"}}'
   ```

   c. クライアント Pod に接続するには、以下のコマンドを入力します。

   ```
   $ oc rsh sctpclient
   ```

   d. SCTP クライアントを起動するには、以下のコマンドを入力します。**<cluster_IP>** を **sctpservice** サービスのクラスター IP アドレスに置き換えます。

   ```
   # nc <cluster_IP> 30102 --sctp
   ```

第8章 PTP ハードウェアの設定

重要

Precision Time Protocol (PTP) ハードウェアはテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

8.1. PTP ハードウェアについて

OpenShift Container Platform には、ノード上で Precision Time Protocol (PTP) ハードウェアを使用する機能が含まれます。linuxptp サービスは、PTP 対応ハードウェアを搭載したクラスターで設定できます。

注記

PTP Operator は、ベアメタルインフラストラクチャーでのみプロビジョニングされるクラスターの PTP 対応デバイスと連携します。

PTP Operator をデプロイし、OpenShift Container Platform コンソールを使用して PTP をインストールできます。PTP Operator は、linuxptp サービスを作成し、管理します。Operator は以下の機能を提供します。

- クラスター内の PTP 対応デバイスの検出。
- linuxptp サービスの設定の管理。

8.2. PTP ネットワークデバイスの自動検出

PTP Operator は **NodePtpDevice.ptp.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。PTP Operator はクラスターで、各ノードの PTP 対応ネットワークデバイスを検索します。Operator は、互換性のある PTP デバイスを提供する各ノードの **NodePtpDevice** カスタムリソース (CR) オブジェクトを作成し、更新します。

1 つの CR がノードごとに作成され、ノードと同じ名前を共有します。**.status.devices** 一覧は、ノード上の PTP デバイスについての情報を提供します。

以下は、PTP Operator によって作成される **NodePtpDevice** CR の例です。

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2019-11-15T08:57:11Z"
  generation: 1
  name: dev-worker-0 (1)
  namespace: openshift-ptp (2)
  resourceVersion: "487462"
```

```
  selfLink: /apis/ptp.openshift.io/v1/namespaces/openshift-ptp/nodeptpdevices/dev-worker-0
  uid: 08d133f7-aae2-403f-84ad-1fe624e5ab3f
spec: {}
status:
  devices: 3
  - name: eno1
  - name: eno2
  - name: ens787f0
  - name: ens787f1
  - name: ens801f0
  - name: ens801f1
  - name: ens802f0
  - name: ens802f1
  - name: ens803
```

1. **name** パラメーターの値はノードの名前と同じです。
2. CR は PTP Operator によって **openshift-ptp** namespace に作成されます。
3. **devices** コレクションには、ノード上の Operator によって検出されるすべての PTP 対応デバイスの一覧が含まれます。

8.3. PTP OPERATOR のインストール

クラスター管理者は、OpenShift Container Platform CLI または Web コンソールを使用して PTP Operator をインストールできます。

8.3.1. CLI: PTP Operator のインストール

クラスター管理者は、CLI を使用して Operator をインストールできます。

前提条件

- PTP に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. PTP Operator の namespace を作成するには、以下のコマンドを入力します。

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  labels:
    name: openshift-ptp
    openshift.io/cluster-monitoring: "true"
EOF
```

2. Operator の Operator グループを作成するには、以下のコマンドを入力します。

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
  - openshift-ptp
EOF
```

3. PTP Operator にサブスクライブします。

   a. 以下のコマンドを実行して、OpenShift Container Platform のメジャーおよびマイナーバージョンを環境変数として設定します。これは次の手順で **channel** の値として使用されます。

```
$ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
    grep -o '[0-9]*[.][0-9]*' | head -1)
```

   b. PTP Operator のサブスクリプションを作成するには、以下のコマンドを入力します。

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "${OC_VERSION}"
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Operator がインストールされていることを確認するには、以下のコマンドを入力します。

```
$ oc get csv -n openshift-ptp \
  -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

出力例

```
Name                                   Phase
ptp-operator.4.4.0-202006160135            Succeeded
```

8.3.2. Web コンソール: PTP Operator のインストール

クラスター管理者は、Web コンソールを使用して Operator をインストールできます。

注記

先のセクションで説明されているように namespace および Operator グループを作成する必要があります。

手順

1. OpenShift Container Platform Web コンソールを使用して PTP Operator をインストールします。
   a. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
   b. 利用可能な Operator の一覧から **PTP Operator** を選択してから **Install** をクリックします。
   c. **Install Operator** ページの **A specific namespace on the cluster**の下で **openshift-ptp** を選択します。次に、**Install** をクリックします。
2. オプション: PTP Operator が正常にインストールされていることを確認します。
   a. **Operators → Installed Operators** ページに切り替えます。
   b. **PTP Operator** が **Status** が **InstallSucceeded** の状態で **openshift-ptp** プロジェクトに一覧表示されていることを確認します。

   注記

   インストール時に、 Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

   Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

   - **Operators → Installed Operators** ページに移動し、**Operator Subscriptions** および **Install Plans** タブで **Status** にエラーがあるかどうかを検査します。
   - **Workloads → Pods** ページに移動し、**openshift-ptp** プロジェクトで Pod のログを確認します。

8.4. LINUXPTP サービスの設定

PTP Operator は **PtpConfig.ptp.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。**PtpConfig** カスタムリソース (CR) オブジェクトを作成して、Linuxptp サービス (ptp4l、phc2sys) を設定できます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- PTP Operator がインストールされていること。

手順

1. 以下の **PtpConfig** CR を作成してから、YAML を **<name>-ptp-config.yaml** ファイルに保存します。**<name>** をこの設定の名前に置き換えます。

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <name> (1)
  namespace: openshift-ptp (2)
spec:
  profile: (3)
  - name: "profile1" (4)
    interface: "ens787f1" (5)
    ptp4lOpts: "-s -2" (6)
    phc2sysOpts: "-a -r" (7)
  recommend: (8)
  - profile: "profile1" (9)
    priority: 10 (10)
    match: (11)
    - nodeLabel: "node-role.kubernetes.io/worker" (12)
      nodeName: "dev-worker-0" (13)
```

1. **PtpConfig** CR の名前を指定します。
2. PTP Operator がインストールされている namespace を指定します。
3. 1 つ以上の **profile** オブジェクトの配列を指定します。
4. プロファイルオブジェクトを一意に識別するために使用されるプロファイルオブジェクトの名前を指定します。
5. **ptp4l** サービスで使用するネットワークインターフェイス名を指定します (例: **ens787f1**)。
6. **ptp4l** サービスのシステム設定オプション (例: **-s -2**) を指定します。これには、インターフェイス名 **-i <interface>** およびサービス設定ファイル **-f /etc/ptp4l.conf** を含めないでください。これらは自動的に追加されます。
7. **phc2sys** サービスのシステム設定オプション (例: **-a -r**) を指定します。
8. **profile** がノードに適用される方法を定義する 1 つ以上の **recommend** オブジェクトの配列を指定します。
9. **profile** セクションに定義される **profile** オブジェクト名を指定します。
10. **0** から **99** までの整数値で **priority** を指定します。数値が大きいほど優先度が低くなるため、**99** の優先度は **10** よりも低くなります。ノードが **match** フィールドで定義されるルールに基づいて複数のプロファイルに一致する場合、優先順位の高い プロファイルがそのノードに適用されます。
11. **match** ルールを、**nodeLabel** または **nodeName** で指定します。
12. **nodeLabel** を、ノードオブジェクトの **node.Labels** の **key** で指定します。
13. **nodeName** をノードオブジェクトの **node.Name** で指定します。

2. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f <filename> (1)
```

(1) **<filename>** を、先の手順で作成したファイルの名前に置き換えます。

3. オプション: **PtpConfig** プロファイルが、 **nodeLabel** または **nodeName** に一致するノードに適用されることを確認します。

```
$ oc get pods -n openshift-ptp -o wide
```

出力例

```
NAME                          READY   STATUS    RESTARTS   AGE   IP             NODE
NOMINATED NODE   READINESS GATES
linuxptp-daemon-4xkbb          1/1     Running   0          43m   192.168.111.15   dev-worker-0
<none>           <none>
linuxptp-daemon-tdspf          1/1     Running   0          43m   192.168.111.11   dev-master-0
<none>           <none>
ptp-operator-657bbb64c8-2f8sj   1/1     Running   0          43m   10.128.0.116     dev-master-0
<none>           <none>

$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1 (1)
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1 (2)
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -s -2 (3)
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r (4)
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
I1115 09:41:18.117934 4143292 daemon.go:186] Starting phc2sys...
I1115 09:41:18.117985 4143292 daemon.go:187] phc2sys cmd: &{Path:/usr/sbin/phc2sys
Args:[/usr/sbin/phc2sys -a -r] Env:[] Dir: Stdin:<nil> Stdout:<nil> Stderr:<nil> ExtraFiles:[]
SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil> lookPathErr:<nil> finished:false
childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[] errch:<nil> waitDone:<nil>}
I1115 09:41:19.118175 4143292 daemon.go:186] Starting ptp4l...
I1115 09:41:19.118209 4143292 daemon.go:187] ptp4l cmd: &{Path:/usr/sbin/ptp4l Args:
[/usr/sbin/ptp4l -m -f /etc/ptp4l.conf -i ens787f1 -s -2] Env:[] Dir: Stdin:<nil> Stdout:<nil>
Stderr:<nil> ExtraFiles:[] SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil>
lookPathErr:<nil> finished:false childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[]
errch:<nil> waitDone:<nil>}
ptp4l[102189.864]: selected /dev/ptp5 as PTP clock
ptp4l[102189.886]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[102189.886]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
```

(1) **Profile Name** は、ノード **dev-worker-0** に適用される名前です。

(2) **Interface** は、**profile1** インターフェイスフィールドに指定される PTP デバイスです。**ptp4l** サービスはこのインターフェイスで実行されます。

(3) **PTP4lOpts** は、**profile1** Ptp4lOpts フィールドで指定される ptp4l sysconfig オプションです。

4 **phc2sysOpts** は、**profile1** phc2sysOpts フィールドで指定される phc2sys sysconfig オプションです。

第9章 ネットワークポリシー

9.1. ネットワークポリシーについて

クラスター管理者は、トラフィックをクラスター内の Pod に制限するネットワークポリシーを定義できます。

9.1.1. ネットワークポリシーについて

Kubernetes ネットワークポリシーをサポートする Kubernetes Container Network Interface (CNI) プラグインを使用するクラスターでは、ネットワークの分離は **NetworkPolicy** オブジェクトによって完全に制御されます。OpenShift Container Platform 4.5 では、OpenShift SDN はデフォルトのネットワーク分離モードでのネットワークポリシーの使用をサポートしています。

注記

OpenShift SDN クラスターネットワークプロバイダーを使用する場合、ネットワークポリシーについて、以下の制限が適用されます。

- **egress** フィールドで指定される egress ネットワークポリシーはサポートされていません。
- IPBlock はネットワークポリシーでサポートされますが、**except** 句はサポートしません。**except** 句を含む IPBlock セクションのあるポリシーを作成する場合、SDN Pod は警告をログに記録し、そのポリシーの IPBlock セクション全体は無視されます。

警告

ネットワークポリシーは、ホストのネットワーク namespace には適用されません。ホストネットワークが有効にされている Pod はネットワークポリシールールによる影響を受けません。

デフォルトで、プロジェクトのすべての Pod は他の Pod およびネットワークのエンドポイントからアクセスできます。プロジェクトで 1 つ以上の Pod を分離するには、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。プロジェクト管理者は独自のプロジェクト内で **NetworkPolicy** オブジェクトの作成および削除を実行できます。

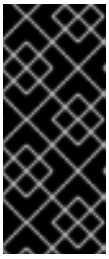
Pod が 1 つ以上の **NetworkPolicy** オブジェクトのセレクターで一致する場合、Pod はそれらの 1 つ以上の **NetworkPolicy** オブジェクトで許可される接続のみを受け入れます。**NetworkPolicy** オブジェクトによって選択されていない Pod は完全にアクセス可能です。

以下のサンプル **NetworkPolicy** オブジェクトは、複数の異なるシナリオをサポートすることを示しています。

- すべてのトラフィックを拒否します。
  プロジェクトに deny by default (デフォルトで拒否) を実行させるには、すべての Pod に一致するが、トラフィックを一切許可しない **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
  ingress: []
```

- OpenShift Container Platform Ingress コントローラーからの接続のみを許可します。
  プロジェクトで OpenShift Container Platform Ingress コントローラーからの接続のみを許可するには、以下の **NetworkPolicy** オブジェクトを追加します。



重要

OVN-Kubernetes ネットワークプロバイダープラグインの場合、Ingress コントローラーが **HostNetwork** エンドポイント公開ストラテジーを使用するように設定されている場合、Ingress トラフィックが許可され、他のすべてのトラフィックが拒否されるようにネットワークポリシーを適用するための方法はサポートされていません。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
```

Ingress コントローラーが **endpointPublishingStrategy: HostNetwork** で設定されている場合、Ingress コントローラー Pod はホストネットワーク上で実行されます。ホストネットワーク上で実行されている場合、Ingress コントローラーからのトラフィックに **netid:0** Virtual Network ID (VNID) が割り当てられます。Ingress Operator に関連付けられる namespace の **netid** は異なるため、**allow-from-openshift-ingress** ネットワークポリシーの **matchLabel** は **default** Ingress コントローラーからのトラフィックに一致しません。OpenShift SDN では、**default** namespace に **netid:0** VNID が割り当てられるため、**default** namespace に **network.openshift.io/policy-group: ingress** でラベルを付けて、**default** Ingress コントローラーからのトラフィックを許可できます。

- プロジェクト内の Pod からの接続のみを受け入れます。
  Pod が同じプロジェクト内の他の Pod からの接続を受け入れるが、他のプロジェクトの Pod からの接続を拒否するように設定するには、以下の **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
```

```
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

- Pod ラベルに基づいて HTTP および HTTPS トラフィックのみを許可します。
  特定のラベル (以下の例の **role=frontend**) の付いた Pod への HTTP および HTTPS アクセスのみを有効にするには、以下と同様の **NetworkPolicy** オブジェクトを追加します。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

- namespace および Pod セレクターの両方を使用して接続を受け入れます。
  namespace と Pod セレクターを組み合わせてネットワークトラフィックのマッチングをするには、以下と同様の **NetworkPolicy** オブジェクトを使用できます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
    matchLabels:
      name: test-pods
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            project: project_name
        podSelector:
          matchLabels:
            name: test-pods
```

NetworkPolicy オブジェクトは加算されるものです。 つまり、複数の **NetworkPolicy** オブジェクトを組み合わせて複雑なネットワーク要件を満すことができます。

たとえば、先の例で定義された **NetworkPolicy** オブジェクトの場合、同じプロジェト内に **allow-same-namespace** と **allow-http-and-https** ポリシーの両方を定義することができます。これにより、ラベル **role=frontend** の付いた Pod は各ポリシーで許可されるすべての接続を受け入れます。つまり、同じ namespace の Pod からのすべてのポート、およびすべての namespace の Pod からのポート **80** および **443** での接続を受け入れます。

9.1.2. ネットワークポリシーの最適化

ネットワークポリシーを使用して、namespace 内でラベルで相互に区別される Pod を分離します。

注記

ネットワークポリシールールを効果的に使用するためのガイドラインは、OpenShift SDN クラスターネットワークプロバイダーのみに適用されます。

NetworkPolicy オブジェクトを単一 namespace 内の多数の個別 Pod に適用することは効率的ではありません。Pod ラベルは IP レベルには存在しないため、ネットワークポリシーは、**podSelector** で選択されるすべての Pod 間のすべてのリンクについての別個の Open vSwitch (OVS) フロールールを生成します。

たとえば、仕様の **podSelector** および **NetworkPolicy** オブジェクト内の ingress **podSelector** のそれぞれが 200 Pod に一致する場合、40,000 (200*200) OVS フロールールが生成されます。これにより、ノードの速度が低下する可能性があります。

ネットワークポリシーを設計する場合は、以下のガイドラインを参照してください。

- namespace を使用して分離する必要のある Pod のグループを組み込み、OVS フロールールの数を減らします。
  namespace 全体を選択する **NetworkPolicy** オブジェクトは、**namespaceSelectors** または空の **podSelectors** を使用して、namespace の VXLAN 仮想ネットワーク ID に一致する単一の OVS フロールールのみを生成します。

- 分離する必要のない Pod は元の namespace に維持し、分離する必要のある Pod は 1 つ以上の異なる namespace に移します。

- 追加のターゲット設定された namespace 間のネットワークポリシーを作成し、分離された Pod から許可する必要のある特定のトラフィックを可能にします。

9.1.3. 次のステップ

- ネットワークポリシーの作成

- オプション: デフォルトネットワークポリシーの定義

9.1.4. 関連情報

- プロジェクトおよび namespace

- マルチテナントネットワークポリシーの設定

- NetworkPolicy API

9.2. ネットワークポリシーの作成

admin ロールを持つユーザーは、namespace のネットワークポリシーを作成できます。

9.2.1. ネットワークポリシーの作成

クラスターの namespace に許可される Ingress または egress ネットワークトラフィックを記述する詳細なルールを定義するには、ネットワークポリシーを作成できます。

注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の namespace でネットワークポリシーを作成できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. ポリシールールを作成します。

   a. **<policy_name>.yaml** ファイルを作成します。

   ```
   $ touch <policy_name>.yaml
   ```

   ここでは、以下のようになります。

   <policy_name>
   ネットワークポリシーファイル名を指定します。

   b. 作成したばかりのファイルで、以下の例のようなネットワークポリシーを定義します。

   すべての namespace のすべての Pod から ingress を拒否します。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: deny-by-default
   spec:
     podSelector:
     ingress: []
   ```

   同じ namespace のすべての Pod から ingress を許可します。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: allow-same-namespace
   spec:
     podSelector:
     ingress:
     - from:
       - podSelector: {}
   ```

2. ネットワークポリシーオブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーファイル名を指定します。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

```
networkpolicy "default-deny" created
```

9.2.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

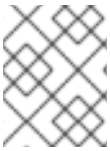
1. NetworkPolicy オブジェクトの **name**。
2. ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
3. ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。
4. トラフィックを受け入れる 1 つ以上の宛先の一覧。

9.3. ネットワークポリシーの表示

admin ロールを持つユーザーは、namespace のネットワークポリシーを表示できます。

9.3.1. ネットワークポリシーの表示

namespace のネットワークポリシーを検査できます。



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内のネットワークポリシーを表示できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

手順

- namespace のネットワークポリシーを一覧表示します。
  - namespace で定義された **NetworkPolicy** オブジェクトを表示するには、以下のコマンドを実行します。

    ```
    $ oc get networkpolicy
    ```

  - オプション: 特定のネットワークポリシーを検査するには、以下のコマンドを入力します。

    ```
    $ oc describe networkpolicy <policy_name> -n <namespace>
    ```

    ここでは、以下のようになります。

    <policy_name>
    検査するネットワークポリシーの名前を指定します。

    <namespace>
    オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

    以下に例を示します。

    ```
    $ oc describe networkpolicy allow-same-namespace
    ```

    oc describe コマンドの出力

    ```
    Name:         allow-same-namespace
    Namespace:    ns1
    Created on:   2021-05-24 22:28:56 -0400 EDT
    Labels:       <none>
    Annotations:  <none>
    Spec:
      PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
      Allowing ingress traffic:
        To Port: <any> (traffic allowed to all ports)
        From:
    ```

```
    PodSelector: <none>
Not affecting egress traffic
Policy Types: Ingress
```

9.3.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

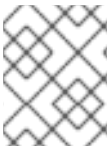
1. NetworkPolicy オブジェクトの **name**。
2. ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
3. ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。
4. トラフィックを受け入れる 1 つ以上の宛先の一覧。

9.4. ネットワークポリシーの編集

admin ロールを持つユーザーは、namespace の既存のネットワークポリシーを編集できます。

9.4.1. ネットワークポリシーの編集

namespace のネットワークポリシーを編集できます。



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の namespace でネットワークポリシーを編集できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

手順

1. オプション: namespace のネットワークポリシーオブジェクトを一覧表示するには、以下のコマンドを入力します。

```
$ oc get networkpolicy -n <namespace>
```

ここでは、以下のようになります。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

2. **NetworkPolicy** オブジェクトを編集します。
   - ネットワークポリシーの定義をファイルに保存した場合は、ファイルを編集して必要な変更を加えてから、以下のコマンドを入力します。

```
$ oc apply -n <namespace> -f <policy_file>.yaml
```

ここでは、以下のようになります。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

<policy_file>
ネットワークポリシーを含むファイルの名前を指定します。

   - **NetworkPolicy** オブジェクトを直接更新する必要がある場合、以下のコマンドを入力できます。

```
$ oc edit networkpolicy <policy_name> -n <namespace>
```

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーの名前を指定します。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

3. **NetworkPolicy** オブジェクトが更新されていることを確認します。

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーの名前を指定します。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

9.4.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

1. NetworkPolicy オブジェクトの **name**。
2. ポリシーが適用される Pod を記述するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
3. ポリシーオブジェクトが Ingress トラフィックを許可する Pod に一致するセレクター。セレクターはすべてのプロジェクトの Pod に一致します。
4. トラフィックを受け入れる 1 つ以上の宛先の一覧。

9.4.3. 関連情報

- ネットワークポリシーの作成

9.5. ネットワークポリシーの削除

admin ロールを持つユーザーは、namespace からネットワークポリシーを削除できます。

9.5.1. ネットワークポリシーの削除

namespace のネットワークポリシーを削除できます。

注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内のネットワークポリシーを削除できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

手順

- **NetworkPolicy** オブジェクトを削除するには、以下のコマンドを入力します。

```
$ oc delete networkpolicy <policy_name> -n <namespace>
```

ここでは、以下のようになります。

<policy_name>
ネットワークポリシーの名前を指定します。

<namespace>
オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

```
networkpolicy.networking.k8s.io/allow-same-namespace deleted
```

9.6. プロジェクトのデフォルトネットワークポリシーの定義

クラスター管理者は、新規プロジェクトの作成時にネットワークポリシーを自動的に含めるように新規プロジェクトテンプレートを変更できます。新規プロジェクトのカスタマイズされたテンプレートがまだない場合には、まずテンプレートを作成する必要があります。

9.6.1. 新規プロジェクトのテンプレートの変更

クラスター管理者は、デフォルトのプロジェクトテンプレートを変更し、新規プロジェクトをカスタム要件に基づいて作成することができます。

独自のカスタムプロジェクトテンプレートを作成するには、以下を実行します。

手順

1. **cluster-admin** 権限を持つユーザーとしてログインしている。

2. デフォルトのプロジェクトテンプレートを生成します。

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. オブジェクトを追加するか、または既存オブジェクトを変更することにより、テキストエディターで生成される **template.yaml** ファイルを変更します。

4. プロジェクトテンプレートは、**openshift-config** namespace に作成される必要があります。変更したテンプレートを読み込みます。

```
$ oc create -f template.yaml -n openshift-config
```

5. Web コンソールまたは CLI を使用し、プロジェクト設定リソースを編集します。

   - Web コンソールの使用

     i. **Administration → Cluster Settings** ページに移動します。

     ii. **Global Configuration** をクリックし、すべての設定リソースを表示します。

     iii. **Project** のエントリーを見つけ、**Edit YAML** をクリックします。

   - CLI の使用

     i. **project.config.openshift.io/cluster** リソースを編集します。

```
$ oc edit project.config.openshift.io/cluster
```

6. **spec** セクションを、**projectRequestTemplate** および **name** パラメーターを組み込むように更新し、アップロードされたプロジェクトテンプレートの名前を設定します。デフォルト名は **project-request** です。

   カスタムプロジェクトテンプレートを含むプロジェクト設定リソース

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7. 変更を保存した後、変更が正常に適用されたことを確認するために、新しいプロジェクトを作成します。

9.6.2. 新規プロジェクトへのネットワークポリシーの追加

クラスター管理者は、ネットワークポリシーを新規プロジェクトのデフォルトテンプレートに追加できます。OpenShift Container Platform は、プロジェクトのテンプレートに指定されたすべての **NetworkPolicy** オブジェクトを自動的に作成します。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするデフォルトの CNI ネットワークプロバイダーを使用する (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインすること。
- 新規プロジェクトのカスタムデフォルトプロジェクトテンプレートを作成していること。

手順

1. 以下のコマンドを実行して、新規プロジェクトのデフォルトテンプレートを編集します。

```
$ oc edit template <project_template> -n openshift-config
```

<project_template> を、クラスターに設定したデフォルトテンプレートの名前に置き換えます。デフォルトのテンプレート名は **project-request** です。

2. テンプレートでは、各 **NetworkPolicy** オブジェクトを要素として **objects** パラメーターに追加します。**objects** パラメーターは、1 つ以上のオブジェクトのコレクションを受け入れます。以下の例では、**objects** パラメーターのコレクションにいくつかの **NetworkPolicy** オブジェクトが含まれます。

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector:
    ingress:
    - from:
      - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
    podSelector: {}
    policyTypes:
    - Ingress
...
```

3. オプション: 以下のコマンドを実行して、新規プロジェクトを作成し、ネットワークポリシーオブジェクトが正常に作成されることを確認します。

   a. 新規プロジェクトを作成します。

```
$ oc new-project <project> (1)
```

1. **<project>** を、作成しているプロジェクトの名前に置き換えます。

b. 新規プロジェクトテンプレートのネットワークポリシーオブジェクトが新規プロジェクトに存在することを確認します。

```
$ oc get networkpolicy
NAME                          POD-SELECTOR   AGE
allow-from-openshift-ingress   <none>        7s
allow-from-same-namespace      <none>        7s
```

9.7. ネットワークポリシーを使用したマルチテナント分離の設定

クラスター管理者は、マルチテナントネットワークの分離を実行するようにネットワークポリシーを設定できます。

注記

OpenShift SDN クラスターネットワークプロバイダーを使用している場合、本セクションで説明されているようにネットワークポリシーを設定すると、マルチテナントモードと同様のネットワーク分離が行われますが、ネットワークポリシーモードが設定されます。

9.7.1. ネットワークポリシーを使用したマルチテナント分離の設定

他のプロジェクト namespace の Pod およびサービスから分離できるようにプロジェクトを設定できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

1. 以下の **NetworkPolicy** オブジェクトを作成します。

   a. **allow-from-openshift-ingress** という名前のポリシー:

重要

OVN-Kubernetes ネットワークプロバイダープラグインの場合、Ingress コントローラーが **HostNetwork** エンドポイント公開ストラテジーを使用するように設定されている場合、Ingress トラフィックが許可され、他のすべてのトラフィックが拒否されるようにネットワークポリシーを適用するための方法はサポートされていません。

```
$ cat << EOF| oc create -f -
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

b. **allow-from-openshift-monitoring** という名前のポリシー。

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

c. **allow-same-namespace** という名前のポリシー:

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

2. **default** Ingress コントローラー設定に **spec.endpointPublishingStrategy: HostNetwork** の値が設定されている場合、ラベルを **default** OpenShift Container Platform namespace に適用し、Ingress コントローラーとプロジェクト間のネットワークトラフィックを許可する必要があります。

   a. **default** Ingress コントローラーが **HostNetwork** エンドポイント公開ストラテジーを使用するかどうかを判別します。

```
$ oc get --namespace openshift-ingress-operator ingresscontrollers/default \
  --output jsonpath='{.status.endpointPublishingStrategy.type}'
```

b. 直前のコマンドによりエンドポイント公開ストラテジーが **HostNetwork** として報告される場合には、**default** namespace にラベルを設定します。

```
$ oc label namespace default 'network.openshift.io/policy-group=ingress'
```

3. 以下のコマンドを実行し、**NetworkPolicy** オブジェクトが現在のプロジェクトに存在することを確認します。

```
$ oc get networkpolicy <policy-name> -o yaml
```

以下の例では、**allow-from-openshift-ingress NetworkPolicy** オブジェクトが表示されています。

```
$ oc get -n project1 networkpolicy allow-from-openshift-ingress -o yaml
```

出力例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
  namespace: project1
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
  - Ingress
```

9.7.2. 次のステップ

- デフォルトのネットワークポリシーの定義

9.7.3. 関連情報

- OpenShift SDN ネットワーク分離モード

第10章 複数ネットワーク

10.1. 複数ネットワークについて

Kubernetes では、コンテナーネットワークは Container Network Interface (CNI) を実装するネットワークプラグインに委任されます。

OpenShift Container Platform は、Multus CNI プラグインを使用して CNI プラグインのチェーンを許可します。クラスターのインストール時に、**デフォルト** の Pod ネットワークを設定します。デフォルトのネットワークは、クラスターのすべての通常のネットワークトラフィックを処理します。利用可能な CNI プラグインに基づいて **追加のネットワーク** を定義し、1 つまたは複数のネットワークを Pod に割り当てることができます。必要に応じて、クラスターの複数のネットワークを追加で定義することができます。これは、スイッチまたはルーティングなどのネットワーク機能を提供する Pod を設定する場合に柔軟性を実現します。

10.1.1. 追加ネットワークの使用シナリオ

データプレーンとコントロールプレーンの分離など、ネットワークの分離が必要な状況で追加のネットワークを使用することができます。トラフィックの分離は、以下のようなパフォーマンスおよびセキュリティー関連の理由で必要になります。

パフォーマンス

各プレーンのトラフィック量を管理するために、2 つの異なるプレーンにトラフィックを送信できます。

セキュリティー

機密トラフィックは、セキュリティー上の考慮に基づいて管理されているネットワークに送信でき、テナントまたはカスタマー間で共有できないプライベートを分離することができます。

クラスターのすべての Pod はクラスター全体のデフォルトネットワークを依然として使用し、クラスター全体での接続性を維持します。すべての Pod には、クラスター全体の Pod ネットワークに割り当てられる **eth0** インターフェイスがあります。Pod のインターフェイスは、**oc exec -it <pod_name> -- ip a** コマンドを使用して表示できます。Multus CNI を使用するネットワークを追加する場合、それらの名前は **net1**、**net2**、...、 **netN** になります。

追加のネットワークを Pod に割り当てるには、インターフェイスの割り当て方法を定義する設定を作成する必要があります。それぞれのインターフェイスは、**NetworkAttachmentDefinition** カスタムリソース (CR) を使用して指定します。これらの CR のそれぞれにある CNI 設定は、インターフェイスの作成方法を定義します。

10.1.2. OpenShift Container Platform の追加ネットワーク

OpenShift Container Platform は、クラスターに追加のネットワークを作成するために使用する以下の CNI プラグインを提供します。

- **bridge**: ブリッジベースの追加ネットワークを作成する ことで、同じホストにある Pod が相互に、かつホストと通信できます。
- **host-device**: ホストデバイスの追加ネットワークを設定する ことで、Pod がホストシステム上の物理イーサネットネットワークデバイスにアクセスすることができます。
- **ipvlan**: ipvlan ベースの追加ネットワークを設定する ことで、macvlan ベースの追加ネットワークと同様に、ホスト上の Pod が他のホストやそれらのホストの Pod と通信できます。macvlan ベースの追加のネットワークとは異なり、各 Pod は親の物理ネットワークインターフェイスと同じ MAC アドレスを共有します。

- **macvlan**: macvlan ベースの追加ネットワークを作成することで、ホスト上の Pod が物理ネットワークインターフェイスを使用して他のホストやそれらのホストの Pod と通信できます。macvlan ベースの追加ネットワークに割り当てられる各 Pod には固有の MAC アドレスが割り当てられます。
  macvlan の追加ネットワークは 2 つの方法で設定できます。
  - 基本的なカスタマイズを使用した macvlan ベースの追加ネットワークの設定
  - macvlan ベースの追加ネットワークの設定
- **SR-IOV**: SR-IOV ベースの追加ネットワークを設定する ことで、Pod を ホストシステム上の SR-IOV 対応ハードウェアの仮想機能 (VF) インターフェイスに割り当てることができます。

10.2. POD の追加のネットワークへの割り当て

クラスターユーザーとして、Pod を追加のネットワークに割り当てることができます。

10.2.1. Pod の追加ネットワークへの追加

Pod を追加のネットワークに追加できます。Pod は、デフォルトネットワークで通常のクラスター関連のネットワークトラフィックを継続的に送信します。

Pod が作成されると、追加のネットワークが割り当てられます。ただし、Pod がすでに存在する場合は、追加のネットワークをこれに割り当てることはできません。

Pod が追加ネットワークと同じ namespace にあること。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- クラスターにログインする。

手順

1. アノテーションを **Pod** オブジェクトに追加します。以下のアノテーション形式のいずれかのみを使用できます。
   a. カスタマイズせずに追加ネットワークを割り当てるには、以下の形式でアノテーションを追加します。**<network>** を、Pod に関連付ける追加ネットワークの名前に置き換えます。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] (1)
```

(1) 複数の追加ネットワークを指定するには、各ネットワークをコンマで区切ります。コンマの間にはスペースを入れないでください。同じ追加ネットワークを複数回指定した場合、Pod は複数のネットワークインターフェイスをそのネットワークに割り当てます。

   b. カスタマイズして追加のネットワークを割り当てるには、以下の形式でアノテーションを追加します。

```
metadata:
```

```
annotations:
  k8s.v1.cni.cncf.io/networks: |-
    [
      {
        "name": "<network>", (1)
        "namespace": "<namespace>", (2)
        "default-route": ["<default-route>"] (3)
      }
    ]
```

1. **NetworkAttachmentDefinition** オブジェクトによって定義される追加のネットワークの名前を指定します。
2. **NetworkAttachmentDefinition** オブジェクトが定義される namespace を指定します。
3. オプション: **192.168.17.1** などのデフォルトルートのオーバーライドを指定します。

2. Pod を作成するには、以下のコマンドを入力します。**<name>** を Pod の名前に置き換えます。

```
$ oc create -f <name>.yaml
```

3. オプション: アノテーションが **Pod** CR に存在することを確認するには、**<name>** を Pod の名前に置き換えて、以下のコマンドを入力します。

```
$ oc get pod <name> -o yaml
```

以下の例では、**example-pod** Pod が追加ネットワークの **net1** に割り当てられています。

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status: |- (1)
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
      }]
  name: example-pod
```

```
  namespace: default
spec:
  ...
status:
  ...
```

1. **k8s.v1.cni.cncf.io/networks-status** パラメーターは、オブジェクトの JSON 配列です。各オブジェクトは、Pod に割り当てられる追加のネットワークのステータスについて説明します。アノテーションの値はプレーンテキストの値として保存されます。

10.2.1.1. Pod 固有のアドレスおよびルーティングオプションの指定

Pod を追加のネットワークに割り当てる場合、特定の Pod でそのネットワークに関するその他のプロパティーを指定する必要がある場合があります。これにより、ルーティングの一部を変更することができ、静的 IP アドレスおよび MAC アドレスを指定できます。これを実行するには、JSON 形式のアノテーションを使用できます。

前提条件

- Pod が追加ネットワークと同じ namespace にあること。
- OpenShift コマンドラインインターフェイス (**oc**) のインストール。
- クラスターにログインすること。

手順

アドレスおよび/またはルーティングオプションを指定する間に Pod を追加のネットワークに追加するには、以下の手順を実行します。

1. **Pod** リソース定義を編集します。既存の **Pod** リソースを編集する場合は、以下のコマンドを実行してデフォルトエディターでその定義を編集します。**<name>** を、編集する **Pod** リソースの名前に置き換えます。

```
$ oc edit pod <name>
```

2. **Pod** リソース定義で、**k8s.v1.cni.cncf.io/networks** パラメーターを Pod の **metadata** マッピングに追加します。**k8s.v1.cni.cncf.io/networks** は、追加のプロパティーを指定するだけでなく、**NetworkAttachmentDefinition** カスタムリソース (CR) 名を参照するオブジェクト一覧の JSON 文字列を受け入れます。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[<network>[,<network>,...]]' (1)
```

1. **<network>** を、以下の例にあるように JSON オブジェクトに置き換えます。一重引用符が必要です。

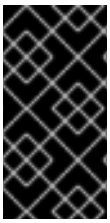
3. 以下の例では、アノテーションで **default-route** パラメーターを使用して、デフォルトルートを持つネットワーク割り当てを指定します。

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '
    {
      "name": "net1"
    },
    {
      "name": "net2", (1)
      "default-route": ["192.0.2.1"] (2)
    }'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

1. **name** キーは、Pod に関連付ける追加ネットワークの名前です。
2. **default-route** キーは、ルーティングテーブルに他のルーティングテーブルがない場合に、ルーティングされるトラフィックに使用されるゲートウェイ値を指定します。複数の **default-route** キーを指定すると、Pod がアクティブでなくなります。

デフォルトのルートにより、他のルートに指定されていないトラフィックがゲートウェイにルーティングされます。



重要

OpenShift Container Platform のデフォルトのネットワークインターフェイス以外のインターフェイスへのデフォルトのルートを設定すると、Pod 間のトラフィックについて予想されるトラフィックが別のインターフェイスでルーティングされる可能性があります。

Pod のルーティングプロパティーを確認する場合、**oc** コマンドを Pod 内で **ip** コマンドを実行するために使用できます。

```
$ oc exec -it <pod_name> -- ip route
```

注記

また、Pod の **k8s.v1.cni.cncf.io/networks-status** を参照して、JSON 形式の一覧のオブジェクトで **default-route** キーの有無を確認し、デフォルトルートが割り当てられている追加ネットワークを確認することができます。

Pod に静的 IP アドレスまたは MAC アドレスを設定するには、JSON 形式のアノテーションを使用できます。これには、この機能をとくに許可するネットワークを作成する必要があります。これは、CNO の rawCNIConfig で指定できます。

1. 以下のコマンドを実行して CNO CR を編集します。

```
$ oc edit networks.operator.openshift.io cluster
```

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

```
name: <name> (1)
namespace: <namespace> (2)
rawCNIConfig: '{ (3)
  ...
}'
type: Raw
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。
3. 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

以下のオブジェクトは、macvlan CNI プラグインを使用して静的 MAC アドレスと IP アドレスを使用するための設定パラメーターについて説明しています。

静的 IP および MAC アドレスを使用した macvlan CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", (1)
  "plugins": [{ (2)
      "type": "macvlan",
      "capabilities": { "ips": true }, (3)
      "master": "eth0", (4)
      "mode": "bridge",
      "ipam": {
        "type": "static"
      }
    }, {
      "capabilities": { "mac": true }, (5)
      "type": "tuning"
    }]
}
```

1. 作成する追加のネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. CNI プラグイン設定の配列を指定します。1 つ目のオブジェクトは、macvlan プラグイン設定を指定し、2 つ目のオブジェクトはチューニングプラグイン設定を指定します。
3. CNI プラグインのランタイム設定機能の静的 IP 機能を有効にするために要求が実行されるように指定します。
4. macvlan プラグインが使用するインターフェイスを指定します。
5. CNI プラグインの静的 MAC アドレス機能を有効にするために要求が実行されるように指定します。

上記のネットワーク割り当ては、特定の Pod に割り当てられる静的 IP アドレスと MAC アドレスを指定するキーと共に、JSON 形式のアノテーションで参照できます。

以下を使用して Pod を編集します。

```
$ oc edit pod <name>
```

静的 IP および MAC アドレスを使用した macvlan CNI プラグイン JSON 設定オブジェクト

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "<name>", (1)
        "ips": [ "192.0.2.205/24" ], (2)
        "mac": "CA:FE:C0:FF:EE:00" (3)
      }
    ]'
```

1. 上記の **rawCNIConfig** を作成する際に、指定されるように **<name>** を使用します。
2. サブネットマスクを含む IP アドレスを指定します。
3. MAC アドレスを指定します。

注記

静的 IP アドレスおよび MAC アドレスを同時に使用することはできません。これらは個別に使用することも、一緒に使用することもできます。

追加のネットワークを持つ Pod の IP アドレスと MAC プロパティーを検証するには、**oc** コマンドを使用して Pod 内で ip コマンドを実行します。

```
$ oc exec -it <pod_name> -- ip a
```

10.3. 追加ネットワークからの POD の削除

クラスターユーザーとして、追加のネットワークから Pod を削除できます。

10.3.1. 追加ネットワークからの Pod の削除

Pod を削除するだけで、追加のネットワークから Pod を削除できます。

前提条件

- 追加のネットワークが Pod に割り当てられている。
- OpenShift CLI (**oc**) をインストールしている。

- クラスターにログインする。

手順

- Pod を削除するには、以下のコマンドを入力します。

```
$ oc delete pod <name> -n <namespace>
```

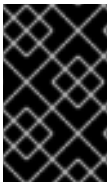
  - **<name>** は Pod の名前です。
  - **<namespace>** は Pod が含まれる namespace です。

10.4. ブリッジネットワークの設定

クラスター管理者は、ブリッジの Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。設定時に、ノード上のすべての Pod が仮想スイッチに接続されます。各 Pod には追加のネットワークの IP アドレスが割り当てられます。

10.4.1. ブリッジ CNI プラグインを使用した追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。



重要

Cluster Network Operator が管理する **NetworkAttachmentDefinition** オブジェクトは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

```
$ oc edit networks.operator.openshift.io cluster
```

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
   以下の YAML はブリッジ CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
```

```
namespace: test-1
type: Raw
rawCNIConfig: '{
  "cniVersion": "0.3.1",
  "name": "test-network-1",
  "type": "bridge",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.1.23/24"
      }
    ]
  }
}'
```

1. 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

4. 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition オブジェクトを作成していることを確認します。**<namespace>** を、ネットワーク割り当ての設定時に指定した namespace に置き換えます。CNO がオブジェクトを作成するまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions -n <namespace>
```

出力例

```
NAME              AGE
test-network-1     14m
```

10.4.1.1. ブリッジの設定

ブリッジ Container Network Interface (CNI) プラグインを使用する追加ネットワーク割り当ての設定は、以下の 2 つの部分に分けて提供されます。

- Cluster Network Operator (CNO) の設定
- CNI プラグインの設定

CNO 設定では、追加ネットワーク割り当ての名前と割り当てを作成する namespace を指定します。このプラグインは、CNO 設定の **rawCNIConfig** パラメーターで指定される JSON オブジェクトで設定されます。

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

```
name: <name> (1)
namespace: <namespace> (2)
rawCNIConfig: '{ (3)
```

```
  ...
}'
type: Raw
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。
3. 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

以下のオブジェクトは、ブリッジ CNI プラグインの設定パラメーターについて説明しています。

ブリッジ CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", (1)
  "type": "bridge",
  "bridge": "<bridge>", (2)
  "ipam": { (3)
    ...
  },
  "ipMasq": false, (4)
  "isGateway": false, (5)
  "isDefaultGateway": false, (6)
  "forceAddress": false, (7)
  "hairpinMode": false, (8)
  "promiscMode": false, (9)
  "vlan": <vlan>, (10)
  "mtu": <mtu> (11)
}
```

1. CNO 設定に以前に指定した **name** パラメーターの値を指定します。
2. 使用する仮想ブリッジの名前を指定します。ブリッジインターフェイスがホストに存在しない場合は、これが作成されます。デフォルト値は **cni0** です。
3. IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、ネットワーク割り当て定義についての IP アドレスの割り当てを管理します。
4. 仮想ネットワークから外すトラフィックについて IP マスカレードを有効にするには、**true** に設定します。すべてのトラフィックのソース IP アドレスは、ブリッジの IP アドレスに書き換えられます。ブリッジに IP アドレスがない場合は、この設定は影響を与えません。デフォルト値は **false** です。
5. IP アドレスをブリッジに割り当てるには **true** に設定します。デフォルト値は **false** です。
6. ブリッジを仮想ネットワークのデフォルトゲートウェイとして設定するには、**true** に設定します。デフォルト値は **false** です。**isDefaultGateway** が **true** に設定される場合、**isGateway** も自動的に **true** に設定されます。

7. 仮想ブリッジの事前に割り当てられた IP アドレスの割り当てを許可するには、**true** に設定します。**false** に設定される場合、重複サブセットの IPv4 アドレスまたは IPv6 アドレスが仮想ブリッ

8. 仮想ブリッジが受信時に使用した仮想ポートでイーサネットフレームを送信できるようにするには、**true** に設定します。このモードは、**Reflective Relay** (リフレクティブリレー) としても知られています。デフォルト値は **false** です。

9. ブリッジで無作為検出モード (Promiscuous Mode) を有効にするには、**true** に設定します。デフォルト値は **false** です。

10. 仮想 LAN (VLAN) タグを整数値として指定します。デフォルトで、VLAN タグは割り当てません。

11. 最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。

10.4.1.1.1. ブリッジ設定の例

以下の例では、**bridge-net** という名前の追加のネットワークを設定します。

```
name: bridge-net
namespace: work-network
type: Raw
rawCNIConfig: '{ (1)
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
    }
}'
```

1. CNI 設定オブジェクトは YAML 文字列として指定されます。

10.4.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。

IP アドレスの割り当てには、以下の方法を使用できます。

- 静的割り当て。
- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

10.4.1.2.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ (1)
      {
        "address": "<address>", (2)
        "gateway": "<gateway>" (3)
      }
    ],
    "routes": [ (4)
      {
        "dst": "<dst>", (5)
        "gw": "<gw>" (6)
      }
    ],
    "dns": { (7)
      "nameservers": ["<nameserver>"], (8)
      "domain": "<domain>", (9)
      "search": ["<search_domain>"] (10)
    }
  }
}
```

1. 仮想インターフェイスに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、追加のネットワークに IP アドレスの **10.10.21.10** が割り当てられ、ネットマスクは **255.255.255.0** になります。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。
5. CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

10.4.1.2.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP 割り当ての設定

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.4.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の JSON は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

Whereabouts 割り当て設定

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", (1)
```

```
    "exclude": ["<exclude_part>, ..."], (2)
  }
}
```

1. IP アドレスと範囲を CIDR 表記で指定します。IP アドレスは、この範囲内のアドレスから割り当てられます。
2. オプション: CIDR 表記で IP アドレスおよび範囲の一覧を指定します。除外されたアドレス範囲内の IP アドレスは割り当てられません。

10.4.1.2.4. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

10.4.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.4.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例

Whereabouts を使用するように IPAM を設定できます。

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

10.4.2. 次のステップ

- 追加ネットワークへの Pod の割り当て

10.5. ホストデバイスネットワークの設定

クラスター管理者は、ホストデバイス Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。このプラグインは、指定されたネットワークデバイスを、ホストのネットワーク namespace から Pod のネットワーク namespace に移動します。

10.5.1. ホストデバイス CNI プラグインを使用した追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

重要

Cluster Network Operator が管理する **NetworkAttachmentDefinition** オブジェクトは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

```
$ oc edit networks.operator.openshift.io cluster
```

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
以下の YAML は、ホストデバイス CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "host-device",
      "device": "eth1",
      "ipam": {
        "type": "static",
        "addresses": [
```

```
        {
          "address": "192.168.1.23/24"
        }
      ]
    }
  }'
```

1. 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

4. 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition オブジェクトを作成していることを確認します。**<namespace>** を、ネットワーク割り当ての設定時に指定した namespace に置き換えます。CNO がオブジェクトを作成するまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions -n <namespace>
```

出力例

```
NAME              AGE
test-network-1     14m
```

10.5.1.1. ホストデバイスの設定

ホストデバイス Container Network Interface (CNI) プラグインを使用する追加ネットワーク割り当ての設定は、以下の 2 つの部分に分けて提供されます。

- Cluster Network Operator (CNO) の設定
- CNI プラグインの設定

CNO 設定では、追加ネットワーク割り当ての名前と割り当てを作成する namespace を指定します。このプラグインは、CNO 設定の **rawCNIConfig** パラメーターで指定される JSON オブジェクトで設定されます。

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

```
name: <name> (1)
namespace: <namespace> (2)
rawCNIConfig: '{ (3)
  ...
}'
type: Raw
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。

2. ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。

3. 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

重要

device、**hwaddr**、 **kernelpath**、または **pciBusID** のいずれかのパラメーターを設定してネットワークデバイスを指定します。

以下のオブジェクトは、ホストデバイス CNI プラグインの設定パラメーターについて説明しています。

ホストデバイス CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", (1)
  "type": "host-device",
  "device": "<device>", (2)
  "hwaddr": "<hwaddr>", (3)
  "kernelpath": "<kernelpath>", (4)
  "pciBusID": "<pciBusID>", (5)
  "ipam": { (6)
    ...
  }
}
```

1. CNO 設定に以前に指定した **name** パラメーターの値を指定します。
2. **eth0** などのデバイスの名前を指定します。
3. デバイスハードウェアの MAC アドレスを指定します。
4. **/sys/devices/pci0000:00/0000:00:1f.6** などの Linux カーネルデバイスを指定します。
5. **0000:00:1f.6** などのネットワークデバイスの PCI アドレスを指定します。
6. IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

10.5.1.1.1. ホストデバイス設定例

以下の例では、**hostdev-net** という名前の追加のネットワークを設定します。

```
name: hostdev-net
namespace: work-network
type: Raw
rawCNIConfig: '{ (1)
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "host-device",
  "device": "eth1",
  "ipam": {
```

```
    "type": "dhcp"
  }
}'
```

1 CNI 設定オブジェクトは YAML 文字列として指定されます。

10.5.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。

IP アドレスの割り当てには、以下の方法を使用できます。

- 静的割り当て。
- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

10.5.1.2.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ 1
      {
        "address": "<address>", 2
        "gateway": "<gateway>" 3
      }
    ],
    "routes": [ 4
      {
        "dst": "<dst>", 5
        "gw": "<gw>" 6
      }
    ],
    "dns": { 7
      "nameservers": ["<nameserver>"], 8
      "domain": "<domain>", 9
      "search": ["<search_domain>"] 10
    }
  }
}
```

1 仮想インターフェイスに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。

2

指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、追加のネットワークに IP アドレスの **10.10.21.10** が割り当てられ、ネットマスクは **255.255.255.0** に

3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。
5. CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

10.5.1.2.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP 割り当ての設定

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.5.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の JSON は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

Whereabouts 割り当て設定

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", (1)
```

```
    "exclude": ["<exclude_part>, ..."], (2)
  }
}
```

1. IP アドレスと範囲を CIDR 表記で指定します。IP アドレスは、この範囲内のアドレスから割り当てられます。
2. オプション: CIDR 表記で IP アドレスおよび範囲の一覧を指定します。除外されたアドレス範囲内の IP アドレスは割り当てられません。

10.5.1.2.4. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

10.5.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.5.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例

Whereabouts を使用するように IPAM を設定できます。

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
       "192.0.2.192/30",
       "192.0.2.196/32"
    ]
  }
}
```

10.5.2. 次のステップ

- 追加ネットワークへの Pod の割り当て

10.6. IPVLAN ネットワークの設定

クラスター管理者は、ipvlan Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。このプラグインにより作成される仮想ネットワークは、指定する物理インターフェイスに関連付けられます。

10.6.1. IPVLAN CNI プラグインを使用した追加のネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

重要

Cluster Network Operator が管理する **NetworkAttachmentDefinition** オブジェクトは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

```
$ oc edit networks.operator.openshift.io cluster
```

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
   以下の YAML は IPVLAN CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "ipvlan",
      "master": "eth1",
      "mode": "l2",
      "ipam": {
        "type": "static",
```

```
        "addresses": [
          {
            "address": "192.168.1.23/24"
          }
        ]
      }
    }'
```

1. 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

4. 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition オブジェクトを作成していることを確認します。**<namespace>** を、ネットワーク割り当ての設定時に指定した namespace に置き換えます。CNO がオブジェクトを作成するまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions -n <namespace>
```

出力例

```
NAME              AGE
test-network-1     14m
```

10.6.1.1. IPVLAN の設定

IPVLAN Container Network Interface (CNI) プラグインを使用する追加のネットワーク割り当ての設定は、以下の 2 つの部分に分けて提供されます。

- Cluster Network Operator (CNO) の設定
- CNI プラグインの設定

CNO 設定では、追加ネットワーク割り当ての名前と割り当てを作成する namespace を指定します。このプラグインは、CNO 設定の **rawCNIConfig** パラメーターで指定される JSON オブジェクトで設定されます。

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

```
name: <name> (1)
namespace: <namespace> (2)
rawCNIConfig: '{ (3)
  ...
}'
type: Raw
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。

2. ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。

3. 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

以下のオブジェクトは、IPVLAN CNI プラグインの設定パラメーターについて説明しています。

IPVLAN CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", (1)
  "type": "ipvlan",
  "mode": "<mode>", (2)
  "master": "<master>", (3)
  "mtu": <mtu>, (4)
  "ipam": { (5)
    ...
  }
}
```

1. CNO 設定に以前に指定した **name** パラメーターの値を指定します。
2. 仮想ネットワークの操作モードを指定します。この値は、**l2**、**l3**、または **l3s** である必要があります。デフォルト値は **l2** です。
3. ネットワーク割り当てに関連付けるイーサネットインターフェイスを指定します。**master** が指定されない場合、デフォルトのネットワークルートのインターフェイスが使用されます。
4. 最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。
5. IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

10.6.1.1.1. IPVLAN 設定例

以下の例では、**ipvlan-net** という名前の追加のネットワークを設定します。

```
name: ipvlan-net
namespace: work-network
type: Raw
rawCNIConfig: '{ (1)
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "l3",
  "ipam": {
    "type": "dhcp"
    }
}'
```

1. CNI 設定オブジェクトは YAML 文字列として指定されます。

10.6.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。

IP アドレスの割り当てには、以下の方法を使用できます。

- 静的割り当て。
- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

10.6.1.2.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ (1)
      {
        "address": "<address>", (2)
        "gateway": "<gateway>" (3)
      }
    ],
    "routes": [ (4)
      {
        "dst": "<dst>", (5)
        "gw": "<gw>" (6)
      }
    ],
    "dns": { (7)
      "nameservers": ["<nameserver>"], (8)
      "domain": "<domain>", (9)
      "search": ["<search_domain>"] (10)
    }
  }
}
```

1. 仮想インターフェイスに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、追加のネットワークに IP アドレスの **10.10.21.10** が割り当てられ、ネットマスクは **255.255.255.0** になります。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。

5. CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

10.6.1.2.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP 割り当ての設定

```
{
  "ipam": {
```

```
    "type": "dhcp"
  }
}
```

10.6.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の JSON は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

Whereabouts 割り当て設定

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", (1)
    "exclude": ["<exclude_part>, ..."], (2)
  }
}
```

1. IP アドレスと範囲を CIDR 表記で指定します。IP アドレスは、この範囲内のアドレスから割り当てられます。
2. オプション: CIDR 表記で IP アドレスおよび範囲の一覧を指定します。除外されたアドレス範囲内の IP アドレスは割り当てられません。

10.6.1.2.4. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

10.6.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.6.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例

Whereabouts を使用するように IPAM を設定できます。

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

10.6.2. 次のステップ

- 追加ネットワークへの Pod の割り当て

10.7. 基本的なカスタマイズを使用した MACVLAN ネットワークの設定

クラスター管理者は、macvlan Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。Pod がネットワークに割り当てられている場合、プラグインはホストの親インターフェイスからサブインターフェイスを作成します。各サブデバイスに対して固有のハードウェアの MAC アドレスが生成されます。

重要

このプラグインがサブインターフェイス用に生成する固有の MAC アドレスは、クラウドプロバイダーのセキュリティーポリシーとの互換性がない場合があります。

基本的な設定を YAML に直接指定します。この方法では、JSON で CNI オブジェクトを直接使用して macvlan 設定を指定するよりも設定オプションが少なくなります。

10.7.1. macvlan CNI プラグインを使用した追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

重要

Cluster Network Operator が管理する **NetworkAttachmentDefinition** オブジェクトは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

```
$ oc edit networks.operator.openshift.io cluster
```

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
以下の YAML は、macvlan CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: SimpleMacvlan
    simpleMacvlanConfig:
      ipamConfig:
        type: static
        staticIPAMConfig:
          addresses:
          - address: 10.1.1.7/24
```

(1) 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

4. 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition オブジェクトを作成していることを確認します。**<namespace>** を、ネットワーク割り当ての設定時に指定した namespace に置き換えます。CNO がオブジェクトを作成するまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions -n <namespace>
```

出力例

```
NAME              AGE
test-network-1     14m
```

10.7.1.1. macvlan CNI プラグインの設定

以下の YAML は、macvlan Container Network Interface (CNI) プラグインの設定パラメーターについて説明しています。

macvlan YAML の設定

```
name: <name> (1)
namespace: <namespace> (2)
type: SimpleMacvlan
simpleMacvlanConfig:
  master: <master> (3)
```

```
mode: <mode> 4
mtu: <mtu> 5
ipamConfig: 6
  ...
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. ネットワークの割り当てを作成する namespace を指定します。値が指定されない場合は、**default** namespace が使用されます。
3. 仮想インターフェイスに関連付けるイーサネットインターフェイス。**master** の値が指定されない場合、ホストシステムのプライマリーイーサネットインターフェイスが使用されます。
4. 仮想ネットワークのトラフィックの可視性を設定します。**bridge**、**passthru**、**private**、または **vepa** のいずれかである必要があります。**mode** の値が指定されない場合、デフォルトの値は **bridge** になります。
5. 最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。
6. IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

10.7.1.1.1. macvlan 設定の例

以下の例では、**macvlan-net** という名前の追加のネットワークを設定します。

```
name: macvlan-net
namespace: work-network
type: SimpleMacvlan
simpleMacvlanConfig:
  ipamConfig:
    type: DHCP
```

10.7.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。

以下の YAML 設定は設定可能なパラメーターについて説明しています。

IPAM CNI プラグイン YAML 設定オブジェクト

```
ipamConfig:
  type: <type> 1
  ... 2
```

1. IP アドレスの割り当てを管理できるようにプラグインを設定するには **static** を指定します。**DHCP** を指定して、DHCP サーバーが IP アドレスの割り当てを管理できるようにします。**DHCP** の値を指定する場合は、追加のパラメーターを指定できません。
2. **type** パラメーターを **static** に設定する場合、**staticIPAMConfig** パラメーターを指定します。

10.7.1.2.1. 静的 IPAM 設定 YAML

以下の YAML は、静的 IP アドレスの割り当ての設定について説明しています。

静的 IPAM 設定 YAML

```
ipamConfig:
  type: static
  staticIPAMConfig:
    addresses: (1)
    - address: <address> (2)
      gateway: <gateway> (3)
    routes: (4)
    - destination: <destination> (5)
      gateway: <gateway> (6)
    dns: (7)
      nameservers: (8)
      - <nameserver>
      domain: <domain> (9)
      search: (10)
      - <search_domain>
```

1. 仮想インターフェイスに割り当てる IP アドレスを定義するマッピングのコレクション。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、追加のネットワークに IP アドレスの **10.10.21.10** が割り当てられ、ネットマスクは **255.255.255.0** になります。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述するマッピングのコレクション。
5. CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーを送信する 1 つ以上の IP アドレスのコレクション。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

10.7.1.2.2. 動的 IPAM 設定 YAML

以下の YAML は、静的 IP アドレスの割り当ての設定について説明しています。

動的 IPAM 設定 YAML

```
ipamConfig:
  type: DHCP
```

10.7.1.2.3. 静的 IP アドレス割り当ての設定例

以下の例は、静的 IP アドレスの IPAM 設定を示しています。

```
ipamConfig:
  type: static
  staticIPAMConfig:
    addresses:
    - address: 198.51.100.11/24
      gateway: 198.51.100.10
    routes:
    - destination: 0.0.0.0/0
      gateway: 198.51.100.1
    dns:
      nameservers:
      - 198.51.100.1
      - 198.51.100.2
      domain: testDNS.example
      search:
      - testdomain1.example
      - testdomain2.example
```

10.7.1.2.4. 動的 IP アドレス割り当ての設定例

以下の例では、DHCP の IPAM 設定を示しています。

```
ipamConfig:
  type: DHCP
```

10.7.2. 次のステップ

- 追加ネットワークへの Pod の割り当て

10.8. MACVLAN ネットワークの設定

クラスター管理者は、高度にカスタマイズされた macvlan Container Network Interface (CNI) プラグインを使用して、クラスターの追加ネットワークを設定できます。Pod がネットワークに割り当てられている場合、プラグインはホストの親インターフェイスからサブインターフェイスを作成します。各サブデバイスに対して固有のハードウェアの MAC アドレスが生成されます。

重要

このプラグインがサブインターフェイス用に生成する固有の MAC アドレスは、クラウドプロバイダーのセキュリティーポリシーとの互換性がない場合があります。

CNI オブジェクトで設定を指定します。この方法では、YAML 設定を使用する場合に利用できない追加の設定オプションを指定できます。

10.8.1. macvlan CNI プラグインを使用した追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

重要

Cluster Network Operator が管理する **NetworkAttachmentDefinition** オブジェクトは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターの追加ネットワークを作成するには、以下の手順を実施します。

1. 以下のコマンドを実行して CNO CR を編集します。

```
$ oc edit networks.operator.openshift.io cluster
```

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。
以下の YAML は、macvlan CNI プラグインを設定します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: (1)
  - name: test-network-1
    namespace: test-1
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "test-network-1",
      "type": "macvlan",
      "master": "eth1",
      "ipam": {
        "type": "static",
        "addresses": [
          {
            "address": "192.168.1.23/24"
          }
        ]
      }
    }'
```

(1) 追加ネットワーク割り当て定義の設定を指定します。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

4. 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition オブジェクトを作成していることを確認します。**<namespace>** を、ネットワーク割り当ての設定時に指定した namespace に置き換えます。CNO がオブジェクトを作成するまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions -n <namespace>
```

出力例

```
NAME              AGE
test-network-1     14m
```

10.8.1.1. macvlan CNI プラグインの設定

macvlan Container Network Interface (CNI) プラグインを使用する追加のネットワーク割り当ての設定は、以下の 2 つの部分に分けて提供されます。

- Cluster Network Operator (CNO) の設定
- CNI プラグインの設定

CNO 設定では、追加ネットワーク割り当ての名前と割り当てを作成する namespace を指定します。このプラグインは、CNO 設定の **rawCNIConfig** パラメーターで指定される JSON オブジェクトで設定されます。

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

```
name: <name> (1)
namespace: <namespace> (2)
rawCNIConfig: '{ (3)
  ...
}'
type: Raw
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。
3. 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

以下のオブジェクトは、macvlan CNI プラグインの設定パラメーターについて説明しています。

macvlan CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", (1)
```

```
  "type": "macvlan",
  "mode": "<mode>", (2)
  "master": "<master>", (3)
  "mtu": <mtu>, (4)
  "ipam": { (5)
    ...
  }
}
```

1. 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
2. 仮想ネットワークのトラフィックの可視性を設定します。**bridge**、**passthru**、**private**、または **vepa** のいずれかである必要があります。値が指定されない場合、デフォルト値は **bridge** になります。
3. 仮想インターフェイスに関連付けるイーサネットインターフェイス。値が指定されない場合、ホストシステムのプライマリーイーサネットインターフェイスが使用されます。
4. 最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。
5. IPAM CNI プラグインの設定オブジェクトを指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

10.8.1.1.1. macvlan 設定の例

以下の例では、**macvlan-net** という名前の追加のネットワークを設定します。

```
name: macvlan-net
namespace: work-network
type: Raw
rawCNIConfig: |-
  {
    "cniVersion": "0.3.1",
    "name": "macvlan-net",
    "type": "macvlan",
    "master": "eth1",
    "mode": "bridge",
    "ipam": {
      "type": "dhcp"
      }
  }
```

10.8.1.2. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。

IP アドレスの割り当てには、以下の方法を使用できます。

- 静的割り当て。

- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

10.8.1.2.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ (1)
      {
        "address": "<address>", (2)
        "gateway": "<gateway>" (3)
      }
    ],
    "routes": [ (4)
      {
        "dst": "<dst>", (5)
        "gw": "<gw>" (6)
      }
    ],
    "dns": { (7)
      "nameservers": ["<nameserver>"], (8)
      "domain": "<domain>", (9)
      "search": ["<search_domain>"] (10)
    }
  }
}
```

1. 仮想インターフェイスに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、追加のネットワークに IP アドレスの **10.10.21.10** が割り当てられ、ネットマスクは **255.255.255.0** になります。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。
5. CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
9.

ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き

10 DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

10.8.1.2.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP 割り当ての設定

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.8.1.2.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の JSON は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

Whereabouts 割り当て設定

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", (1)
    "exclude": ["<exclude_part>, ..."], (2)
  }
}
```

1. IP アドレスと範囲を CIDR 表記で指定します。IP アドレスは、この範囲内のアドレスから割り当てられます。
2. オプション: CIDR 表記で IP アドレスおよび範囲の一覧を指定します。除外されたアドレス範囲内の IP アドレスは割り当てられません。

10.8.1.2.4. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

10.8.1.2.5. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

10.8.1.2.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例

Whereabouts を使用するように IPAM を設定できます。

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
```

```
        "192.0.2.192/30",
        "192.0.2.196/32"
      ]
    }
  }
```

10.8.2. 次のステップ

- 追加ネットワークへの Pod の割り当て

10.9. 追加ネットワークの編集

クラスター管理者は、既存の追加ネットワークの設定を変更することができます。

10.9.1. 追加ネットワーク割り当て定義の変更

クラスター管理者は、既存の追加ネットワークに変更を加えることができます。追加ネットワークに割り当てられる既存の Pod は更新されません。

前提条件

- クラスター用に追加のネットワークを設定している。
- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターの追加ネットワークを編集するには、以下の手順を実行します。

1. 以下のコマンドを実行し、デフォルトのテキストエディターで Cluster Network Operator (CNO) CR を編集します。

   ```
   $ oc edit networks.operator.openshift.io cluster
   ```

2. **additionalNetworks** コレクションで、追加ネットワークを変更内容で更新します。
3. 変更を保存し、テキストエディターを終了して、変更をコミットします。
4. オプション: 以下のコマンドを実行して、CNO が **NetworkAttachmentDefinition** オブジェクトを更新していることを確認します。**<network-name>** を表示する追加ネットワークの名前に置き換えます。CNO が **NetworkAttachmentDefinition** オブジェクトを更新して変更内容が反映されるまでに遅延が生じる可能性があります。

   ```
   $ oc get network-attachment-definitions <network-name> -o yaml
   ```

   たとえば、以下のコンソールの出力は **net1** という名前の **NetworkAttachmentDefinition** オブジェクトを表示します。

   ```
   $ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\n" .spec.config}}'
   { "cniVersion": "0.3.1", "type": "macvlan",
   "master": "ens5",
   "mode": "bridge",
   ```

```
"ipam":      {"type":"static","routes":[{"dst":"0.0.0.0/0","gw":"10.128.2.1"}],"addresses":
[{"address":"10.128.2.100/23","gateway":"10.128.2.1"}],"dns":{"nameservers":
["172.30.0.10"],"domain":"us-west-2.compute.internal","search":["us-west-
2.compute.internal"]}} }
```

10.10. 追加ネットワークの削除

クラスター管理者は、追加のネットワーク割り当てを削除できます。

10.10.1. 追加ネットワーク割り当て定義の削除

クラスター管理者は、追加ネットワークを OpenShift Container Platform クラスターから削除できます。追加ネットワークは、割り当てられている Pod から削除されません。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターから追加ネットワークを削除するには、以下の手順を実行します。

1. 以下のコマンドを実行して、デフォルトのテキストエディターで Cluster Network Operator (CNO) を編集します。

```
$ oc edit networks.operator.openshift.io cluster
```

2. 削除しているネットワーク割り当て定義の **additionalNetworks** コレクションから設定を削除し、CR を変更します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: [] (1)
```

(1) **additionalNetworks** コレクションの追加ネットワーク割り当てのみの設定マッピングを削除する場合、空のコレクションを指定する必要があります。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。
4. オプション: 以下のコマンドを実行して、追加ネットワーク CR が削除されていることを確認します。

```
$ oc get network-attachment-definition --all-namespaces
```

第11章 ハードウェアネットワーク

11.1. SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) ハードウェアネットワークについて

Single Root I/O Virtualization (SR-IOV) 仕様は、単一デバイスを複数の Pod で共有できる PCI デバイス割り当てタイプの標準です。

SR-IOV を使用すると、準拠したネットワークデバイス (ホストノードで物理機能 (PF) として認識される) を複数の仮想機能 (VF) にセグメント化することができます。VF は他のネットワークデバイスと同様に使用されます。デバイスの SR-IOV デバイスドライバーは、VF がコンテナーで公開される方法を判別します。

- **netdevice** ドライバー: コンテナーの **netns** 内の通常のカーネルネットワークデバイス
- **vfio-pci** ドライバー: コンテナーにマウントされるキャラクターデバイス

高帯域幅または低レイテンシーを必要とするアプリケーション用に、OpenShift Container Platform クラスターの追加のネットワークと共に SR-IOV ネットワークデバイスを使用できます。

11.1.1. SR-IOV ネットワークデバイスを管理するコンポーネント

SR-IOV Network Operator は SR-IOV スタックのコンポーネントを作成し、管理します。以下の機能を実行します。

- SR-IOV ネットワークデバイスの検出および管理のオーケストレーション
- SR-IOV Container Network Interface (CNI) の **NetworkAttachmentDefinition** カスタムリソースの生成
- SR-IOV ネットワークデバイスプラグインの設定の作成および更新
- ノード固有の **SriovNetworkNodeState** カスタムリソースの作成
- 各 **SriovNetworkNodeState** カスタムリソースの **spec.interfaces** フィールドの更新

Operator は以下のコンポーネントをプロビジョニングします。

SR-IOV ネットワーク設定デーモン

SR-IOV Operator の起動時にワーカーノードにデプロイされる DaemonSet。デーモンは、クラスターで SR-IOV ネットワークデバイスを検出し、初期化します。

SR-IOV Operator Webhook

Operator カスタムリソースを検証し、未設定フィールドに適切なデフォルト値を設定する動的受付コントローラー Webhook。

SR-IOV Network Resources Injector

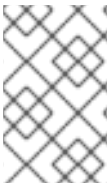
SR-IOV VF などのカスタムネットワークリソースの要求および制限のある Kubernetes Pod 仕様のパッチを適用するための機能を提供する動的受付コントローラー Webhook。

SR-IOV ネットワークデバイスプラグイン

SR-IOV ネットワーク Virtual Function (VF) リソースの検出、公開、割り当てを実行するデバイスプラグイン。デバイスプラグインは、とりわけ物理デバイスでの制限されたリソースの使用を有効にするために Kubernetes で使用されます。デバイスプラグインは Kubernetes スケジューラーにリソースの可用性を認識させるため、スケジューラーはリソースが十分にあるノードで Pod をスケジュールできます。

SR-IOV CNI プラグイン

SR-IOV デバイスプラグインから割り当てられる VF インターフェイスを直接 Pod に割り当てる CNI プラグイン。



注記

SR-IOV Network Resources Injector および SR-IOV Network Operator Webhook は、デフォルトで有効にされ、**default** の **SriovOperatorConfig** CR を編集して無効にできます。

11.1.1.1. サポートされるデバイス

以下の Network Interface Card (NIC) モデルは、OpenShift Container Platform でサポートされています。

- Intel XXV710 25GbE SFP28 (ベンダー ID **0x8086** およびデバイス ID **0x158b**)
- Mellanox MT27710 Family [ConnectX-4 Lx] 25GbE dual-port SFP28 (ベンダー ID **0x15b3** およびデバイス ID **0x1015**)
- Mellanox MT27800 Family [ConnectX-5] 25GbE dual-port SFP28 (ベンダー ID **0x15b3** およびデバイス ID **0x1017**)
- Mellanox MT27800 Family [ConnectX-5] 100GbE (ベンダー ID **0x15b3** およびデバイス ID **0x1017**)

11.1.1.2. SR-IOV ネットワークデバイスの自動検出

SR-IOV Network Operator は、クラスターでワーカーノード上の SR-IOV 対応ネットワークデバイスを検索します。Operator は、互換性のある SR-IOV ネットワークデバイスを提供する各ワーカーノードの SriovNetworkNodeState カスタムリソース (CR) を作成し、更新します。

CR にはワーカーノードと同じ名前が割り当てられます。**status.interfaces** 一覧は、ノード上のネットワークデバイスについての情報を提供します。

重要

SriovNetworkNodeState オブジェクトは変更しないでください。Operator はこれらのリソースを自動的に作成し、管理します。

11.1.1.2.1. SriovNetworkNodeState オブジェクトの例

以下の YAML は、SR-IOV Network Operator によって作成される **SriovNetworkNodeState** オブジェクトの例です。

SriovNetworkNodeState オブジェクト

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 (1)
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
```

```
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: 2
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
  syncStatus: Succeeded
```

1. **name** フィールドの値はワーカーノードの名前と同じです。
2. **interfaces** スタンザには、ワーカーノード上の Operator によって検出されるすべての SR-IOV デバイスの一覧が含まれます。

11.1.1.3. Pod での Virtual Function (VF) の使用例

SR-IOV VF が割り当てられている Pod で、Remote Direct Memory Access (RDMA) または Data Plane Development Kit (DPDK) アプリケーションを実行できます。

以下の例では、RDMA モードで Virtual Function (VF) を使用する Pod を示しています。

RDMA モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
  - name: testpmd
    image: <RDMA_image>
    imagePullPolicy: IfNotPresent
    securityContext:
     capabilities:
        add: ["IPC_LOCK"]
    command: ["sleep", "infinity"]
```

以下の例は、DPDK モードの VF のある Pod を示しています。

DPDK モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
     capabilities:
        add: ["IPC_LOCK"]
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
      requests:
        memory: "1Gi"
        cpu: "2"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

オプションのライブラリーは、コンテナーで実行されるアプリケーションによる Pod 関連のネットワーク情報を収集を支援するために利用できます。このライブラリーは 'app-netutil' と呼ばれます。**app-netutil** GitHub リポジトリー でライブラリーのソースコードを参照してください。

このライブラリーは、DPDK モードの SR-IOV VF のコンテナーへの統合を容易にすることを目的としています。ライブラリーは、GO API と C API、および両方の言語の使用例を提供します。

また、サンプルの Docker イメージ 'dpdk-app-centos' も用意されています。このイメージは、Pod 仕様の l2fwd、l3wd または testpmd の環境変数に基づいて、以下の DPDK サンプルアプリケーションのいずれかを実行できます。この Docker イメージは、app-netutil をコンテナーイメージ自体に統合するサンプルを提供します。ライブラリーも、必要なデータを収集し、データを既存の DPDK ワークロードに渡す init-container に統合できます。

11.1.2. 次のステップ

- SR-IOV Network Operator のインストール
- オプション: SR-IOV ネットワーク Operator の設定
- SR-IOV ネットワークデバイスの設定
- OpenShift Virtualization を使用する場合: 仮想マシンの SR-IOV ネットワークデバイスの設定
- SR-IOV ネットワーク割り当ての設定
- Pod の SR-IOV の追加ネットワークへの追加

11.2. SR-IOV NETWORK OPERATOR のインストール

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator をクラスターにインストールし、SR-IOV ネットワークデバイスとネットワークの割り当てを管理できます。

11.2.1. SR-IOV Network Operator のインストール

クラスター管理者は、OpenShift Container Platform CLI または Web コンソールを使用して SR-IOV Network Operator をインストールできます。

11.2.1.1. CLI: SR-IOV Network Operator のインストール

クラスター管理者は、CLI を使用して Operator をインストールできます。

前提条件

- SR-IOV に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つアカウント。

手順

1. **openshift-sriov-network-operator** namespace を作成するには、以下のコマンドを入力します。

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
EOF
```

2. OperatorGroup CR を作成するには、以下のコマンドを実行します。

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
EOF
```

3. SR-IOV Network Operator にサブスクライブします。

   a. 以下のコマンドを実行して OpenShift Container Platform のメジャーおよびマイナーバージョンを取得します。これは、次の手順の **channel** の値に必要です。

```
$ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
    grep -o '[0-9]*[.][0-9]*' | head -1)
```

   b. SR-IOV Network Operator の Subscription CR を作成するには、以下のコマンドを入力します。

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  namespace: openshift-sriov-network-operator
spec:
  channel: "${OC_VERSION}"
  name: sriov-network-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Operator がインストールされていることを確認するには、以下のコマンドを入力します。

```
$ oc get csv -n openshift-sriov-network-operator \
  -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

出力例

```
Name                                      Phase
sriov-network-operator.4.4.0-202006160135   Succeeded
```

11.2.1.2. Web コンソール: SR-IOV Network Operator のインストール

クラスター管理者は、Web コンソールを使用して Operator をインストールできます。

注記

CLI を使用して Operator グループを作成する必要があります。

前提条件

- SR-IOV に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つアカウント。

手順

1. SR-IOV Network Operator の namespace を作成します。
   a. OpenShift Container Platform Web コンソールで、**Administration → Namespaces** をクリックします。
   b. **Create Namespace** をクリックします。
   c. **Name** フィールドに **openshift-sriov-network-operator** を入力し、**Create** をクリックします。
2. SR-IOV Network Operator をインストールします。
   a. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
   b. 利用可能な Operator の一覧から **SR-IOV Network Operator** を選択してから **Install** をクリックします。
   c. **Install Operator** ページの **A specific namespace on the cluster**の下で、**openshift-sriov-network-operator** を選択します。
   d. **Install** をクリックします。
3. SR-IOV Network Operator が正常にインストールされていることを確認します。
   a. **Operators → Installed Operators** ページに移動します。
   b. **Status** が **InstallSucceeded** の状態で、**SR-IOV Network Operator** が **openshift-sriov-network-operator** プロジェクトに一覧表示されていることを確認します。

注記

インストール時に、 Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

- **Operator Subscriptions** および **Install Plans** タブで、**Status** の下の失敗またはエラーの有無を確認します。
- **Workloads → Pods** ページに移動し、**openshift-sriov-network-operator** プロジェクトで Pod のログを確認します。

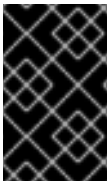
11.2.2. 次のステップ

- オプション: SR-IOV ネットワーク Operator の設定

11.3. SR-IOV NETWORK OPERATOR の設定

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator は、クラスターで SR-IOV ネットワークデバイスおよびネットワーク割り当てを管理します。

11.3.1. SR-IOV Network Operator の設定



重要

通常、SR-IOV Network Operator 設定を変更する必要はありません。デフォルト設定は、ほとんどのユースケースで推奨されます。Operator のデフォルト動作がユースケースと互換性がない場合にのみ、関連する設定を変更する手順を実行します。

SR-IOV Network Operator は **SriovOperatorConfig.sriovnetwork.openshift.io** CustomResourceDefinition リソースを追加します。Operator は、**openshift-sriov-network-operator** namespace に **default** という名前の SriovOperatorConfig カスタムリソース (CR) を自動的に作成します。

注記

default CR には、クラスターの SR-IOV Network Operator 設定が含まれます。Operator 設定を変更するには、この CR を変更する必要があります。

SriovOperatorConfig オブジェクトは、Operator を設定するための複数のフィールドを提供します。

- **enableInjector** を使用すると、プロジェクト管理者は Network Resources Injector デーモンセットを有効または無効にすることができます。
- **enableOperatorWebhook** を使用すると、プロジェクト管理者は Operator Admission Controller webhook デーモンセットを有効または無効にすることができます。
- **configDaemonNodeSelector** を使用すると、プロジェクト管理者は選択したノードで SR-IOV Network Config Daemon をスケジュールできます。

11.3.1.1. Network Resources Injector について

Network Resources Injector は Kubernetes Dynamic Admission Controller アプリケーションです。これは、以下の機能を提供します。

- SR-IOV リソース名を SR-IOV ネットワーク割り当て定義アノテーションに従って追加するための、**Pod** 仕様でのリソース要求および制限の変更。
- Pod のアノテーションおよびラベルを **/etc/podnetinfo** パスの下にあるファイルとして公開するための、Downward API ボリュームでの **Pod** 仕様の変更。

デフォルトで、Network Resources Injector は SR-IOV Operator によって有効にされ、すべてのマスターノードでデーモンセットとして実行されます。以下は、3 つのマスターノードを持つクラスターで実行される Network Resources Injector Pod の例です。

```
$ oc get pods -n openshift-sriov-network-operator
```

出力例

```
NAME                                      READY   STATUS    RESTARTS   AGE
network-resources-injector-5cz5p          1/1     Running   0          10m
network-resources-injector-dwqpx          1/1     Running   0          10m
network-resources-injector-lktz5          1/1     Running   0          10m
```

11.3.1.2. SR-IOV Operator Admission Controller Webhook について

SR-IOV Operator Admission Controller Webbook は Kubernetes Dynamic Admission Controller アプリケーションです。これは、以下の機能を提供します。

- 作成時または更新時の **SriovNetworkNodePolicy** CR の検証
- CR の作成または更新時の **priority** および **deviceType** フィールドのデフォルト値の設定による **SriovNetworkNodePolicy** CR の変更

デフォルトで、SR-IOV Operator Admission Controller Webhook は Operator によって有効にされ、すべてのマスターノードでデーモンセットとして実行されます。以下は、3 つのマスターノードを持つクラスターで実行される Operator Admission Controller Webhook Pod の例です。

```
$ oc get pods -n openshift-sriov-network-operator
```

出力例

```
NAME                                      READY   STATUS    RESTARTS   AGE
operator-webhook-9jkw6                    1/1     Running   0          16m
operator-webhook-kbr5p                    1/1     Running   0          16m
operator-webhook-rpfrl                    1/1     Running   0          16m
```

11.3.1.3. カスタムノードセレクターについて

SR-IOV Network Config デーモンは、クラスターノード上の SR-IOV ネットワークデバイスを検出し、設定します。デフォルトで、これはクラスター内のすべての **worker** ノードにデプロイされます。ノードラベルを使用して、SR-IOV Network Config デーモンが実行するノードを指定できます。

11.3.1.4. Network Resources Injector の無効化または有効化

デフォルトで有効にされている Network Resources Injector を無効にするか、または有効にするには、以下の手順を実行します。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- SR-IOV Operator がインストールされていること。

手順

- **enableInjector** フィールドを設定します。**<value>** を **false** に置き換えて機能を無効にするか、または **true** に置き換えて機能を有効にします。

```
$ oc patch sriovoperatorconfig default \
  --type=merge -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableInjector": <value> } }'
```

11.3.1.5. SR-IOV Operator Admission Controller Webhook の無効化または有効化

デフォルトで有効にされている なっている受付コントローラー Webhook を無効にするか、または有効にするには、以下の手順を実行します。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- SR-IOV Operator がインストールされていること。

手順

- **enableOperatorWebhook** フィールドを設定します。**<value>** を **false** に置き換えて機能を無効するか、**true** に置き換えて機能を有効にします。

```
$ oc patch sriovoperatorconfig default --type=merge \
  -n openshift-sriov-network-operator \
  --patch '{ "spec": { "enableOperatorWebhook": <value> } }'
```

11.3.1.6. SRIOV Network Config Daemon のカスタム NodeSelector の設定

SR-IOV Network Config デーモンは、クラスターノード上の SR-IOV ネットワークデバイスを検出し、設定します。デフォルトで、これはクラスター内のすべての **worker** ノードにデプロイされます。ノードラベルを使用して、SR-IOV Network Config デーモンが実行するノードを指定できます。

SR-IOV Network Config デーモンがデプロイされるノードを指定するには、以下の手順を実行します。

重要

configDaemonNodeSelector フィールドを更新する際に、SR-IOV Network Config デーモンがそれぞれの選択されたノードに再作成されます。デーモンが再作成されている間、クラスターのユーザーは新規の SR-IOV Network ノードポリシーを適用したり、新規の SR-IOV Pod を作成したりできません。

手順

- Operator のノードセレクターを更新するには、以下のコマンドを入力します。

```
$ oc patch sriovoperatorconfig default --type=json \
  -n openshift-sriov-network-operator \
  --patch '[{
      "op": "replace",
      "path": "/spec/configDaemonNodeSelector",
      "value": {<node-label>}
    }]'
```

以下の例のように、**<node-label>** を適用するラベルに置き換えます: **"node-role.kubernetes.io/worker": ""**

11.3.2. 次のステップ

- SR-IOV ネットワークデバイスの設定

11.4. SR-IOV ネットワークデバイスの設定

クラスターで Single Root I/O Virtualization (SR-IOV) デバイスを設定できます。

11.4.1. SR-IOV ネットワークノード設定オブジェクト

SriovNetworkNodePolicy オブジェクトを定義することで、ノードの SR-IOV ネットワークデバイス設定を指定します。オブジェクトは **sriovnetwork.openshift.io** API グループの一部です。

以下の YAML は **SriovNetworkNodePolicy** オブジェクトについて説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: <sriov_resource_name> (3)
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" (4)
  priority: <priority> (5)
  mtu: <mtu> (6)
  numVfs: <num> (7)
  nicSelector: (8)
    vendor: "<vendor_code>" (9)
    deviceID: "<device_id>" (10)
    pfNames: ["<pf_name>", ...] (11)
    rootDevices: ["<pci_bus_id>", "..."] (12)
  deviceType: <device_type> (13)
  isRdma: false (14)
```

(1) CR オブジェクトの名前。

2. SR-IOV Operator がインストールされている namespace。

3. SR-IOV デバイスプラグインのリソース名。1 つのリソース名に複数の **SriovNetworkNodePolicy** オブジェクトを作成できます。

4. 設定されたノードを選択するノードセレクター。選択したノード上の SR-IOV ネットワークデバイスのみが設定されます。SR-IOV Container Network Interface (CNI) プラグインおよびデバイスプラグインは、選択したノードにのみデプロイされます。

5. オプション: **0** から **99** までの整数値。数値が小さいほど優先度が高くなります。したがって、**10** は **99** よりも優先度が高くなります。デフォルト値は **99** です。

6. オプション: 仮想機能 (VF) の最大転送単位 (MTU)。MTU の最大値は NIC モデルによって異なります。

7. SR-IOV 物理ネットワークデバイス用に作成する仮想機能 (VF) の数。Intel Network Interface Card (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **128** よりも大きくすることはできません。

8. **nicSelector** マッピングは、Operator が設定するデバイスを選択します。すべてのパラメーターの値を指定する必要はありません。意図せずにデバイスを選択しないように、ネットワークデバイスを極めて正確に特定することが推奨されます。**rootDevices** を指定する場合、**vendor**、**deviceID**、または **pfName** の値も指定する必要があります。**pfNames** および **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスをポイントすることを確認します。

9. オプション: SR-IOV ネットワークデバイスのベンダー 16 進コード。許可される値は **8086** および **15b3** のみになります。

10. オプション: SR-IOV ネットワークデバイスのデバイス 16 進コード。許可される値は **158b**、**1015**、および **1017** のみになります。

11. オプション: 1 つ以上のデバイスの物理機能 (PF) 名の配列。

12. デバイスの PF 用の 1 つ以上の PCI バスアドレスの配列。以下の形式でアドレスを指定します: **0000:02:00.1**

13. オプション: 仮想機能 (VF) のドライバータイプ。許可される値は **netdevice** および **vfio-pci** のみです。デフォルト値は **netdevice** です。

> **注記**
>
> Mellanox カードをベアメタルノードの Data Plane Development Kit (DPDK) モードで機能させるには、**netdevice** ドライバータイプを使用し、**isRdma** を **true** に設定します。

14. オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうか。デフォルト値は **false** です。

> **注記**
>
> **isRDMA** パラメーターが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。

11.4.1.1. SR-IOV デバイスの仮想機能 (VF) パーティション設定

Virtual Function (VF) を同じ物理機能 (PF) から複数のリソースプールに分割する必要がある場合があります。たとえば、VF の一部をデフォルトドライバーで読み込み、残りの VF を **vfio-pci** ドライバーで読み込む必要がある場合などです。このようなデプロイメントでは、SriovNetworkNodePolicy カスタムリソース (CR) の **pfNames** セレクターは、以下の形式を使用してプールの VF の範囲を指定するために使用できます: **<pfname>#<first_vf>-<last_vf>**

たとえば、以下の YAML は、VF が **2** から **7** まである **netpf0** という名前のインターフェイスのセレクターを示します。

```
pfNames: ["netpf0#2-7"]
```

- **netpf0** は PF インターフェイス名です。
- **2** は、範囲に含まれる最初の VF インデックス (0 ベース) です。
- **7** は、範囲に含まれる最後の VF インデックス (0 ベース) です。

以下の要件を満たす場合、異なるポリシー CR を使用して同じ PF から VF を選択できます。

- **numVfs** の値は、同じ PF を選択するポリシーで同一である必要があります。
- VF インデックスは、**0** から **<numVfs>-1** の範囲にある必要があります。たとえば、**numVfs** が **8** に設定されているポリシーがある場合、**<first_vf>** の値は **0** よりも小さくすることはできず、**<last_vf>** は **7** よりも大きくすることはできません。
- 異なるポリシーの VF の範囲は重複しないようにしてください。
- **<first_vf>** は **<last_vf>** よりも大きくすることはできません。

以下の例は、SR-IOV デバイスの NIC パーティション設定を示しています。

ポリシー **policy-net-1** は、デフォルトの VF ドライバーと共に PF **netpf0** の VF **0** が含まれるリソースプール **net-1** を定義します。ポリシー **policy-net-1-dpdk** は、**vfio** VF ドライバーと共に PF **netpf0** の VF **8** から **15** までが含まれるリソースプール **net-1-dpdk** を定義します。

ポリシー **policy-net-1**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#0-0"]
  deviceType: netdevice
```

ポリシー **policy-net-1-dpdk**:

```
apiVersion: sriovnetwork.openshift.io/v1
```

```
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1-dpdk
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#8-15"]
  deviceType: vfio-pci
```

11.4.2. SR-IOV ネットワークデバイスの設定

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition を OpenShift Container Platform に追加します。SR-IOV ネットワークデバイスは、SriovNetworkNodePolicy カスタムリソース (CR) を作成して設定できます。

注記

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。

設定の変更が適用されるまでに数分かかる場合があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- SR-IOV Network Operator がインストールされている。
- ドレイン (解放) されたノードからエビクトされたワークロードを処理するために、クラスター内に利用可能な十分なノードがあること。
- SR-IOV ネットワークデバイス設定についてコントロールプレーンノードを選択していないこと。

手順

1. **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **<name>-sriov-node-network.yaml** ファイルに保存します。**<name>** をこの設定の名前に置き換えます。
2. **SriovNetworkNodePolicy** オブジェクトを作成します。

   ```
   $ oc create -f <name>-sriov-node-network.yaml
   ```

   ここで、**<name>** はこの設定の名前を指定します。

   設定の更新が適用された後に、**sriov-network-operator** namespace のすべての Pod が **Running** ステータスに移行します。

3. SR-IOV ネットワークデバイスが設定されていることを確認するには、以下のコマンドを実行します。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

11.4.3. 次のステップ

- SR-IOV ネットワーク割り当ての設定

11.5. SR-IOV イーサネットネットワーク割り当ての設定

クラスター内の Single Root I/O Virtualization (SR-IOV) デバイスのイーサネットネットワーク割り当てを設定できます。

11.5.1. イーサネットデバイス設定オブジェクト

イーサネットネットワークデバイスは、**SriovNetwork** オブジェクトを定義して設定できます。

以下の YAML は **SriovNetwork** オブジェクトについて説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: <sriov_resource_name> (3)
  networkNamespace: <target_namespace> (4)
  vlan: <vlan> (5)
  spoofChk: "<spoof_check>" (6)
  ipam: |- (7)
    {}
  linkState: <link_state> (8)
  maxTxRate: <max_tx_rate> (9)
  minTxRate: <min_tx_rate> (10)
  vlanQoS: <vlan_qos> (11)
  trust: "<trust_vf>" (12)
  capabilities: <capabilities> (13)
```

1. オブジェクトの名前。SR-IOV Network Operator は、同じ名前を持つ **NetworkAttachmentDefinition** オブジェクトを作成します。
2. SR-IOV Network Operator がインストールされている namespace を指定します。
3. この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーターの値。
4. **SriovNetwork** オブジェクトのターゲット namespace。ターゲット namespace の Pod のみを追加ネットワークに割り当てることができます。

5. オプション: 追加ネットワークの仮想 LAN (VLAN) ID。整数値は **0** から **4095** である必要があります。デフォルト値は **0** です。

6. オプション: VF の spoof チェックモード。許可される値は、文字列の **"on"** および **"off"** です。

> **重要**
>
> 指定する値を引用符で囲む必要があります。そうしないと、オブジェクトは SR-IOV ネットワーク Operator によって拒否されます。

7. YAML ブロックスケーラーとしての IPAM CNI プラグインの設定オブジェクトプラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

8. オプション: Virtual Function (VF) のリンク状態。許可される値は、**enable**、**disable**、および **auto** です。

9. オプション: VF の最大伝送レート (Mbps)。

10. オプション: VF の最小伝送レート (Mbps)。この値は、最大伝送レート以下である必要があります。

> **注記**
>
> Intel NIC は **minTxRate** パラメーターをサポートしません。詳細は、BZ#1772847 を参照してください。

11. オプション: VF の IEEE 802.1p 優先度レベル。デフォルト値は **0** です。

12. オプション: VF の信頼モード。許可される値は、文字列の **"on"** および **"off"** です。

> **重要**
>
> 指定する値を引用符で囲む必要があります。囲まないと、SR-IOV Network Operator はオブジェクトを拒否します。

13. オプション: この追加ネットワークに設定する機能。IP アドレスのサポートを有効にするには、**"{ "ips": true }"** を指定できます。または、MAC アドレスのサポートを有効にするには **"{ "mac": true }"** を指定します。

11.5.1.1. IPAM CNI プラグインの設定

IPAM Container Network Interface (CNI) プラグインは、他の CNI プラグインに IP アドレス管理 (IPAM) を提供します。

IP アドレスの割り当てには、以下の方法を使用できます。

- 静的割り当て。
- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

11.5.1.1.1. 静的 IP アドレス割り当ての設定

以下の JSON は、静的 IP アドレスの割り当ての設定について説明しています。

静的割り当ての設定

```
{
  "ipam": {
    "type": "static",
    "addresses": [ (1)
      {
        "address": "<address>", (2)
        "gateway": "<gateway>" (3)
      }
    ],
    "routes": [ (4)
      {
        "dst": "<dst>", (5)
        "gw": "<gw>" (6)
      }
    ],
    "dns": { (7)
      "nameservers": ["<nameserver>"], (8)
      "domain": "<domain>", (9)
      "search": ["<search_domain>"] (10)
    }
  }
}
```

1. 仮想インターフェイスに割り当てる IP アドレスを記述する配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
2. 指定する IP アドレスおよびネットワーク接頭辞。たとえば、**10.10.21.10/24** を指定すると、追加のネットワークに IP アドレスの **10.10.21.10** が割り当てられ、ネットマスクは **255.255.255.0** になります。
3. egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。
4. Pod 内で設定するルートを記述する配列。
5. CIDR 形式の IP アドレス範囲 (**192.168.17.0/24**、またはデフォルトルートの **0.0.0.0/0**)。
6. ネットワークトラフィックがルーティングされるゲートウェイ。
7. オプション: DNS 設定。
8. DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
9. ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが **example.com** に設定されている場合、**example-host** の DNS ルックアップクエリーは **example-host.example.com** として書き換えられます。
10. DNS ルックアップのクエリー時に非修飾ホスト名に追加されるドメイン名の配列 (例: **example-host**)。

11.5.1.1.2. 動的 IP アドレス割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

SR-IOV ネットワーク Operator は DHCP サーバーデプロイメントを作成しません。Cluster Network Operator は最小限の DHCP サーバーデプロイメントを作成します。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
```

DHCP 割り当ての設定

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

11.5.1.1.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の JSON は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

Whereabouts 割り当て設定

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "<range>", (1)
    "exclude": ["<exclude_part>, ..."], (2)
  }
}
```

1. IP アドレスと範囲を CIDR 表記で指定します。IP アドレスは、この範囲内のアドレスから割り当てられます。
2. オプション: CIDR 表記で IP アドレスおよび範囲の一覧を指定します。除外されたアドレス範囲内の IP アドレスは割り当てられません。

11.5.1.1.4. 静的 IP アドレス割り当ての設定例

静的 IP アドレスの割り当てに IPAM を設定することができます。

```
{
  "ipam": {
    "type": "static",
      "addresses": [
        {
          "address": "191.168.1.7"
        }
      ]
  }
}
```

11.5.1.1.5. DHCP を使用した動的 IP アドレス割り当ての設定例

DHCP に IPAM を設定できます。

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

11.5.1.1.6. Whereabouts を使用した動的 IP アドレス割り当ての設定例

Whereabouts を使用するように IPAM を設定できます。

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
```

```
    ]
  }
}
```

11.5.2. SR-IOV の追加ネットワークの設定

SriovNetwork オブジェクト を作成して、SR-IOV ハードウェアを使用する追加のネットワークを設定できます。**SriovNetwork** オブジェクトの作成時に、SR-IOV Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

注記

SriovNetwork オブジェクトが **running** 状態の Pod に割り当てられている場合、これを変更したり、削除したりしないでください。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **SriovNetwork** オブジェクトを作成してから、YAML を **<name>.yaml** ファイルに保存します。**<name>** はこの追加ネットワークの名前になります。オブジェクト仕様は以下の例のようになります。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
```

2. オブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc create -f <name>.yaml
```

ここで、**<name>** は追加ネットワークの名前を指定します。

3. オプション: 以下のコマンドを実行して、直前の手順で作成した **SriovNetwork** オブジェクトに関連付けられた **NetworkAttachmentDefinition** オブジェクトが存在することを確認するには、以下のコマンドを入力します。**<namespace>** を **SriovNetwork** オブジェクトで指定した networkNamespace に置き換えます。

```
$ oc get net-attach-def -n <namespace>
```

11.5.3. 次のステップ

- Pod の SR-IOV の追加ネットワークへの追加

11.5.4. 関連情報

- SR-IOV ネットワークデバイスの設定

11.6. POD の SR-IOV の追加ネットワークへの追加

Pod を既存の Single Root I/O Virtualization (SR-IOV) ネットワークに追加できます。

11.6.1. ネットワーク割り当てのランタイム設定

Pod を追加のネットワークに割り当てる場合、ランタイム設定を指定して Pod の特定のカスタマイズを行うことができます。たとえば、特定の MAC ハードウェアアドレスを要求できます。

Pod 仕様にアノテーションを設定して、ランタイム設定を指定します。アノテーションキーは **k8s.v1.cni.cncf.io/networks** で、ランタイム設定を記述する JSON オブジェクトを受け入れます。

11.6.1.1. イーサネットベースの SR-IOV 割り当てのランタイム設定

以下の JSON は、イーサネットベースの SR-IOV ネットワーク割り当て用のランタイム設定オプションを説明しています。

```
[
  {
    "name": "<name>", (1)
    "mac": "<mac_address>", (2)
    "ips": ["<cidr_range>"] (3)
  }
]
```

1. SR-IOV ネットワーク割り当て定義 CR の名前。
2. オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの MAC アドレス。この機能を使用するには、**SriovNetwork** オブジェクトで **{ "mac": true }** も指定する必要があります。
3. オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレス。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovNetwork** オブジェクトで **{ "ips": true }** も指定する必要があります。

ランタイム設定の例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
```

```
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "net1",
          "mac": "20:04:0f:f1:88:01",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

11.6.2. Pod の追加ネットワークへの追加

Pod を追加のネットワークに追加できます。Pod は、デフォルトネットワークで通常のクラスター関連のネットワークトラフィックを継続的に送信します。

Pod が作成されると、追加のネットワークが割り当てられます。ただし、Pod がすでに存在する場合は、追加のネットワークをこれに割り当てることはできません。

Pod が追加ネットワークと同じ namespace にあること。

注記

ネットワークの割り当てが SR-IOV Network Operator によって管理される場合、SR-IOV Network Resource Injector は **resource** フィールドを **Pod** オブジェクトに自動的に追加します。

重要

Deployment オブジェクトまたは **ReplicationController** オブジェクトの SR-IOV ハードウェアネットワークを指定する場合、 **NetworkAttachmentDefinition** オブジェクトの namespace を指定する必要があります。詳細は、以下の BZ#1846333 および BZ#1840962 のバグを参照してください。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- クラスターにログインする。
- SR-IOV Operator のインストール。
- Pod を割り当てる **SriovNetwork** オブジェクトを作成します。

手順

1. アノテーションを **Pod** オブジェクトに追加します。以下のアノテーション形式のいずれかのみを使用できます。
   a. カスタマイズせずに追加ネットワークを割り当てるには、以下の形式でアノテーションを追加します。**<network>** を、Pod に関連付ける追加ネットワークの名前に置き換えます。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] (1)
```

(1) 複数の追加ネットワークを指定するには、各ネットワークをコンマで区切ります。コンマの間にはスペースを入れないでください。同じ追加ネットワークを複数回指定した場合、Pod は複数のネットワークインターフェイスをそのネットワークに割り当てます。

b. カスタマイズして追加のネットワークを割り当てるには、以下の形式でアノテーションを追加します。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>", (1)
          "namespace": "<namespace>", (2)
          "default-route": ["<default-route>"] (3)
        }
      ]
```

(1) **NetworkAttachmentDefinition** オブジェクトによって定義される追加のネットワークの名前を指定します。

(2) **NetworkAttachmentDefinition** オブジェクトが定義される namespace を指定します。

(3) オプション: **192.168.17.1** などのデフォルトルートのオーバーライドを指定します。

2. Pod を作成するには、以下のコマンドを入力します。**<name>** を Pod の名前に置き換えます。

```
$ oc create -f <name>.yaml
```

3. オプション: アノテーションが **Pod** CR に存在することを確認するには、**<name>** を Pod の名前に置き換えて、以下のコマンドを入力します。

```
$ oc get pod <name> -o yaml
```

以下の例では、**example-pod** Pod が追加ネットワークの **net1** に割り当てられています。

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status: |- (1)
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
```

```
            "ips": [
                "10.128.2.14"
            ],
            "default": true,
            "dns": {}
        },{
            "name": "macvlan-bridge",
            "interface": "net1",
            "ips": [
                "20.2.2.100"
            ],
            "mac": "22:2f:60:a5:f8:00",
            "dns": {}
        }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

1. **k8s.v1.cni.cncf.io/networks-status** パラメーターは、オブジェクトの JSON 配列です。各オブジェクトは、Pod に割り当てられる追加のネットワークのステータスについて説明します。アノテーションの値はプレーンテキストの値として保存されます。

11.6.3. Non-Uniform Memory Access (NUMA) で配置された SR-IOV Pod の作成

NUMA で配置された SR-IOV Pod は、**restricted** または **single-numa-node** Topology Manager ポリシーで同じ NUMA ノードから割り当てられる SR-IOV および CPU リソースを制限することによって作成できます。

前提条件

- OpenShift CLI (**oc**) をインストールすること。
- LatencySensitive プロファイルを有効にし、CPU マネージャーのポリシーを **static** に設定する。

手順

1. 以下の SR-IOV Pod 仕様を作成してから、YAML を **<name>-sriov-pod.yaml** ファイルに保存します。**<name>** をこの Pod の名前に置き換えます。
   以下の例は、SR-IOV Pod 仕様を示しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: <name> (1)
spec:
  containers:
  - name: sample-container
    image: <image> (2)
```

```
command: ["sleep", "infinity"]
resources:
  limits:
    memory: "1Gi" (3)
    cpu: "2" (4)
  requests:
    memory: "1Gi"
    cpu: "2"
```

1. **<name>** を、SR-IOV ネットワーク割り当て定義 CR の名前に置き換えます。
2. **<image>** を **sample-pod** イメージの名前に置き換えます。
3. Guaranteed QoS を指定して SR-IOV Pod を作成するには、**メモリー要求** に等しい **メモリー制限** を設定します。
4. Guaranteed QoS を指定して SR-IOV Pod を作成するには、**cpu 要求** に等しい **cpu 制限** を設定します。

2. 以下のコマンドを実行して SR-IOV Pod のサンプルを作成します。

```
$ oc create -f <filename> (1)
```

1. **<filename>** を、先の手順で作成したファイルの名前に置き換えます。

3. **sample-pod** が Guaranteed QoS を指定して設定されていることを確認します。

```
$ oc describe pod sample-pod
```

4. **sample-pod** が排他的 CPU を指定して割り当てられていることを確認します。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. **sample-pod** に割り当てられる SR-IOV デバイスと CPU が同じ NUMA ノード上にあることを確認します。

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

11.6.4. 関連情報

- SR-IOV イーサネットネットワーク割り当ての設定

11.7. 高パフォーマンスのマルチキャストの使用

Single Root I/O Virtualization (SR-IOV) ハードウェアネットワーク上でマルチキャストを使用できます。

11.7.1. 高パフォーマンスのマルチキャストの設定

OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーは、デフォルトネットワーク上の Pod 間のマルチキャストをサポートします。これは低帯域幅の調整またはサービ

スの検出での使用に最も適しており、高帯域幅のアプリケーションには適していません。インターネットプロトコルテレビ (IPTV) やマルチポイントビデオ会議など、ストリーミングメディアなどのアプリケーションでは、Single Root I/O Virtualization (SR-IOV) ハードウェアを使用してネイティブに近いパフォーマンスを提供できます。

マルチキャストに追加の SR-IOV インターフェイスを使用する場合:

- マルチキャストパッケージは、追加の SR-IOV インターフェイス経由で Pod によって送受信される必要があります。
- SR-IOV インターフェイスに接続する物理ネットワークは、OpenShift Container Platform で制御されないマルチキャストルーティングとトポロジーを判別します。

11.7.2. マルチキャストでの SR-IOV インターフェイスの使用

以下の手順では、サンプルのマルチキャスト用の SR-IOV インターフェイスを作成します。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

1. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ['ens803f0']
    rootDevices: ['0000:86:00.0']
```

2. **SriovNetwork** オブジェクトを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | (1)
    {
      "type": "host-local", (2)
      "subnet": "10.56.217.0/24",
```

```
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {"dst": "224.0.0.0/5"},
        {"dst": "232.0.0.0/5"}
      ],
      "gateway": "10.56.217.1"
    }
  resourceName: example
```

1 2 DHCP を IPAM として設定する選択をした場合は、DHCP サーバー経由でデフォルトルート (**224.0.0.0/5** および **232.0.0.0/5**) をプロビジョニングするようにしてください。これにより、デフォルトのネットワークプロバイダーによって設定された静的なマルチキャストルートが上書きされます。

3. マルチキャストアプリケーションで Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] 1
    command: [ "sleep", "infinity"]
```

1 **NET_ADMIN** 機能は、アプリケーションがマルチキャスト IP アドレスを SR-IOV インターフェイスに割り当てる必要がある場合にのみ必要です。それ以外の場合は省略できます。

11.8. DPDK および RDMA モードでの仮想機能 (VF) の使用

Single Root I/O Virtualization (SR-IOV) ネットワークハードウェアは、Data Plane Development Kit (DPDK) および Remote Direct Memory Access (RDMA) で利用できます。

11.8.1. DPDK および RDMA モードでの仮想機能 (VF) の使用例

重要

Data Plane Development Kit (DPDK) はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

重要

Remote Direct Memory Access (RDMA) はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

11.8.2. 前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- SR-IOV Network Operator がインストールされていること。

11.8.3. Intel NIC を使用した DPDK モードでの仮想機能 (VF) の使用例

手順

1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **intel-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
```

```
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: vfio-pci (1)
```

1. 仮想機能 (VF) のドライバータイプを **vfio-pci** に指定します。

> **注記**
>
> **SriovNetworkNodePolicy** の各オプションに関する詳細は、**SR-IOV ネットワークデバイスの設定** セクションを参照してください。
>
> **SriovNetworkNodePolicy** オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。
>
> 設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f intel-dpdk-node-policy.yaml
```

3. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **intel-dpdk-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: "{}" (1)
  vlan: <vlan>
  resourceName: intelnics
```

1. IPAM CNI プラグインの空のオブジェクト **"{}"** を指定します。DPDK はユーザー空間モードで機能し、IP アドレスは必要ありません。

> **注記**
>
> **SriovNetwork** の各オプションに関する詳細は、**SR-IOV の追加ネットワークの設定** セクションを参照してください。

4. 以下のコマンドを実行して、**SriovNetwork** オブジェクトを作成します。

```
$ oc create -f intel-dpdk-network.yaml
```

5. 以下の **Pod** 仕様を作成してから、YAML を **intel-dpdk-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> (1)
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> (2)
    securityContext:
     capabilities:
        add: ["IPC_LOCK"] (3)
    volumeMounts:
    - mountPath: /dev/hugepages (4)
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1" (5)
        memory: "1Gi"
        cpu: "4" (6)
        hugepages-1Gi: "4Gi" (7)
      requests:
        openshift.io/intelnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. **SriovNetwork** オブジェクトの **intel-dpdk-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を **Pod** 仕様および **SriovNetowrk** オブジェクトの両方で変更します。

2. アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。

3. コンテナー内の hugepage メモリーを割り当てるためにアプリケーションが必要とする **IPC_LOCK** 機能を指定します。

4. hugepage ボリュームを、**/dev/hugepages** の下にある DPDK Pod にマウントします。hugepage ボリュームは、medium が **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5. オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースインジェクターによって自動的に追加されます。SR-IOV ネットワークリソースインジェクターは、SR-IOV Operator によって管理される受付コントローラーコンポーネントです。これはデフォルトで有効にされており、デフォルト **SriovOperatorConfig** CR で **enableInjector** オプションを **false** に設定して無効にすることができます。

6. CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS

7. hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。たとえば、カーネル引数 **default_hugepagesz=1GB**、**hugepagesz=1G** および **hugepages=16** を追加すると、**16*1Gi** hugepage がシステムの起動時に割り当てられます。

6. 以下のコマンドを実行して DPDK Pod を作成します。

```
$ oc create -f intel-dpdk-pod.yaml
```

11.8.4. Mellanox NIC を使用した DPDK モードでの仮想機能 (VF) の使用例

手順

1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **mlx-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

1. SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。Mellanox カードに許可される値は **1015**、**1017** です。

2. Virtual Function (VF) のドライバータイプを **netdevice** に指定します。Mellanox SR-IOV VF は、**vfio-pci** デバイスタイプを使用せずに DPDK モードで機能します。VF デバイスは、コンテナー内のカーネルネットワークインターフェイスとして表示されます。

3. RDMA モードを有効にします。これは、DPDK モードで機能するために Mellanox カードで必要とされます。

注記

SriovNetworkNodePolicy の各オプションに関する詳細は、**SR-IOV ネットワークデバイスの設定** セクションを参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f mlx-dpdk-node-policy.yaml
```

3. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **mlx-dpdk-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- (1)
    ...
  vlan: <vlan>
  resourceName: mlxnics
```

(1) IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

注記

SriovNetwork の各オプションに関する詳細は、**SR-IOV の追加ネットワークの設定** セクションを参照してください。

4. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f mlx-dpdk-network.yaml
```

5. 以下の **Pod** 仕様を作成してから、YAML を **mlx-dpdk-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> (1)
  annotations:
```

```
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> (2)
    securityContext:
     capabilities:
        add: ["IPC_LOCK","NET_RAW"] (3)
    volumeMounts:
    - mountPath: /dev/hugepages (4)
      name: hugepage
    resources:
      limits:
        openshift.io/mlxnics: "1" (5)
        memory: "1Gi"
        cpu: "4" (6)
        hugepages-1Gi: "4Gi" (7)
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. **SriovNetwork** オブジェクトの **mlx-dpdk-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を **Pod** 仕様および **SriovNetowrk** オブジェクトの両方で変更します。

2. アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。

3. コンテナー内の hugepage メモリーを割り当てるためにアプリケーションが必要とする **IPC_LOCK** 機能、およびアプリケーションがネットワークインターフェイスにアクセスするために **NET_RAW** を指定します。

4. hugepage ボリュームを、**/dev/hugepages** の下にある DPDK Pod にマウントします。hugepage ボリュームは、medium が **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5. オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースインジェクターによって自動的に追加されます。SR-IOV ネットワークリソースインジェクターは、SR-IOV Operator によって管理される受付コントローラーコンポーネントです。これはデフォルトで有効にされており、デフォルト **SriovOperatorConfig** CR で **enableInjector** オプションを **false** に設定して無効にすることができます。

6. CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。

7. hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り

6. 以下のコマンドを実行して DPDK Pod を作成します。

```
$ oc create -f mlx-dpdk-pod.yaml
```

11.8.5. Mellanox NIC を使った RDMA モードでの仮想機能 (VF) の例

RoCE (RDMA over Converged Ethernet) は、OpenShift Container Platform で RDMA を使用する場合に唯一サポートされているモードです。

手順

1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **mlx-rdma-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

(1) SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。Mellanox カードに許可される値は **1015**、**1017** です。

(2) Virtual Function (VF) のドライバータイプを **netdevice** に指定します。

(3) RDMA モードを有効にします。

注記

SriovNetworkNodePolicy の各オプションに関する詳細は、**SR-IOV ネットワークデバイスの設定** セクションを参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分にあることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f mlx-rdma-node-policy.yaml
```

3. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **mlx-rdma-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-rdma-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- (1)
    ...
  vlan: <vlan>
  resourceName: mlxnics
```

(1) IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。

注記

SriovNetwork の各オプションに関する詳細は、**SR-IOV の追加ネットワークの設定** セクションを参照してください。

4. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f mlx-rdma-network.yaml
```

5. 以下の **Pod** 仕様を作成してから、YAML を **mlx-rdma-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  namespace: <target_namespace> (1)
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> (2)
    securityContext:
     capabilities:
        add: ["IPC_LOCK"] (3)
    volumeMounts:
    - mountPath: /dev/hugepages (4)
      name: hugepage
    resources:
      limits:
```

```
        memory: "1Gi"
        cpu: "4" (5)
        hugepages-1Gi: "4Gi" (6)
      requests:
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. **SriovNetwork** オブジェクトの **mlx-rdma-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を **Pod** 仕様および **SriovNetowrk** オブジェクトの両方で変更します。

2. アプリケーションとアプリケーションが使用する RDMA ライブラリーが含まれる RDMA イメージを指定します。

3. コンテナー内の hugepage メモリーを割り当てるためにアプリケーションが必要とする **IPC_LOCK** 機能を指定します。

4. hugepage ボリュームを、**/dev/hugepages** の下にある RDMA Pod にマウントします。hugepage ボリュームは、medium が **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。

5. CPU の数を指定します。RDMA Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。

6. hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、RDMA Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。

6. 以下のコマンドを実行して RDMA Pod を作成します。

```
$ oc create -f mlx-rdma-pod.yaml
```

第12章 OPENSHIFT SDN デフォルト CNI ネットワークプロバイダー

12.1. OPENSHIFT SDN デフォルト CNI ネットワークプロバイダーについて

OpenShift Container Platform は、Software Defined Networking (SDN) アプローチを使用して、クラスターのネットワークを統合し、OpenShift Container Platform クラスターの Pod 間の通信を可能にします。OpenShift SDN により、このような Pod ネットワークが確立され、メンテナーンスされます。OpenShift SDN は Open vSwitch (OVS) を使用してオーバーレイネットワークを設定します。

12.1.1. OpenShift SDN ネットワーク分離モード

OpenShift SDN では以下のように、Pod ネットワークを設定するための SDN モードを 3 つ提供します。

- **ネットワークポリシー**モードは、プロジェクト管理者が **NetworkPolicy** オブジェクトを使用して独自の分離ポリシーを設定することを可能にします。ネットワークポリシーは、OpenShift Container Platform 4.5 のデフォルトモードです。
- **マルチテナント** モードは、Pod およびサービスのプロジェクトレベルの分離を可能にします。異なるプロジェクトの Pod は、別のプロジェクトの Pod およびサービスとパケットの送受信をすることができなくなります。プロジェクトの分離を無効にし、クラスター全体のすべての Pod およびサービスにネットワークトラフィックを送信したり、それらの Pod およびサービスからネットワークトラフィックを受信したりすることができます。
- **サブネット** モードは、すべての Pod が他のすべての Pod およびサービスと通信できる Pod ネットワークを提供します。ネットワークポリシーモードは、サブネットモードと同じ機能を提供します。

12.1.2. サポートされるデフォルトの CNI ネットワークプロバイダー機能マトリクス

OpenShift Container Platform は、OpenShift SDN と OVN-Kubernetes の 2 つのサポート対象のオプションをデフォルトの Container Network Interface (CNI) ネットワークプロバイダーに提供します。以下の表は、両方のネットワークプロバイダーの現在の機能サポートをまとめたものです。

表12.1 デフォルトの CNI ネットワークプロバイダー機能の比較

機能	OpenShift SDN	OVN-Kubernetes [1]
Egress IP	サポート対象	サポート対象外
Egress ファイアウォール [2]	サポート対象	サポート対象外
Egress ルーター	サポート対象	サポート対象外
Kubernetes ネットワークポリシー	一部サポート対象 [3]	サポート対象
マルチキャスト	サポート対象	サポート対象

1. OpenShift Container Platform 4.5 では、テクノロジープレビュー機能としてのみ利用できます。

2. egress ファイアウォールは、OpenShift SDN では egress ネットワークポリシーとしても知られています。これはネットワークポリシーの egress とは異なります。

3. egress ルールおよび一部の **ipBlock** ルールをサポートしません。

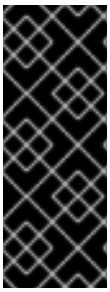
12.2. プロジェクトの EGRESS IP の設定

クラスター管理者は、OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーが 1 つ以上の egress IP アドレスをプロジェクトに割り当てるように設定できます。

12.2.1. プロジェクトの egress トラフィックについての egress IP アドレスの割り当て

プロジェクトの egress IP アドレスを設定することにより、指定されたプロジェクトからのすべての外部送信接続が同じ固定ソース IP アドレスを共有します。外部リソースは、egress IP アドレスに基づいて特定のプロジェクトからのトラフィックを認識できます。プロジェクトに割り当てられる egress IP アドレスは、トラフィックを特定の宛先に送信するために使用される egress ルーターとは異なります。

egress IP アドレスは、ノードのプライマリーネットワークインターフェイスの追加 IP アドレスとして実装され、ノードのプライマリー IP アドレスと同じサブネットにある必要があります。



重要

egress IP アドレスは、**ifcfg-eth0** などのように Linux ネットワーク設定ファイルで設定することはできません。

一部のクラウドまたは仮想マシンソリューションを使用する場合に、プライマリーネットワークインターフェイスで追加の IP アドレスを許可するには追加の設定が必要になる場合があります。

egress IP アドレスは、**NetNamespace** オブジェクトの **egressIPs** パラメーターを設定して namespace に割り当てることができます。egress IP がプロジェクトに関連付けられた後に、OpenShift SDN は 2 つの方法で Egress IP をホストに割り当てることを可能にします。

- **自動的に割り当てる** 方法では、egress IP アドレス範囲はノードに割り当てられます。
- **手動で割り当てる** 方法では、1 つ以上の egress IP アドレスの一覧がノードに割り当てられます。

egress IP アドレスを要求する namespace は、それらの egress IP アドレスをホストできるノードに一致し、egress IP アドレスはそれらのノードに割り当てられます。**egressIPs** パラメーターが **NetNamespace** オブジェクトに設定されるものの、ノードがその egress IP アドレスをホストしない場合、namespace からの egress トラフィックはドロップされます。

ノードの高可用性は自動的に実行されます。egress IP アドレスをホストするノードが到達不可能であり、egress IP アドレスをホストできるノードがある場合、egress IP アドレスは新規ノードに移行します。到達不可能なノードが再びオンラインに戻ると、ノード間で egress IP アドレスのバランスを図るために egress IP アドレスは自動的に移行します。

重要

OpenShift SDN クラスターネットワークプロバイダーで egress IP アドレスを使用する場合、以下の制限が適用されます。

- 手動で割り当てられた egress IP アドレスと、自動的に割り当てられた egress IP アドレスは同じノードで使用することができません。
- IP アドレス範囲から egress IP アドレスを手動で割り当てる場合、その範囲を自動の IP 割り当てで利用可能にすることはできません。
- OpenShift SDN egress IP アドレス実装を使用して、複数の namespace で egress IP アドレスを共有することはできません。

注記

OpenShift SDN をマルチテナントモードで使用する場合、それらに関連付けられたプロジェクトによって別の namespace に参加している namespace と共に egress IP アドレスを使用することはできません。たとえば、**project1** および **project2** に **oc adm pod-network join-projects --to=project1 project2** コマンドを実行して参加している場合、どちらもプロジェクトも egress IP アドレスを使用できません。詳細は、BZ#1645577 を参照してください。

12.2.1.1. 自動的に割り当てられた egress IP アドレスを使用する場合の考慮事項

egress IP アドレスの自動割り当て方法を使用する場合、以下の考慮事項が適用されます。

- 各ノードの **HostSubnet** リソースの **egressCIDRs** パラメーターを設定して、ノードでホストできる egress IP アドレスの範囲を指定します。OpenShift Container Platform は、指定する IP アドレス範囲に基づいて **HostSubnet** リソースの **egressIPs** パラメーターを設定します。
- 自動割り当てモードを使用する場合、namespace ごとに単一の egress IP アドレスのみがサポートされます。

namespace の egress IP アドレスをホストするノードに到達できない場合、OpenShift Container Platform は互換性のある egress IP アドレス範囲を持つ別のノードに egress IP アドレスを再割り当てします。自動割り当て方法は、追加の IP アドレスをノードに関連付ける柔軟性のある環境にインストールされたクラスターに最も適しています。

12.2.1.2. 手動で割り当てられた egress IP アドレスを使用する場合の考慮事項

この方法は、パブリッククラウド環境にインストールされたクラスター用に推奨されます。この場合、追加の IP アドレスをノードに関連付ける上で制限がある場合があります。

egress IP アドレスに手動割り当て方法を使用する場合、以下の考慮事項が適用されます。

- 各ノードの **HostSubnet** リソースの **egressIPs** パラメーターを設定して、ノードでホストできる IP アドレスを指定します。
- namespace ごとに複数の egress IP アドレスがサポートされます。

namespace に複数の egress IP アドレスがある場合、最初の egress IP アドレスをホストするノードに到達できない場合、OpenShift Container Platform は最初の egress IP アドレスが再び到達可能になるまで、次に利用可能な egress IP アドレスの使用に自動的に切り替えます。

この方法は、パブリッククラウド環境にインストールされたクラスター用に推奨されます。この場合、追加の IP アドレスをノードに関連付ける上で制限がある場合があります。

12.2.2. namespace の自動的に割り当てられた egress IP アドレスの有効化

OpenShift Container Platform では、1 つ以上のノードで特定の namespace の egress IP アドレスの自動的な割り当てを有効にできます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 以下の JSON を使用して、**NetNamespace** オブジェクトを egress IP アドレスで更新します。

```
$ oc patch netnamespace <project_name> --type=merge -p \ (1)
 '{
   "egressIPs": [
     "<ip_address>" (2)
   ]
 }'
```

(1) プロジェクトのターゲットを指定します。

(2) 単一 egress IP アドレスを指定します。複数の IP アドレスの使用はサポートされません。

たとえば、**project1** を IP アドレスの 192.168.1.100 に、 **project2** を IP アドレスの 192.168.1.101 に割り当てるには、以下を実行します。

```
$ oc patch netnamespace project1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100"]}'
$ oc patch netnamespace project2 --type=merge -p \
  '{"egressIPs": ["192.168.1.101"]}'
```

注記

OpenShift SDN は **NetNamespace** オブジェクトを管理するため、既存の **NetNamespace** オブジェクトを変更することによってのみ変更を加えることができます。新規 **NetNamespace** オブジェクトは作成しません。

2. 以下の JSON を使用して、各ホストの **egressCIDRs** パラメーターを設定して egress IP アドレスをホストできるノードを示します。

```
$ oc patch hostsubnet <node_name> --type=merge -p \ (1)
  '{
    "egressCIDRs": [
      "<ip_address_range_1>", "<ip_address_range_2>" (2)
    ]
  }'
```

1. ノード名を指定します。
2. CIDR 形式で 1 つ以上の IP アドレス範囲を指定します。

たとえば、**node1** および **node2** を、192.168.1.0 から 192.168.1.255 の範囲で egress IP アドレスをホストするように設定するには、以下を実行します。

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
$ oc patch hostsubnet node2 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
```

OpenShift Container Platform はバランスを取りながら特定の egress IP アドレスを利用可能なノードに自動的に割り当てます。この場合、egress IP アドレス 192.168.1.100 を **node1** に、egress IP アドレス 192.168.1.101 を **node2** に割り当て、その逆も行います。

12.2.3. namespace の手動で割り当てられた egress IP アドレスの設定

OpenShift Container Platform で、1 つ以上の egress IP アドレスを namespace に関連付けることができます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 以下の JSON オブジェクトを必要な IP アドレスで指定して、**NetNamespace** オブジェクトを更新します。

```
$ oc patch netnamespace <project> --type=merge -p \ (1)
  '{
    "egressIPs": [ (2)
      "<ip_address>"
    ]
  }'
```

1. プロジェクトのターゲットを指定します。
2. 1 つ以上の egress IP アドレスを指定します。**egressIPs** パラメーターは配列です。

たとえば、**project1** プロジェクトを **192.168.1.100** の IP アドレスに割り当てるには、以下を実行します。

```
$ oc patch netnamespace project1 --type=merge \
  -p '{"egressIPs": ["192.168.1.100"]}'
```

egressIPs を異なるノードの 2 つ以上の IP アドレスに設定し、高可用性を確保することができます。複数の egress IP アドレスが設定される場合、Pod は egress の一覧にある最初の IP を使用しますが、IP アドレスをホストするノードが失敗する場合、Pod は短時間の遅延の後に一覧にある次の IP の使用に切り替えます。

注記

OpenShift SDN は **NetNamespace** オブジェクトを管理するため、既存の **NetNamespace** オブジェクトを変更することによってのみ変更を加えることができます。新規 **NetNamespace** オブジェクトは作成しません。

2. egress IP をノードホストに手動で割り当てます。**egressIPs** パラメーターを、ノードホストの **HostSubnet** オブジェクトに設定します。以下の JSON を使用して、そのノードホストに割り当てる必要のある任意の数の IP を含めることができます。

```
$ oc patch hostsubnet <node_name> --type=merge -p \ (1)
  '{
    "egressIPs": [ (2)
      "<ip_address_1>",
      "<ip_address_N>"
      ]
  }'
```

(1) ノードの名前を指定します。

(2) 1 つ以上の egress IP アドレスを指定します。**egressIPs** フィールドは配列です。

たとえば、**node1** に Egress IP **192.168.1.100**、 **192.168.1.101**、および **192.168.1.102** が設定されるように指定するには、以下を実行します。

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100", "192.168.1.101", "192.168.1.102"]}'
```

直前の例では、**project1** のすべての egress トラフィックは、指定された egress IP をホストするノードにルーティングされてから、その IP アドレスに (NAT を使用して) 接続されます。

12.3. 外部 IP アドレスへのアクセスを制御するための EGRESS ファイアウォールの設定

クラスター管理者は、OpenShift Container Platform クラスター外に出るプロジェクトのプロジェクについて、egress トラフィックを制限する egress ファイアウォールを作成できます。

12.3.1. egress ファイアウォールのプロジェクトでの機能

クラスター管理者は、 **egress ファイアウォール** を使用して、一部またはすべての Pod がクラスター内からアクセスできる外部ホストを制限できます。egress ファイアウォールポリシーは以下のシナリオをサポートします。

- Pod の接続を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
- Pod の接続をパブリックインターネットに制限し、OpenShift Container Platform クラスター外にある内部ホストへの接続を開始できないようにする。
- Pod は OpenShift Container Platform クラスター外の指定された内部サブネットまたはホストにアクセスできません。
- Pod は特定の外部ホストにのみ接続することができます。

egress ファイアウォールポリシーは、EgressNetworkPolicy カスタムリソース (CR) オブジェクトを作成し、IP アドレス範囲を CIDR 形式で指定するか、または DNS 名を指定して設定します。たとえば、指定された IP 範囲へのあるプロジェクトへのアクセスを許可する一方で、別のプロジェクトへの同じアクセスを拒否することができます。または、アプリケーション開発者の (Python) pip mirror からの更新を制限したり、更新を承認されたソースからの更新のみに強制的に制限したりすることができます。

重要

egress ファイアウォールポリシーを設定するには、ネットワークポリシーまたはマルチテナントモードのいずれかを使用するように OpenShift SDN を設定する必要があります。

ネットワークポリシーモードを使用している場合、egress ポリシーは namespace ごとに 1 つのポリシーとのみ互換性を持ち、グローバルプロジェクトなどのネットワークを共有するプロジェクトでは機能しません。

警告

egress ファイアウォールルールは、ルーターを通過するトラフィックには適用されません。ルート CR オブジェクトを作成するパーミッションを持つユーザーは、禁止されている宛先を参照するルートを作成することにより、egress ネットワークポリシールールをバイパスできます。

12.3.1.1. egress ファイアウォールの制限

egress ファイアウォールには以下の制限があります。

- いずれのプロジェクトも複数の EgressNetworkPolicy オブジェクトを持つことができません。
- 最大 50 のルールを持つ最大 1 EgressNetworkPolicy オブジェクトはプロジェクトごとに定義できます。
- **default** プロジェクトは egress ネットワークポリシーを使用できません。
- マルチテナントモードで OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーを使用する場合、以下の制限が適用されます。
  - グローバルプロジェクトは egress ファイアウォールを使用できません。**oc adm pod-network make-projects-global** コマンドを使用して、プロジェクトをグローバルにすることができます。
  - **oc adm pod-network join-projects** コマンドを使用してマージされるプロジェクトでは、結合されたプロジェクトのいずれでも egress ファイアウォールを使用することはできません。

上記の制限のいずれかに違反すると、プロジェクトの egress ネットワークポリシーに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

12.3.1.2. egress ネットワークポリシールールのマッチング順序

egress ネットワークポリシールールは、最初から最後へと定義された順序で評価されます。Pod からの egress 接続に一致する最初のルールが適用されます。この接続では、後続のルールは無視されます。

12.3.1.3. DNS (Domain Name Server) 解決の仕組み

egress ファイアウォールポリシールールのいずれかで DNS 名を使用する場合、ドメイン名の適切な解決には、以下の制限が適用されます。

- ドメイン名の更新は、ローカルの非権威サーバーのドメインの TTL (time to live) 値に基づいてポーリングされます。
- Pod は、必要に応じて同じローカルネームサーバーからドメインを解決する必要があります。そうしない場合、egress ファイアウォールコントローラーと Pod によって認識されるドメインの IP アドレスが異なる可能性があります。ホスト名の IP アドレスが異なる場合、egress ファイアウォールは一貫して実行されないことがあります。
- egress ファイアウォールコントローラーおよび Pod は同じローカルネームサーバーを非同期にポーリングするため、Pod は egress コントローラーが実行する前に更新された IP アドレスを取得する可能性があります。これにより、競合状態が生じます。この現時点の制限により、EgressNetworkPolicy オブジェクトのドメイン名の使用は、IP アドレスの変更が頻繁に生じないドメインの場合にのみ推奨されます。

注記

egress ファイアウォールは、DNS 解決用に Pod が置かれるノードの外部インターフェイスに Pod が常にアクセスできるようにします。

ドメイン名を egress ファイアウォールで使用し、DNS 解決がローカルノード上の DNS サーバーによって処理されない場合は、Pod でドメイン名を使用している場合には DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールを追加する必要があります。

12.3.2. EgressNetworkPolicy カスタムリソース (CR) オブジェクト

以下の YAML は EgressNetworkPolicy CR オブジェクトについて説明しています。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> (1)
spec:
  egress: (2)
    ...
```

1. egress ファイアウォールポリシーの **name** を指定します。
2. 以下のセクションで説明されているように、egress ネットワークポリシールールのコレクションを指定します。

12.3.2.1. EgressNetworkPolicy ルール

以下の YAML は egress ファイアウォールルールオブジェクトについて説明しています。**egress** キーは、単一または複数のオブジェクトの配列を予想します。

```
egress:
- type: <type> (1)
  to: (2)
    cidrSelector: <cidr> (3)
    dnsName: <dns-name> (4)
```

1. ルールのタイプを指定します。値には **Allow** または **Deny** のいずれかを指定する必要があります。
2. ルールの **cidrSelector** キーまたは **dnsName** キーのいずれかの値を指定します。ルールで両方のキーを使用することはできません。
3. CIDR 形式の IP アドレス範囲を指定します。
4. ドメイン名を指定します。

12.3.2.2. EgressNetworkPolicy CR オブジェクトの例

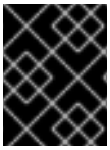
以下の例では、複数の egress ファイアウォールポリシールールを定義します。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default-rules (1)
spec:
  egress: (2)
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1. ポリシーオブジェクトの名前。
2. egress ファイアウォールポリシールールオブジェクトのコレクション。

12.3.3. egress ファイアウォールポリシーオブジェクトの作成

クラスター管理者は、プロジェクトの egress ファイアウォールポリシーオブジェクトを作成できます。



重要

プロジェクトに EgressNetworkPolicy オブジェクトがすでに定義されている場合、既存のポリシーを編集して egress ファイアウォールルールを変更する必要があります。

前提条件

- OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) をインストールしている。
- クラスター管理者としてクラスターにログインする必要があります。

手順

1. ポリシールールを作成します。
   a. **<policy-name>.yaml** ファイルを作成します。 この場合、 **<policy-name>** は egress ポリシールールを記述します。
   b. 作成したファイルで、egress ポリシーオブジェクトを定義します。
2. 以下のコマンドを入力してポリシーオブジェクトを作成します。

```
$ oc create -f <policy-name>.yaml -n <project>
```

以下の例では、新規の EgressNetworkPolicy オブジェクトが **project1** という名前のプロジェクトに作成されます。

```
$ oc create -f default-rules.yaml -n project1
```

出力例

```
egressnetworkpolicy.network.openshift.io/default-rules created
```

3. オプション: 後に変更できるように **<policy-name>.yaml** を保存します。

12.4. プロジェクトの EGRESS ファイアウォールの編集

クラスター管理者は、既存の egress ファイアウォールのネットワークトラフィックルールを変更できます。

12.4.1. EgressNetworkPolicy オブジェクトの編集

クラスター管理者は、プロジェクトの egress ファイアウォールを更新できます。

前提条件

- OpenShiftSDN ネットワークプラグインを使用するクラスター。
- OpenShift CLI (**oc**) のインストール。
- クラスター管理者としてクラスターにログインする必要があります。

手順

プロジェクトの既存の egress ネットワークポリシーオブジェクトを編集するには、以下の手順を実行します。

1. プロジェクトの EgressNetworkPolicy オブジェクトの名前を検索します。**<project>** をプロジェクトの名前に置き換えます。

```
$ oc get -n <project> egressnetworkpolicy
```

2. オプション: egress ネットワークファイアウォールの作成時に EgressNetworkPolicy オブジェクトのコピーを保存しなかった場合には、以下のコマンドを入力してコピーを作成します。

```
$ oc get -n <project> \ (1)
  egressnetworkpolicy <name> \ (2)
  -o yaml > <filename>.yaml (3)
```

(1) **<project>** をプロジェクトの名前に置き換えます。

(2) **<name>** をオブジェクトの名前に置き換えます。

(3) **<filename>** をファイルの名前に置き換え、YAML を保存します。

3. 以下のコマンドを入力し、EgressNetworkPolicy オブジェクトを置き換えます。**<filename>** を、更新された EgressNetworkPolicy オブジェクトを含むファイルの名前に置き換えます。

```
$ oc replace -f <filename>.yaml
```

12.4.2. EgressNetworkPolicy カスタムリソース (CR) オブジェクト

以下の YAML は EgressNetworkPolicy CR オブジェクトについて説明しています。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> (1)
spec:
  egress: (2)
    ...
```

(1) egress ファイアウォールポリシーの **name** を指定します。

(2) 以下のセクションで説明されているように、egress ネットワークポリシールールのコレクションを指定します。

12.4.2.1. EgressNetworkPolicy ルール

以下の YAML は egress ファイアウォールルールオブジェクトについて説明しています。**egress** キーは、単一または複数のオブジェクトの配列を予想します。

```
egress:
- type: <type> (1)
  to: (2)
    cidrSelector: <cidr> (3)
    dnsName: <dns-name> (4)
```

1. ルールのタイプを指定します。値には **Allow** または **Deny** のいずれかを指定する必要があります。
2. ルールの **cidrSelector** キーまたは **dnsName** キーのいずれかの値を指定します。ルールで両方のキーを使用することはできません。
3. CIDR 形式の IP アドレス範囲を指定します。
4. ドメイン名を指定します。

12.4.2.2. EgressNetworkPolicy CR オブジェクトの例

以下の例では、複数の egress ファイアウォールポリシールールを定義します。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default-rules (1)
spec:
  egress: (2)
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1. ポリシーオブジェクトの名前。
2. egress ファイアウォールポリシールールオブジェクトのコレクション。

12.5. プロジェクトからの EGRESS ファイアウォールの削除

クラスター管理者は、プロジェクトから egress ファイアウォールを削除して、OpenShift Container Platform クラスター外に出るプロジェクトからネットワークトラフィックについてのすべての制限を削除できます。

12.5.1. EgressNetworkPolicy オブジェクトの削除

クラスター管理者は、プロジェクトから Egress ファイアウォールを削除できます。

前提条件

- OpenShiftSDN ネットワークプラグインを使用するクラスター。
- OpenShift CLI (**oc**) のインストール。
- クラスター管理者としてクラスターにログインする必要があります。

手順

プロジェクトの egress ネットワークポリシーオブジェクトを削除するには、以下の手順を実行します。

1. プロジェクトの EgressNetworkPolicy オブジェクトの名前を検索します。**<project>** をプロジェクトの名前に置き換えます。

```
$ oc get -n <project> egressnetworkpolicy
```

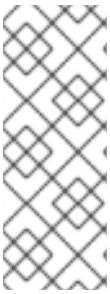
2. 以下のコマンドを入力し、EgressNetworkPolicy オブジェクトを削除します。**<project>** をプロジェクトの名前に、 **<name>** をオブジェクトの名前に置き換えます。

```
$ oc delete -n <project> egressnetworkpolicy <name>
```

12.6. EGRESS ルーター POD の使用についての考慮事項

12.6.1. Egress ルーター Pod について

OpenShift Container Platform egress ルーター Pod は、他の用途で使用されていないプライベートソース IP アドレスを使用して、指定されたリモートサーバーにトラフィックをリダイレクトします。これにより、特定の IP アドレスからのアクセスのみを許可するように設定されたサーバーにネットワークトラフィックを送信できます。



注記

egress ルーター Pod はすべての発信接続のために使用されることが意図されていません。多数の egress ルーター Pod を作成することで、ネットワークハードウェアの制限を引き上げられる可能性があります。たとえば、すべてのプロジェクトまたはアプリケーションに egress ルーター Pod を作成すると、ソフトウェアの MAC アドレスのフィルターに戻る前にネットワークインターフェイスが処理できるローカル MAC アドレス数の上限を超えてしまう可能性があります。

重要

egress ルーターイメージには Amazon AWS, Azure Cloud またはレイヤー 2 操作をサポートしないその他のクラウドプラットフォームとの互換性がありません。 それらに macvlan トラフィックとの互換性がないためです。

12.6.1.1. Egress ルーターモード

リダイレクトモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスにリダイレクトするために iptables ルールをセットアップします。予約されたソース IP アドレスを使用する必要のあるクライアント Pod は、宛先 IP に直接接続するのでなく、egress ルーターに接続するように変更される必要があります。

HTTP プロキシーモードでは、egress ルーター Pod はポート **8080** で HTTP プロキシーとして実行されます。このモードは、HTTP または HTTPS ベースのサービスと通信するクライアントの場合にのみ機能しますが、通常それらを機能させるのにクライアント Pod への多くの変更は不要です。環境変数を設定することで、数多くのプログラムは HTTP プロキシーを使用するように指示されます。

DNS プロキシーモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスに送信する TCP ベースのサービスの DNS プロキシーとして実行されます。予約されたソース IP アドレスを使用するには、クライアント Pod は、宛先 IP アドレスに直接接続するのでな

く、egress ルーター Pod に接続するように変更される必要があります。この修正により、外部の宛先でトラフィックが既知のソースから送信されているかのように処理されます。

リダイレクトモードは、HTTP および HTTPS 以外のすべてのサービスで機能します。HTTP および HTTPS サービスの場合は、HTTP プロキシーモードを使用します。IP アドレスまたはドメイン名を持つ TCP ベースのサービスの場合は、DNS プロキシーモードを使用します。

12.6.1.2. egress ルーター Pod の実装

egress ルーター Pod の設定は、初期化コンテナーで実行されます。このコンテナーは特権付きコンテキストで実行され、macvlan インターフェイスを設定して **iptables** ルールを設定できます。初期化コンテナーが **iptables** ルールの設定を終了すると、終了します。次に、egress ルーター Pod はコンテナーを実行して egress ルーターのトラフィックを処理します。使用されるイメージは、egress ルーターモードによって異なります。

環境変数は、egress-router イメージが使用するアドレスを判別します。イメージは macvlan インターフェイスを、 **EGRESS_SOURCE** をその IP アドレスとして使用し、 **EGRESS_GATEWAY** をゲートウェイの IP アドレスとして使用するように設定します。

ネットワークアドレス変換 (NAT) ルールは、TCP ポートまたは UDP ポート上の Pod のクラスター IP アドレスへの接続が **EGRESS_DESTINATION** 変数で指定される IP アドレスの同じポートにリダイレクトされるように設定されます。

クラスター内の一部のノードのみが指定されたソース IP アドレスを要求でき、指定されたゲートウェイを使用できる場合、受け入れ可能なノードを示す **nodeName** または **nodeSelector** を指定することができます。

12.6.1.3. デプロイメントに関する考慮事項

egress ルーター Pod は追加の IP アドレスおよび MAC アドレスをノードのプライマリーネットワークインターフェイスに追加します。その結果、ハイパーバイザーまたはクラウドプロバイダーを、追加のアドレスを許可するように設定する必要がある場合があります。

Red Hat OpenStack Platform (RHOSP)

OpenShift Container Platform を RHOSP でデプロイしている場合、OpenStack 環境で IP および MAC アドレスのホワイトリストを作成する必要があります。 これを行わないと、通信は失敗します。

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

Red Hat Virtualization (RHV)

RHV を使用している場合は、仮想インターフェイスカード (vNIC) に **No Network Filter** を選択する必要があります。

VMware vSphere

VMware vSphere を使用している場合は、vSphere 標準スイッチのセキュリティー保護についての VMWare ドキュメント を参照してください。vSphere Web クライアントからホストの仮想スイッチを選択して、VMWare vSphere デフォルト設定を表示し、変更します。

とくに、以下が有効にされていることを確認します。

- MAC アドレスの変更
- 偽装転送 (Forged Transit)

- 無作為別モード (Promiscuous Mode) 操作

12.6.1.4. フェイルオーバー設定

ダウンタイムを回避するには、以下の例のように **Deployment** リソースで egress ルーター Pod をデプロイできます。サンプルのデプロイメント用に新規 **Service** オブジェクトを作成するには、**oc expose deployment/egress-demo-controller** コマンドを使用します。

```
apiVersion: v1
kind: Deployment
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 (1)
  selector:
    name: egress-router
  template:
    metadata:
      name: egress-router
      labels:
        name: egress-router
      annotations:
        pod.network.openshift.io/assign-macvlan: "true"
    spec: (2)
      initContainers:
        ...
      containers:
        ...
```

1. 1 つの Pod のみが指定される egress ソース IP アドレスを使用できるため、レプリカが **1** に設定されていることを確認します。これは、単一コピーのルーターのみがノード実行されることを意味します。
2. egress ルーター Pod の **Pod** オブジェクトテンプレートを指定します。

12.6.2. 関連情報

- リダイレクトモードでの egress ルーターのデプロイ
- HTTP プロキシーモードでの egress ルーターのデプロイ
- DNS プロキシーモードでの egress ルーターのデプロイ

12.7. リダイレクトモードでの EGRESS ルーター POD のデプロイ

クラスター管理者は、トラフィックを指定された宛先 IP アドレスにリダイレクトするように設定された egress ルーター Pod をデプロイできます。

12.7.1. リダイレクトモードの egress ルーター Pod 仕様

Pod オブジェクトで egress ルーター Pod の設定を定義します。以下の YAML は、リダイレクトモードでの egress ルーター Pod の設定のフィールドについて説明しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: <egress_router>
    - name: EGRESS_GATEWAY (3)
      value: <egress_gateway>
    - name: EGRESS_DESTINATION (4)
      value: <egress_destination>
    - name: EGRESS_ROUTER_MODE
      value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift3/ose-pod
```

1. **egress-router** コンテナーを起動する前に、プライマリーネットワークインターフェイスで macvlan ネットワークインターフェイスを作成し、そのインターフェイスを Pod ネットワーク namespace に移動します。引用符を **"true"** 値の周囲に含める必要があります。プライマリーネットワークインターフェイス以外のネットワークインターフェイスで macvlan インターフェイスを作成するには、アノテーションの値を該当インターフェイスの名前に設定します。たとえば、**eth1** を使用します。

2. ノードが置かれている物理ネットワークの IP アドレスは egress ルーター Pod で使用するために予約されます。オプション: サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

3. ノードで使用されるデフォルトゲートウェイと同じ値です。

4. トラフィックの送信先となる外部サーバー。この例では、Pod の接続は **203.0.113.25** にリダイレクトされます。 ソース IP アドレスは **192.168.12.99** です。

egress ルーター Pod 仕様の例

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
```

```
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE
      value: 192.168.12.99/24
    - name: EGRESS_GATEWAY
      value: 192.168.12.1
    - name: EGRESS_DESTINATION
      value: |
        80   tcp 203.0.113.25
        8080 tcp 203.0.113.26 80
        8443 tcp 203.0.113.26 443
        203.0.113.27
    - name: EGRESS_ROUTER_MODE
      value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift3/ose-pod
```

12.7.2. egress 宛先設定形式

egress ルーター Pod がリダイレクトモードでデプロイされる場合、以下のいずれかの形式を使用してリダイレクトルールを指定できます。

- **<port> <protocol> <ip_address>**: 指定される **<port>** への着信接続が指定される **<ip_address>** の同じポートにリダイレクトされます。**<protocol>** は **tcp** または **udp** のいずれかになります。
- **<port> <protocol> <ip_address> <remote_port>**: 接続が **<ip_address>** の別の **<remote_port>** にリダイレクトされるのを除き、上記と同じになります。
- **<ip_address>**: 最後の行が単一 IP アドレスである場合、それ以外のポートの接続はその IP アドレスの対応するポートにリダイレクトされます。フォールバック IP アドレスがない場合、他のポートでの接続は拒否されます。

以下の例では、複数のルールが定義されます。

- 最初の行はローカルポート **80** から **203.0.113.25** のポート **80** にトラフィックをリダイレクトします。
- 2 番目と 3 番目の行では、ローカルポート **8080** および **8443** を、**203.0.113.26** のリモートポート **80** および **443** にリダイレクトします。
- 最後の行は、先のルールで指定されていないポートのトラフィックに一致します。

設定例

```
80   tcp 203.0.113.25
8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443
203.0.113.27
```

12.7.3. リダイレクトモードでの egress ルーター Pod のデプロイ

リダイレクトモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスにリダイレクトするために iptables ルールをセットアップします。予約されたソース IP アドレスを使用する必要のあるクライアント Pod は、宛先 IP に直接接続するのでなく、egress ルーターに接続するように変更される必要があります。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. egress ルーター Pod の作成
2. 他の Pod が egress ルーター Pod の IP アドレスを見つられるようにするには、以下の例のように、egress ルーター Pod を参照するサービスを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http
    port: 80
  - name: https
    port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにリダイレクトされます。

12.7.4. 関連情報

- ConfigMap を使用した egress ルーターの宛先マッピングの設定

12.8. HTTP プロキシーモードでの EGRESS ルーター POD のデプロイ

クラスター管理者は、トラフィックを指定された HTTP および HTTPS ベースのサービスにプロキシーするように設定された egress ルーター Pod をデプロイできます。

12.8.1. HTTP モードの egress ルーター Pod 仕様

Pod オブジェクトで egress ルーター Pod の設定を定義します。以下の YAML は、HTTP モードでの egress ルーター Pod の設定のフィールドについて説明しています。

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: <egress-router>
    - name: EGRESS_GATEWAY (3)
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: http-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/ose-egress-http-proxy
    env:
    - name: EGRESS_HTTP_PROXY_DESTINATION (4)
      value: |-
        ...
    ...
```

1. **egress-router** コンテナーを起動する前に、プライマリーネットワークインターフェイスで macvlan ネットワークインターフェイスを作成し、そのインターフェイスを Pod ネットワーク namespace に移動します。引用符を **"true"** 値の周囲に含める必要があります。プライマリーネットワークインターフェイス以外のネットワークインターフェイスで macvlan インターフェイスを作成するには、アノテーションの値を該当インターフェイスの名前に設定します。たとえば、**eth1** を使用します。

2. ノードが置かれている物理ネットワークの IP アドレスは egress ルーター Pod で使用するために予約されます。オプション: サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

3. ノードで使用されるデフォルトゲートウェイと同じ値です。

4. プロキシーの設定方法を指定する文字列または YAML の複数行文字列です。これは、init コンテナーの他の環境変数ではなく、HTTP プロキシーコンテナーの環境変数として指定されることに注意してください。

12.8.2. egress 宛先設定形式

egress ルーター Pod が HTTP プロキシーモードでデプロイされる場合、以下の形式のいずれかを使用してリダイレクトルールを指定できます。これはすべてのリモート宛先への接続を許可することを意味します。 設定の各行には、許可または拒否する接続の 1 つのグループを指定します。

- IP アドレス (例: **192.168.1.1**) は該当する IP アドレスへの接続を許可します。

- CIDR 範囲 (例: **192.168.1.0/24**) は CIDR 範囲への接続を許可します。
- ホスト名 (例: **www.example.com**) は該当ホストへのプロキシーを許可します。
- ***.** が前に付けられているドメイン名 (例: ***.example.com**) は該当ドメインおよびそのサブドメインのすべてへのプロキシーを許可します。
- 先の一致 (match) 式のいずれかの後に来る **!** は接続を拒否します。
- 最後の行が * の場合、明示的に拒否されていないすべてのものが許可されます。それ以外の場合、許可されないすべてのものが拒否されます。

* を使用してすべてのリモート宛先への接続を許可することもできます。

設定例

```
!*.example.com
!192.168.1.0/24
192.168.2.1
*
```

12.8.3. HTTP プロキシーモードでの egress ルーター Pod のデプロイ

HTTP プロキシーモードでは、egress ルーター Pod はポート **8080** で HTTP プロキシーとして実行されます。このモードは、HTTP または HTTPS ベースのサービスと通信するクライアントの場合にのみ機能しますが、通常それらを機能させるのにクライアント Pod への多くの変更は不要です。環境変数を設定することで、数多くのプログラムは HTTP プロキシーを使用するように指示されます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. egress ルーター Pod の作成
2. 他の Pod が egress ルーター Pod の IP アドレスを見つられるようにするには、以下の例のように、egress ルーター Pod を参照するサービスを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: http-proxy
    port: 8080 (1)
  type: ClusterIP
  selector:
    name: egress-1
```

(1) **http** ポートが常に **8080** に設定されていることを確認します。

3. **http_proxy** または **https_proxy** 変数を設定して、クライアント Pod (egress プロキシー Pod ではない) を HTTP プロキシーを使用するように設定します。

```
apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  containers:
    env:
    - name: http_proxy
      value: http://egress-1:8080/ (1)
    - name: https_proxy
      value: http://egress-1:8080/
    ...
```

(1) 先の手順で作成したサービス。

注記

すべてのセットアップに **http_proxy** および **https_proxy** 環境変数が必要になる訳ではありません。上記を実行しても作業用セットアップが作成されない場合は、Pod で実行しているツールまたはソフトウェアについてのドキュメントを参照してください。

12.8.4. 関連情報

- ConfigMap を使用した egress ルーターの宛先マッピングの設定

12.9. DNS プロキシーモードでの EGRESS ルーター POD のデプロイ

クラスター管理者は、トラフィックを指定された DNS 名および IP アドレスにプロキシーするように設定された egress ルーター Pod をデプロイできます。

12.9.1. DNS モードの egress ルーター Pod 仕様

Pod オブジェクトで egress ルーター Pod の設定を定義します。以下の YAML は、DNS モードでの egress ルーター Pod の設定のフィールドについて説明しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" (1)
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
```

```
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: <egress-router>
    - name: EGRESS_GATEWAY (3)
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: dns-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-dns-proxy
    securityContext:
      privileged: true
    env:
    - name: EGRESS_DNS_PROXY_DESTINATION (4)
      value: |-
        ...
    - name: EGRESS_DNS_PROXY_DEBUG (5)
      value: "1"
    ...
```

1. **egress-router** コンテナーを起動する前に、プライマリーネットワークインターフェイスで macvlan ネットワークインターフェイスを作成し、そのインターフェイスを Pod ネットワーク namespace に移動します。引用符を **"true"** 値の周囲に含める必要があります。プライマリーネットワークインターフェイス以外のネットワークインターフェイスで macvlan インターフェイスを作成するには、アノテーションの値を該当インターフェイスの名前に設定します。たとえば、**eth1** を使用します。

2. ノードが置かれている物理ネットワークの IP アドレスは egress ルーター Pod で使用するために予約されます。オプション: サブネットの長さ **/24** 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

3. ノードで使用されるデフォルトゲートウェイと同じ値です。

4. 1 つ以上のプロキシー宛先の一覧を指定します。

5. オプション: DNS プロキシーログ出力を **stdout** に出力するために指定します。

12.9.2. egress 宛先設定形式

ルーターが DNS プロキシーモードでデプロイされる場合、ポートおよび宛先マッピングの一覧を指定します。宛先には、IP アドレスまたは DNS 名のいずれかを使用できます。

egress ルーター Pod は、ポートおよび宛先マッピングを指定するために以下の形式をサポートします。

ポートおよびリモートアドレス

送信元ポートおよび宛先ホストは、2 つのフィールド形式 (**<port> <remote_address>**) を使用して指定できます。

ホストには、IP アドレスまたは DNS 名を指定できます。DNS 名を指定すると、DNS 解決が起動時に行われます。特定のホストについては、プロキシーは、宛先ホスト IP アドレスへの接続時に、宛先ホストの指定された送信元ポートに接続されます。

ポートとリモートアドレスペアの例

```
80 172.16.12.11
100 example.com
```

ポート、リモートアドレス、およびリモートポート

送信元ポート、宛先ホスト、および宛先ポートは、**<port> <remote_address> <remote_port>** の 3 つのフィールドからなる形式を使用して指定できます。

3 つのフィールド形式は、2 つのフィールドバージョンと同じように動作しますが、宛先ポートが送信元ポートとは異なる場合があります。

ポート、リモートアドレス、およびリモートポートの例

```
8080 192.168.60.252 80
8443 web.example.com 443
```

12.9.3. DNS プロキシーモードでの egress ルーター Pod のデプロイ

DNS プロキシーモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスに送信する TCP ベースのサービスの DNS プロキシーとして機能します。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. egress ルーター Pod の作成
2. egress ルーター Pod のサービスを作成します。
   a. 以下の YAML 定義が含まれる **egress-router-service.yaml** という名前のファイルを作成します。**spec.ports** を、 **EGRESS_DNS_PROXY_DESTINATION** 環境変数に先に定義したポートの一覧に設定します。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
    ...
  type: ClusterIP
  selector:
    name: egress-dns-proxy
```

以下に例を示します。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
  - name: con1
    protocol: TCP
    port: 80
    targetPort: 80
  - name: con2
    protocol: TCP
    port: 100
    targetPort: 100
  type: ClusterIP
  selector:
    name: egress-dns-proxy
```

b. サービスを作成するには、以下のコマンドを入力します。

```
$ oc create -f egress-router-service.yaml
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにプロキシー送信されます。

12.9.4. 関連情報

- ConfigMap を使用した egress ルーターの宛先マッピングの設定

12.10. 設定マップからの EGRESS ルーター POD 宛先一覧の設定

クラスター管理者は、egress ルーター Pod の宛先マッピングを指定する **ConfigMap** オブジェクトを定義できます。設定の特定の形式は、egress ルーター Pod のタイプによって異なります。形式についての詳細は、特定の egress ルーター Pod のドキュメントを参照してください。

12.10.1. 設定マップを使用した egress ルーター宛先マッピングの設定

宛先マッピングのセットのサイズが大きいか、またはこれが頻繁に変更される場合、設定マップを使用して一覧を外部で維持できます。この方法の利点は、設定マップを編集するパーミッションを **cluster-admin** 権限を持たないユーザーに委任できることです。egress ルーター Pod には特権付きコンテナーを必要とするため、**cluster-admin** 権限を持たないユーザーは Pod 定義を直接編集することはできません。

注記

egress ルーター Pod は、設定マップが変更されても自動的に更新されません。更新を取得するには、egress ルーター Pod を再起動する必要があります。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の例のように、egress ルーター Pod のマッピングデータが含まれるファイルを作成します。

```
# Egress routes for Project "Test", version 3

80   tcp 203.0.113.25

8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

空の行とコメントをこのファイルに追加できます。

2. このファイルから **ConfigMap** オブジェクトを作成します。

```
$ oc delete configmap egress-routes --ignore-not-found
```

```
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

直前のコマンドで、**egress-routes** 値は、作成する **ConfigMap** オブジェクトの名前で、 **my-egress-destination.txt** はデータの読み取り元のファイルの名前です。

3. egress ルーター Pod 定義を作成し、environment スタンザの **EGRESS_DESTINATION** フィールドに **configMapKeyRef** スタンザを指定します。

```
...
env:
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
...
```

12.10.2. 関連情報

- リダイレクトモード
- HTTP_PROXY
- DNS プロキシーモード

12.11. プロジェクトのマルチキャストの有効化

12.11.1. マルチキャストについて

IP マルチキャストを使用すると、データが多数の IP アドレスに同時に配信されます。

重要

現時点で、マルチキャストは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のソリューションとしては適していません。

OpenShift Container Platform の Pod 間のマルチキャストトラフィックはデフォルトで無効にされます。OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーを使用している場合、プロジェクトごとにマルチキャストを有効にできます。

networkpolicy 分離モードで OpenShift SDN ネットワークプラグインを使用する場合は、以下を行います。

- Pod によって送信されるマルチキャストパケットは、 **NetworkPolicy** オブジェクトに関係なく、プロジェクトの他のすべての Pod に送信されます。Pod はユニキャストで通信できない場合でもマルチキャストで通信できます。
- 1 つのプロジェクトの Pod によって送信されるマルチキャストパケットは、**NetworkPolicy** オブジェクトがプロジェクト間の通信を許可する場合であっても、それ以外のプロジェクトの Pod に送信されることはありません。

multinenant 分離モードで OpenShift SDN ネットワークプラグインを使用する場合は、以下を行います。

- Pod で送信されるマルチキャストパケットはプロジェクトにあるその他の全 Pod に送信されます。
- あるプロジェクトの Pod によって送信されるマルチキャストパケットは、各プロジェクトが結合し、マルチキャストが結合した各プロジェクトで有効にされている場合にのみ、他のプロジェクトの Pod に送信されます。

12.11.2. Pod 間のマルチキャストの有効化

プロジェクトの Pod でマルチキャストを有効にすることができます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行し、プロジェクトのマルチキャストを有効にします。**<namespace>** を、マルチキャストを有効にする必要のある namespace に置き換えます。

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

検証

マルチキャストがプロジェクトについて有効にされていることを確認するには、以下の手順を実行します。

1. 現在のプロジェクトを、マルチキャストを有効にしたプロジェクトに切り替えます。**<project>** をプロジェクト名に置き換えます。

```
$ oc project <project>
```

2. マルチキャストレシーバーとして機能する Pod を作成します。

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
          name: mlistener
          protocol: UDP
EOF
```

3. マルチキャストセンダーとして機能する Pod を作成します。

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. マルチキャストリスナーを起動します。

   a. Pod の IP アドレスを取得します。

   ```
   $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
   ```

   b. マルチキャストリスナーを起動するには、新しいターミナルウィンドウまたはタブで以下のコマンドを入力します。

   ```
   $ oc exec mlistener -i -t -- \
       socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
   ```

5. マルチキャストトランスミッターを開始します。

   a. Pod ネットワーク IP アドレス範囲を取得します。

```
$ CIDR=$(oc get Network.config.openshift.io cluster \
    -o jsonpath='{.status.clusterNetwork[0].cidr}')
```

   b. マルチキャストメッセージを送信するには、以下のコマンドを入力します。

```
$ oc exec msender -i -t -- \
    /bin/bash -c "echo | socat STDIO UDP4-
DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
```

   マルチキャストが機能している場合、直前のコマンドは以下の出力を返します。

```
mlistener
```

12.12. プロジェクトのマルチキャストの無効化

12.12.1. Pod 間のマルチキャストの無効化

プロジェクトの Pod でマルチキャストを無効にすることができます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行して、マルチキャストを無効にします。

```
$ oc annotate netnamespace <namespace> \ (1)
    netnamespace.network.openshift.io/multicast-enabled-
```

(1) マルチキャストを無効にする必要のあるプロジェクトの **namespace**。

12.13. OPENSHIFT SDN を使用したネットワーク分離の設定

クラスターが OpenShift SDN CNI プラグインのマルチテナント分離モードを使用するように設定されている場合、各プロジェクトはデフォルトで分離されます。ネットワークトラフィックは、マルチテナント分離モードでは、異なるプロジェクトの Pod およびサービス間で許可されません。

プロジェクトのマルチテナント分離の動作を 2 つの方法で変更することができます。

- 1 つ以上のプロジェクトを結合し、複数の異なるプロジェクトの Pod とサービス間のネットワークトラフィックを可能にします。
- プロジェクトのネットワーク分離を無効にできます。これはグローバルにアクセスできるようになり、他のすべてのプロジェクトの Pod およびサービスからのネットワークトラフィックを受け入れます。グローバルにアクセス可能なプロジェクトは、他のすべてのプロジェクトの

Pod およびサービスにアクセスできます。

12.13.1. 前提条件

- クラスターは、マルチテナント分離ノードで OpenShift SDN Container Network Interface (CNI) プラグインを使用するように設定されている必要があります。

12.13.2. プロジェクトの結合

2 つ以上のプロジェクトを結合し、複数の異なるプロジェクトの Pod とサービス間のネットワークトラフィックを可能にします。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

1. 以下のコマンドを使用して、プロジェクトを既存のプロジェクトネットワークに参加させます。

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

   または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

2. オプション: 以下のコマンドを実行し、結合した Pod ネットワークを表示します。

```
$ oc get netnamespaces
```

   同じ Pod ネットワークのプロジェクトには、**NETID** 列に同じネットワーク ID があります。

12.13.3. プロジェクトの分離

他のプロジェクトの Pod およびサービスがその Pod およびサービスにアクセスできないようにするためにプロジェクトを分離することができます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- クラスターのプロジェクトを分離するには、以下のコマンドを実行します。

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

  または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

12.13.4. プロジェクトのネットワーク分離の無効化

プロジェクトのネットワーク分離を無効にできます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- プロジェクトの以下のコマンドを実行します。

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

  または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

12.14. KUBE-PROXY の設定

Kubernetes メットワークプロキシー (kube-proxy) は各ノードで実行され、Cluster Network Operator (CNO) で管理されます。kube-proxy は、サービスに関連付けられたエンドポイントの接続を転送するためのネットワークルールを維持します。

12.14.1. iptables ルールの同期について

同期の期間は、Kubernetes ネットワークプロキシー (kube-proxy) がノードで iptables ルールを同期する頻度を定めます。

同期は、以下のイベントのいずれかが生じる場合に開始します。

- サービスまたはエンドポイントのクラスターへの追加、またはクラスターからの削除などのイベントが発生する。
- 最後の同期以後の時間が kube-proxy に定義される同期期間を超過している。

12.14.2. kube-proxy 設定パラメーター

以下の **kubeProxyConfig** パラメーターを変更することができます。

重要

OpenShift Container Platform 4.3 以降で強化されたパフォーマンスの向上により、**iptablesSyncPeriod** パラメーターを調整する必要はなくなりました。

表12.2 パラメーター

パラメーター	説明	値	デフォルト

パラメーター	説明	値	デフォルト
iptablesSyncPeriod	**iptables** ルールの更新期間。	**30s** または **2m** などの期間。有効な接尾辞には、**s**、**m**、および **h** などが含まれ、これらについては、Go Package time ドキュメントで説明されています。	**30s**
proxyArguments.iptables-min-sync-period	**iptables** ルールを更新する前の最小期間。このパラメーターにより、更新の頻度が高くなり過ぎないようにできます。デフォルトでは、**iptables** ルールに影響する変更が生じるとすぐに、更新が開始されます。	**30s** または **2m** などの期間。有効な接尾辞には、**s**、**m**、および **h** が含まれ、これらについては、Go Package time で説明されています。	**0s**

12.14.3. kube-proxy 設定の変化

クラスターの Kubernetes ネットワークプロキシー設定を変更することができます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールで実行中のクラスターにログインします。

手順

1. 以下のコマンドを実行して、**Network.operator.openshift.io** カスタムリソース (CR) を編集します。

```
$ oc edit network.operator.openshift.io cluster
```

2. 以下のサンプル CR のように、kube-proxy 設定への変更内容で、CR の **kubeProxyConfig** パラメーターを変更します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period: ["30s"]
```

3. ファイルを保存し、テキストエディターを編集します。

構文は、ファイルを保存し、エディターを終了する際に **oc** コマンドによって検証されます。変更内容に構文エラーが含まれる場合、エディターはファイルを開き、エラーメッセージを表示します。

4. 以下のコマンドを実行して、設定の更新を確認します。

```
$ oc get networks.operator.openshift.io -o yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
    defaultNetwork:
      type: OpenShiftSDN
    kubeProxyConfig:
      iptablesSyncPeriod: 30s
      proxyArguments:
        iptables-min-sync-period:
        - 30s
    serviceNetwork:
    - 172.30.0.0/16
  status: {}
kind: List
```

5. オプション: 以下のコマンドを実行し、Cluster Network Operator が設定変更を受け入れていることを確認します。

```
$ oc get clusteroperator network
```

出力例

```
NAME      VERSION     AVAILABLE   PROGRESSING   DEGRADED   SINCE
network   4.1.0-0.9   True        False         False      1m
```

設定の更新が正常に適用されると、**AVAILABLE** フィールドが **True** になります。

第13章 OVN-KUBERNETES デフォルト CNI ネットワークプロバイダー

13.1. OVN-KUBERNETES デフォルト CONTAINER NETWORK INTERFACE (CNI) ネットワークプロバイダーについて

OpenShift Container Platform クラスターは、Pod およびサービスネットワークに仮想化ネットワークを使用します。OVN-Kubernetes Container Network Interface (CNI) プラグインは、デフォルトのクラスターネットワークのネットワークプロバイダーです。

13.1.1. OVN-Kubernetes の機能

OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダーは、以下の機能を実装します。

- Open Virtual Network (OVN) を使用してネットワークトラフィックフローを管理します。OVN はコミュニティーで開発され、ベンダーに依存しないネットワーク仮想化ソリューションです。
- ingress および egress ルールを含む Kubernetes ネットワークポリシーのサポートを実装します。
- ノード間にオーバーレイネットワークを作成するには、VXLAN ではなく GENEVE (Generic Network Virtualization Encapsulation) プロトコルを使用します。

13.1.2. サポートされるデフォルトの CNI ネットワークプロバイダー機能マトリクス

OpenShift Container Platform は、OpenShift SDN と OVN-Kubernetes の 2 つのサポート対象のオプションをデフォルトの Container Network Interface (CNI) ネットワークプロバイダーに提供します。以下の表は、両方のネットワークプロバイダーの現在の機能サポートをまとめたものです。

表13.1 デフォルトの CNI ネットワークプロバイダー機能の比較

機能	OVN-Kubernetes [1]	OpenShift SDN
Egress IPs	サポート対象外	サポート対象
Egress ファイアウォール [2]	サポート対象外	サポート対象
Egress ルーター	サポート対象外	サポート対象
Kubernetes ネットワークポリシー	サポート対象	一部サポート対象 [3]
マルチキャスト	サポート対象	サポート対象

1. OpenShift Container Platform 4.5 では、テクノロジープレビュー機能としてのみ利用できます。
2. egress ファイアウォールは、OpenShift SDN では egress ネットワークポリシーとしても知られています。これはネットワークポリシーの egress とは異なります。

3. egress ルールおよび一部の **ipBlock** ルールをサポートしません。

13.1.3. OVN-Kubernetes の公開メトリクス

OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダーは、Prometheus ベースの OpenShift Container Platform クラスターモニタリングスタックで使用される特定のメトリクスを公開します。

表13.2 OVN-Kubernetes によって公開されるメトリクス

名前	説明
ovnkube_master_pod_creation_latency_seconds	Pod が作成された時点から Pod が OVN-Kubernetes によってアノテーションが付けられる時点に生じるレイテンシー。レイテンシーが高いほど、Pod がネットワーク接続で利用可能になる前に経過する時間が長くなります。

追加リソース

- プロジェクトのマルチキャストの有効化
- プロジェクトのマルチキャストの無効化

13.2. OPENSHIFT SDN デフォルト CNI ネットワークプロバイダーからの移行

クラスター管理者は、OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダーを OpenShift SDN デフォルト CNI ネットワークプロバイダーから移行できます。

重要

Open Virtual Networking (OVN) Kubernetes ネットワークプラグインは、テクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

OVN テクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/articles/4380121 を参照してください。

OVN-Kubernetes についての詳細は、OVN-Kubernetes ネットワークプロバイダーについて を参照してください。

13.2.1. OVN-Kubernetes デフォルト CNI ネットワークプロバイダーへの移行

クラスター管理者は、クラスターのデフォルトの Container Network Interface (CNI) ネットワークプロバイダーを OVN-Kubernetes に変更できます。移行時に、クラスター内のすべてのノードを再起動する必要があります。

重要

移行の実行中はクラスターを利用できず、ワークロードが中断される可能性があります。サービスの中断が許容可能な場合にのみ移行を実行します。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。
- OpenShift SDN デフォルト CNI ネットワークプロバイダーで設定されたベアメタルインフラストラクチャーにクラスターがインストールされていること。
- クラスターは既知の正常な状態にあり、エラーがないこと。

手順

1. クラスターネットワークの設定のバックアップを作成するには、以下のコマンドを入力します。

```
$ oc get Network.config.openshift.io cluster -o yaml > cluster-openshift-sdn.yaml
```

2. 移行を有効にするには、以下のコマンドを入力して Cluster Network Operator 設定オブジェクトにアノテーションを設定します。

```
$ oc annotate Network.operator.openshift.io cluster \
  'networkoperator.openshift.io/network-migration'=""
```

3. デフォルトの CNI ネットワークプロバイダーを変更するには、以下のコマンドを入力します。

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{ "spec": { "networkType": "OVNKubernetes" } }'
```

4. 移行により OpenShift SDN デフォルト CNI ネットワークプロバイダーが無効になり、すべての OpenShift SDN Pos が削除されたことを確認するには、以下のコマンドを入力します。すべての OpenShift SDN Pod が停止するまでしばらく時間がかかる可能性があります。

```
$ watch oc get pod -n openshift-sdn
```

5. 移行を完了するには、クラスター内の各ノードを再起動します。たとえば、以下のような bash スクリプトを使用できます。このスクリプトは、**ssh** を使用して各ホストに接続でき、**sudo** がパスワードを要求しないように設定されていることを前提としています。

```
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
(@.type=="InternalIP")].address}')
do
   echo "reboot node $ip"
   ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

6. クラスターのノードが再起動したら、移行が正常に完了したことを確認します。

a. デフォルトの CNI ネットワークプロバイダーが OVN-Kubernetes であることを確認するには、以下のコマンドを入力します。**status.networkType** の値は **OVNKubernetes** である必要があります。

```
$ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
```

b. クラスターノードが **Ready** 状態にあることを確認するには、以下のコマンドを実行します。

```
$ oc get nodes
```

ノードが **NotReady** 状態のままになっている場合は、再度ノードを再起動します。

c. Pod がエラー状態ではないことを確認するには、以下のコマンドを入力します。

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

ノードの Pod がエラー状態にある場合は、そのノードを再起動します。

7. 以下の手順は、移行に成功し、クラスターの状態が正常である場合にのみ実行します。

a. Cluster Network Operator 設定オブジェクトから移行アノテーションを削除するには、以下のコマンドを入力します。

```
$ oc annotate Network.operator.openshift.io cluster \
  networkoperator.openshift.io/network-migration-
```

b. OpenShift SDN ネットワークプロバイダー namespace を削除するには、以下のコマンドを入力します。

```
$ oc delete namespace openshift-sdn
```

13.3. OPENSHIFT SDN ネットワークプロバイダーへのロールバック

クラスター管理者は、OVN-Kubernetes デフォルト CNI ネットワークプロバイダーから OpenShift SDN クラスターのデフォルト Container Network Interface (CNI) プロバイダーにロールバックできます (OVN-Kubernetes への移行に失敗した場合)。

13.3.1. デフォルトの CNI ネットワークプロバイダーの OpenShift SDN へのロールバック

クラスター管理者は、クラスターを OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーにロールバックできます。ロールバック時に、クラスター内のすべてのノードを再起動する必要があります。

重要

OVN-Kubernetes への移行に失敗した場合にのみ OpenShift SDN にロールバックします。

前提条件

- OpenShift CLI (**oc**) をインストールしている。

- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセス。
- OVN-Kubernetes デフォルト CNI ネットワークプロバイダーで設定されたベアメタルインフラストラクチャーにクラスターがインストールされていること。

1. 移行を有効にするには、以下のコマンドを入力して Cluster Network Operator 設定オブジェクトにアノテーションを設定します。

```
$ oc annotate Network.operator.openshift.io cluster \
  'networkoperator.openshift.io/network-migration'=""
```

2. デフォルトの CNI ネットワークプロバイダーを変更するには、以下のコマンドを入力します。

```
$ oc patch Network.config.openshift.io cluster \
  --type='merge' --patch '{ "spec": { "networkType": "OpenShiftSDN" } }'
```

3. オプション: 移行前に作成したクラスターネットワーク設定のバックアップを使用して、ネットワーク設定に対して作成した可能性のあるカスタマイズを復元します。カスタマイズを復元するには、以下のコマンドを入力して Cluster Network Operator 設定を編集します。

```
$ oc edit Network.config.openshift.io cluster
```

4. 移行により OVN-Kubernetes デフォルト CNI ネットワークプロバイダーが無効になり、すべての OVN-Kubernetes Pod が削除されたことを確認するには、以下のコマンドを入力します。すべての OVN-Kubernetes Pod が停止するまでしばらく時間がかかる可能性があります。

```
$ watch oc get pod -n openshift-ovn-kubernetes
```

5. ロールバックを完了するには、クラスター内の各ノードを再起動します。たとえば、以下のような bash スクリプトを使用できます。このスクリプトは、**ssh** を使用して各ホストに接続でき、**sudo** がパスワードを要求しないように設定されていることを前提としています。

```
#!/bin/bash

for ip in $(oc get nodes  -o jsonpath='{.items[*].status.addresses[?
(@.type=="InternalIP")].address}')
do
  echo "reboot node $ip"
  ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

6. クラスターのノードが再起動したら、以下のコマンドを実行してデフォルトの CNI ネットワークプロバイダーが OpenShift SDN であることを確認します。**status.networkType** の値は **OpenShiftSDN** である必要があります。

```
$ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
```

7. OpenShift SDN Pod が **READY** の状態にあることを確認するには、以下のコマンドを入力します。

```
$ oc get pod -n openshift-sdn --watch
```

8. Cluster Network Operator 設定オブジェクトから移行アノテーションを削除するには、以下のコマンドを入力します。

```
$ oc annotate Network.operator.openshift.io cluster \
  networkoperator.openshift.io/network-migration-
```

9. OVN-Kubernetes ネットワークプロバイダー namespace を削除するには、以下のコマンドを入力します。

```
$ oc delete namespace openshift-ovn-kubernetes
```

13.4. プロジェクトのマルチキャストの有効化

重要

Open Virtual Networking (OVN) Kubernetes ネットワークプラグインは、テクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

OVN テクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/articles/4380121 を参照してください。

注記

OpenShift Container Platform 4.5 では、バグにより、同じ namespace にあるが、異なるノードに割り当てられている Pod がマルチキャストで通信できなくなります。詳細は、BZ#1843695 を参照してください。

13.4.1. マルチキャストについて

IP マルチキャストを使用すると、データが多数の IP アドレスに同時に配信されます。

重要

現時点で、マルチキャストは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のソリューションとしては適していません。

OpenShift Container Platform の Pod 間のマルチキャストトラフィックはデフォルトで無効にされます。OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダーを使用している場合には、プロジェクトごとにマルチキャストを有効にすることができます。

13.4.2. Pod 間のマルチキャストの有効化

プロジェクトの Pod でマルチキャストを有効にすることができます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行し、プロジェクトのマルチキャストを有効にします。**<namespace>** を、マルチキャストを有効にする必要のある namespace に置き換えます。

```
$ oc annotate namespace <namespace> \
    k8s.ovn.org/multicast-enabled=true
```

検証

マルチキャストがプロジェクトについて有効にされていることを確認するには、以下の手順を実行します。

1. 現在のプロジェクトを、マルチキャストを有効にしたプロジェクトに切り替えます。**<project>** をプロジェクト名に置き換えます。

```
$ oc project <project>
```

2. マルチキャストレシーバーとして機能する Pod を作成します。

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
          name: mlistener
          protocol: UDP
EOF
```

3. マルチキャストセンダーとして機能する Pod を作成します。

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
```

```
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi8
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. マルチキャストリスナーを起動します。

   a. Pod の IP アドレスを取得します。

   ```
   $ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
   ```

   b. マルチキャストリスナーを起動するには、新しいターミナルウィンドウまたはタブで以下のコマンドを入力します。

   ```
   $ oc exec mlistener -i -t -- \
       socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
   ```

5. マルチキャストトランスミッターを開始します。

   a. Pod ネットワーク IP アドレス範囲を取得します。

   ```
   $ CIDR=$(oc get Network.config.openshift.io cluster \
       -o jsonpath='{.status.clusterNetwork[0].cidr}')
   ```

   b. マルチキャストメッセージを送信するには、以下のコマンドを入力します。

   ```
   $ oc exec msender -i -t -- \
       /bin/bash -c "echo | socat STDIO UDP4-
   DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
   ```

   マルチキャストが機能している場合、直前のコマンドは以下の出力を返します。

   ```
   mlistener
   ```

13.5. プロジェクトのマルチキャストの無効化

重要

Open Virtual Networking (OVN) Kubernetes ネットワークプラグインは、テクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。

OVN テクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/articles/4380121 を参照してください。

13.5.1. Pod 間のマルチキャストの無効化

プロジェクトの Pod でマルチキャストを無効にすることができます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行して、マルチキャストを無効にします。

```
$ oc annotate namespace <namespace> \ (1)
    k8s.ovn.org/multicast-enabled-
```

(1) マルチキャストを無効にする必要のあるプロジェクトの **namespace**。

第14章 ルートの作成

14.1. ルート設定

14.1.1. ルートのタイムアウトの設定

Service Level Availability (SLA) で必要とされる、低タイムアウトが必要なサービスや、バックエンドでの処理速度が遅いケースで高タイムアウトが必要なサービスがある場合は、既存のルートに対してデフォルトのタイムアウトを設定することができます。

前提条件

- 実行中のクラスターでデプロイ済みの Ingress コントローラーが必要になります。

手順

1. **oc annotate** コマンドを使用して、ルートにタイムアウトを追加します。

```
$ oc annotate route <route_name> \
    --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> (1)
```

(1) サポートされる時間単位は、マイクロ秒 (us)、ミリ秒 (ms)、秒 (s)、分 (m)、時間 (h)、または日 (d) です。

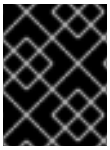
以下の例では、2 秒のタイムアウトを **myroute** という名前のルートに設定します。

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

14.1.2. HTTP Strict Transport Security の有効化

HTTP Strict Transport Security (HSTS) ポリシーは、ホストで HTTPS トラフィックのみを許可するセキュリティーの拡張機能です。デフォルトで、すべての HTTP 要求はドロップされます。これは、web サイトとの対話の安全性を確保したり、ユーザーのためにセキュアなアプリケーションを提供するのに役立ちます。

HSTS が有効にされると、HSTS はサイトから Strict Transport Security ヘッダーを HTTPS 応答に追加します。リダイレクトするルートで **insecureEdgeTerminationPolicy** 値を使用し、HTTP を HTTPS に送信するようにします。ただし、HSTS が有効にされている場合は、要求の送信前にクライアントがすべての要求を HTTP URL から HTTPS に変更するためにリダイレクトの必要がなくなります。これはクライアントでサポートされる必要はなく、**max-age=0** を設定することで無効にできます。



重要

HSTS はセキュアなルート (edge termination または re-encrypt) でのみ機能します。この設定は、HTTP またはパススルールートには適していません。

手順

- ルートに対して HSTS を有効にするには、**haproxy.router.openshift.io/hsts_header** 値を edge termination または re-encrypt ルートに追加します。

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
(1) (2) (3)
```

1. **max-age** は唯一の必須パラメーターです。これは、HSTS ポリシーが有効な期間 (秒単位) を測定します。クライアントは、ホストから HSTS ヘッダーのある応答を受信する際には常に **max-age** を更新します。**max-age** がタイムアウトになると、クライアントはポリシーを破棄します。

2. **includeSubDomains** はオプションです。これが含まれる場合、クライアントに対し、ホストのすべてのサブドメインがホストと同様に処理されるように指示します。

3. **preload** はオプションです。**max-age** が 0 より大きい場合、**preload** を **haproxy.router.openshift.io/hsts_header** に組み込むことにより、外部サービスはこのサイトをそれぞれの HSTS プリロード一覧に含めることができます。たとえば、Google などのサイトは **preload** が設定されているサイトの一覧を作成します。ブラウザーはこれらの一覧を使用し、サイトと対話する前でも HTTPS 経由で通信できるサイトを判別できます。**preload** 設定がない場合、ブラウザーはヘッダーを取得するために HTTPS 経由でサイトと通信している必要があります。

14.1.3. スループット関連の問題のトラブルシューティング

OpenShift Container Platform でデプロイされるアプリケーションでは、特定のサービス間で非常に長い待ち時間が発生するなど、ネットワークのスループットの問題が生じることがあります。

Pod のログが問題の原因を指摘しない場合は、以下の方法を使用してパフォーマンスの問題を分析します。

- ping または tcpdump などのパケットアナライザーを使用して Pod とそのノード間のトラフィックを分析します。
  たとえば、問題を生じさせる動作を再現している間に各 Pod で tcpdump ツールを実行します。両サイトでキャプチャーしたデータを確認し、送信および受信タイムスタンプを比較して Pod への/からのトラフィックの待ち時間を分析します。待ち時間は、ノードのインターフェイスが他の Pod やストレージデバイス、またはデータプレーンからのトラフィックでオーバーロードする場合に OpenShift Container Platform で発生する可能性があります。

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> (1)
  ```

  1. **podip** は Pod の IP アドレスです。 **oc get pod <pod_name> -o wide** コマンドを実行して Pod の IP アドレスを取得します。

  tcpdump は 2 つの Pod 間のすべてのトラフィックが含まれる **/tmp/dump.pcap** のファイルを生成します。理想的には、ファイルサイズを最小限に抑えるために問題を再現するすぐ前と問題を再現したすぐ後ににアナライザーを実行することが良いでしょう。以下のようにノード間でパケットアナライザーを実行することもできます (式から SDN を排除する)。

  ```
  $ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
  ```

- ストリーミングのスループットおよび UDP スループットを測定するために iperf などの帯域幅測定ツールを使用します。ボトルネックの特定を試行するには、最初に Pod から、次にノードからツールを実行します。
  - iperf のインストールおよび使用についての詳細は、こちらの Red Hat ソリューション を参照してください。

14.1.4. Cookie に使用によるルートのステートフル性の維持

OpenShift Container Platform は、すべてのトラフィックを同じエンドポイントにヒットさせることによりステートフルなアプリケーションのトラフィックを可能にするスティッキーセッションを提供します。ただし、エンドポイント Pod が再起動、スケーリング、または設定の変更などによって終了する場合、このステートフル性はなくなります。

OpenShift Container Platform は Cookie を使用してセッションの永続化を設定できます。Ingress コントローラーはユーザー要求を処理するエンドポイントを選択し、そのセッションの Cookie を作成します。Cookie は要求の応答として戻され、ユーザーは Cookie をセッションの次の要求と共に送り返します。Cookie は Ingress コントローラーに対し、セッションを処理しているエンドポイントを示し、クライアント要求が Cookie を使用して同じ Pod にルーティングされるようにします。

14.1.4.1. Cookie を使用したルートのアノテーション

ルート用に自動生成されるデフォルト名を上書きするために Cookie 名を設定できます。これにより、ルートトラフィックを受信するアプリケーションが Cookie 名を認識できるようになります。Cookie を削除すると、次の要求でエンドポイントの再選択が強制的に実行される可能性があります。そのためサーバーがオーバーロードしている場合には、クライアントからの要求を取り除き、それらの再分配を試行します。

手順

1. 指定される cookie 名でルートにアノテーションを付けます。

```
$ oc annotate route <route_name> router.openshift.io/cookie_name="<cookie_name>"
```

ここでは、以下のようになります。

<route_name>
Pod の名前を指定します。

<cookie_name>
cookie の名前を指定します。

たとえば、ルート **my_route** に cookie 名 **my_cookie** でアノテーションを付けるには、以下を実行します。

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2. 変数でルートのホスト名を取得します。

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

ここでは、以下のようになります。

<route_name>

Pod の名前を指定します。

3. cookie を保存してからルートにアクセスします。

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

ルートに接続する際に、直前のコマンドによって保存される cookie を使用します。

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

14.1.5. パスベースのルート

パスベースのルートは、URL に対して比較できるパスコンポーネントを指定します。この場合、ルートのトラフィックは HTTP ベースである必要があります。そのため、それぞれが異なるパスを持つ同じホスト名を使用して複数のルートを提供できます。ルーターは、最も具体的なパスの順に基づいてルートと一致する必要があります。ただし、これはルーターの実装によって異なります。

以下の表は、ルートのサンプルおよびそれらのアクセシビリティーを示しています。

表14.1 ルートの可用性

ルート	比較対象	アクセス可能
www.example.com/test	www.example.com/test	はい
	www.example.com	いいえ
www.example.com/test および www.example.com	www.example.com/test	はい
	www.example.com	はい
www.example.com	www.example.com/text	Yes (ルートではなく、ホストで一致)
	www.example.com	はい

パスが 1 つでセキュリティー保護されていないルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" (1)
  to:
    kind: Service
    name: service-name
```

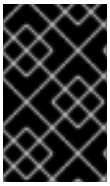
(1) パスは、パスベースのルートに唯一追加される属性です。

注記

ルーターは TLS を終了させず、要求のコンテンツを読み込みことができないので、パスベースのルーティングは、パススルー TLS を使用する場合には利用できません。

14.1.6. ルート固有のアノテーション

Ingress コントローラーは、公開するすべてのルートのデフォルトオプションを設定できます。個別のルートは、アノテーションに個別の設定を指定して、デフォルトの一部を上書きできます。Red Hat では、ルートアノテーションの Operator 管理ルートへの追加はサポートしません。



重要

複数のソース IP またはサブネットのホワイトリストを作成するには、スペースで区切られたリストを使用します。他の区切りタイプを使用すると、一覧が警告やエラーメッセージなしに無視されます。

表14.2 ルートアノテーション

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/balance	ロードバランシングアルゴリズムを設定します。使用できるオプションは **source**、**roundrobin**、および **leastconn** です。	パススルールートの **ROUTER_TCP_BALANCE_SCHEME** です。それ以外の場合は **ROUTER_LOAD_BALANCE_ALGORITHM** を使用します。
haproxy.router.openshift.io/disable_cookies	関連の接続を追跡する cookie の使用を無効にします。**true** または **TRUE** に設定する場合は、分散アルゴリズムを使用して、受信する HTTP 要求ごとに、どのバックエンドが接続を提供するかを選択します。	
router.openshift.io/cookie_name	このルートに使用するオプションの cookie を指定します。名前は、大文字、小文字、数字、"_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。	

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/pod-concurrent-connections	ルーターからバッキングされる Pod に対して許容される接続最大数を設定します。注意: Pod が複数ある場合には、それぞれに対応する接続数を設定できますが、ルーターが複数ある場合には、ルーター間の連携がなく、それぞれの接続回数はルーターの数と同じとなります。ただし、複数のルーターがある場合は、それらのルーター間で調整は行われず、それぞれがこれに複数回接続する可能性があります。設定されていない場合または 0 に設定されている場合には制限はありません。	
haproxy.router.openshift.io/rate-limit-connections	レート制限機能を有効にするために **true** または **TRUE** を設定します。	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	IP アドレスで共有される同時 TCP 接続の数を制限します。	
haproxy.router.openshift.io/rate-limit-connections.rate-http	IP アドレスが HTTP 要求を実行できるレートを制限します。	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	IP アドレスが TCP 接続を行うレートを制限します。	
haproxy.router.openshift.io/timeout	ルートのサーバー側のタイムアウトを設定します。(TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**
router.openshift.io/haproxy.health.check.interval	バックエンドのヘルスチェックの間隔を設定します。(TimeUnits)	**ROUTER_BACKEND_CHECK_INTERVAL**
haproxy.router.openshift.io/ip_whitelist	ルートのホワイトリストを設定します。ホワイトリストは、承認したソースアドレスの IP アドレスおよび CIDR 範囲の一覧をスペース区切りにします。ホワイトリストに含まれていない IP アドレスからの要求は破棄されます。 ホワイトリストの許可される IP アドレスおよび CIDR 範囲の最大数は 61 です。	

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/hsts_header	edge terminated または re-encrypt ルートの Strick-Transport-Security ヘッダーを設定します。	
haproxy.router.openshift.io/log-send-hostname	Syslog ヘッダーの **hostname** フィールドを設定します。システムのホスト名を使用します。サイドカーや Syslog ファシリティーなどの Ingress API ロギングメソッドがルーターに対して有効になっている場合、**log-send-hostname** はデフォルトで有効になります。	

注記

環境変数を編集することはできません。

ルート設定のカスタムタイムアウト

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms (1)
...
```

(1) HAProxy 対応の単位 (**us**、**ms**、**s**、 **m**、**h**、**d**) で新規のタイムアウトを指定します。単位が指定されていない場合は、**ms** がデフォルトになります。

注記

パススルールートのサーバー側のタイムアウト値を低く設定し過ぎると、WebSocket 接続がそのルートで頻繁にタイムアウトする可能性があります。

特定の IP アドレスを 1 つだけ許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

複数の IP アドレスを許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

IP アドレスの CIDR ネットワークを許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

IP アドレスと IP アドレスの CIDR ネットワークの両方を許可するルート

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

14.1.7. ルートの受付ポリシーの設定

管理者およびアプリケーション開発者は、同じドメイン名を持つ複数の namespace でアプリケーションを実行できます。これは、複数のチームが同じホスト名で公開されるマイクロサービスを開発する組織を対象としています。

警告

複数の namespace での要求の許可は、namespace 間の信頼のあるクラスターに対してのみ有効にする必要があります。有効にしないと、悪意のあるユーザーがホスト名を乗っ取る可能性があります。このため、デフォルトの受付ポリシーは複数の namespace 間でのホスト名の要求を許可しません。

前提条件

- クラスター管理者の権限。

手順

- 以下のコマンドを使用して、**ingresscontroller** リソース変数の **.spec.routeAdmission** フィールドを編集します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec":
{"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --type=merge
```

イメージコントローラー設定例

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
```

14.2. セキュリティー保護されたルート

セキュアなルートは、複数の TLS 終端タイプを使用してクライアントに証明書を提供できます。以下のセクションでは、カスタム証明書を使用して re-encrypt、edge、および passthrough ルートを作成する方法を説明します。

重要

パブリックエンドポイントを使用して Microsoft Azure にルートを作成する場合、リソース名は制限されます。特定の用語を使用するリソースを作成することはできません。Azure が制限する語の一覧は、Azure ドキュメントの Resolve reserved resource name errors を参照してください。

14.2.1. カスタム証明書を使用した re-encrypt ルートの作成

oc create route コマンドを使用し、カスタム証明書と共に reencrypt TLS termination を使用してセキュアなルートを設定できます。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。 ここで、証明書はルートホストに対して有効である必要があります。
- 証明書チェーンを完了する PEM エンコードされたファイルの別の CA 証明書が必要です。
- PEM エンコードされたファイルの別の宛先 CA 証明書が必要です。
- 公開する必要のあるサービスが必要です。

注記

パスワードで保護されるキーファイルはサポートされません。キーファイルからパスフレーズを削除するには、以下のコマンドを使用します。

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

手順

この手順では、カスタム証明書および reencrypt TLS termination を使用して **Route** リソースを作成します。以下では、証明書/キーのペアが現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提としています。また、Ingress コントローラーがサービスの証明書を信頼できるように宛先 CA 証明書を指定する必要もあります。必要な場合には、証明書チェーンを完了するために CA 証明書を指定することもできます。**tls.crt**、 **tls.key**、**cacert.crt**、および (オプションで) **ca.crt** を実際のパス名に置き換えます。**frontend** を、公開する必要のある **Service** リソースに置き換えます。**www.example.com** を適切な名前に置き換えます。

- reencrypt TLS 終端およびカスタム証明書を使用してセキュアな **Route** リソースを作成します。

  ```
  $ oc create route reencrypt --service=frontend --cert=tls.crt --key=tls.key --dest-ca-cert=destca.crt --ca-cert=ca.crt --hostname=www.example.com
  ```

  結果として生成される **Route** リソースを検査すると、以下のようになります。

  セキュアなルートの YAML 定義

```
apiVersion: v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    destinationCACertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

他のオプションについては、**oc create route reencrypt --help** を参照してください。

14.2.2. カスタム証明書を使用した edge ルートの作成

oc create route コマンドを使用し、edge TLS termination とカスタム証明書を使用してセキュアなルートを設定できます。edge ルートの場合、Ingress コントローラーは、トラフィックを宛先 Pod に転送する前に TLS 暗号を終了します。ルートは、Ingress コントローラーがルートに使用する TLS 証明書およびキーを指定します。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。 ここで、証明書はルートホストに対して有効である必要があります。
- 証明書チェーンを完了する PEM エンコードされたファイルの別の CA 証明書が必要です。
- 公開する必要のあるサービスが必要です。

注記

パスワードで保護されるキーファイルはサポートされません。キーファイルからパスフレーズを削除するには、以下のコマンドを使用します。

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

手順

この手順では、カスタム証明書および edge TLS termination を使用して **Route** リソースを作成します。以下では、証明書/キーのペアが現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提としています。必要な場合には、証明書チェーンを完了するために CA 証明書を指定することもできます。**tls.crt**、 **tls.key**、および (オプションで) **ca.crt** を実際のパス名に置き換えます。**frontend** を、公開する必要のあるサービスの名前に置き換えます。**www.example.com** を適切な名前に置き換えます。

- edge TLS termination およびカスタム証明書を使用して、セキュアな **Route** リソースを作成します。

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=www.example.com
```

結果として生成される **Route** リソースを検査すると、以下のようになります。

セキュアなルートの YAML 定義

```
apiVersion: v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: edge
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

他のオプションについては、**oc create route edge --help** を参照してください。

14.2.3. passthrough ルートの作成

oc create route コマンドを使用し、passthrough termination を使用してセキュアなルートを設定できます。passthrough termination では、暗号化されたトラフィックが TLS 終端を提供するルーターなしに宛先に直接送信されます。そのため、ルートでキーや証明書は必要ありません。

前提条件

- 公開する必要のあるサービスが必要です。

手順

- **Route** リソースを作成します。

```
$ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
```

結果として生成される **Route** リソースを検査すると、以下のようになります。

passthrough termination を使用したセキュリティー保護されたルート

```
apiVersion: v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough (2)
    insecureEdgeTerminationPolicy: None (3)
  to:
    kind: Service
    name: frontend
```

1. オブジェクトの名前で、63 文字に制限されます。
2. **termination** フィールドを **passthrough** に設定します。これは、必要な唯一の **tls** フィールドです。
3. オプションの **insecureEdgeTerminationPolicy**。唯一有効な値は **None**、**Redirect**、または空の値です (無効にする場合)。

宛先 Pod は、エンドポイントでトラフィックに証明書を提供します。これは、必須となるクライアント証明書をサポートするための唯一の方法です (相互認証とも呼ばれる)。

第15章 INGRESS クラスタートラフィックの設定

15.1. INGRESS クラスタートラフィックの設定の概要

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする以下の方法を提供します。

以下の方法が推奨されます。以下は、これらの方法の優先される順です。

- HTTP/HTTPS を使用する場合は Ingress コントローラーを使用する。
- HTTPS 以外の TLS で暗号化されたプロトコルを使用する場合、たとえば、SNI ヘッダーを使用する TLS の場合は、Ingress コントローラーを使用します。
- それ以外の場合は、ロードバランサー、外部 IP、または **NodePort** を使用します。

方法	目的
Ingress コントローラーの使用	HTTP/HTTPS トラフィックおよび HTTPS 以外の TLS で暗号化されたプロトコル (TLS と SNI ヘッダーの使用など) へのアクセスを許可します。
ロードバランサーサービスを使用した外部 IP の自動割り当て	プールから割り当てられた IP アドレスを使った非標準ポートへのトラフィックを許可します。
外部 IP のサービスへの手動割り当て	特定の IP アドレスを使った非標準ポートへのトラフィックを許可します。
NodePort の設定	クラスターのすべてのノードでサービスを公開します。

15.2. サービスの EXTERNALIP の設定

クラスター管理者は、トラフィックをクラスター内のサービスに送信できるクラスター外の IP アドレスブロックを指定できます。

この機能は通常、ベアメタルハードウェアにインストールされているクラスターに最も役立ちます。

15.2.1. 前提条件

- ネットワークインフラストラクチャーは、外部 IP アドレスのトラフィックをクラスターにルーティングする必要があります。

15.2.2. ExternalIP について

クラウド以外の環境では、OpenShift Container Platform は **ExternalIP** 機能を使用して外部 IP アドレスの **Service** オブジェクトの **spec.externalIPs[]** フィールドへの割り当てをサポートします。このフィールドを設定すると、OpenShift Container Platform は追加の仮想 IP アドレスをサービスに割り当てます。IP アドレスは、クラスターに定義されたサービスネットワーク外に指定できます。**type=NodePort** が設定されたサービスと同様に ExternalIP 機能で設定されたサービスにより、トラフィックを負荷分散のためにローカルノードに転送することができます。

ネットワークインフラストラクチャーを設定し、定義する外部 IP アドレスブロックがクラスターにルーティングされるようにする必要があります。

OpenShift Container Platform は以下の機能を追加して Kubernetes の ExternalIP 機能を拡張します。

- 設定可能なポリシーでの、ユーザーによる外部 IP アドレスの使用の制限
- 要求時の外部 IP アドレスのサービスへの自動割り当て

警告

ExternalIP 機能の使用はデフォルトで無効にされます。これは、外部 IP アドレスへのクラスター内のトラフィックがそのサービスにダイレクトされるため、セキュリティー上のリスクを生じさせる可能性があります。これにより、クラスターユーザーは外部リソースについての機密性の高いトラフィックをインターセプトできるようになります。

重要

この機能は、クラウド以外のデプロイメントでのみサポートされます。クラウドデプロイメントの場合、クラウドの自動デプロイメントのためにロードバランサーサービスを使用し、サービスのエンドポイントをターゲットに設定します。

以下の方法で外部 IP アドレスを割り当てることができます。

外部 IP の自動割り当て

OpenShift Container Platform は、**spec.type=LoadBalancer** を設定して **Service** オブジェクトを作成する際に、IP アドレスを **autoAssignCIDRs** CIDR ブロックから **spec.externalIPs[]** 配列に自動的に割り当てます。この場合、OpenShift Container Platform はロードバランサーサービスタイプのクラウド以外のバージョンを実装し、IP アドレスをサービスに割り当てます。自動割り当てはデフォルトで無効にされており、以下のセクションで説明されているように、これはクラスター管理者が設定する必要があります。

外部 IP の手動割り当て

OpenShift Container Platform は **Service** オブジェクトの作成時に **spec.externalIPs[]** 配列に割り当てられた IP アドレスを使用します。別のサービスによってすでに使用されている IP アドレスを指定することはできません。

15.2.2.1. ExternalIP の設定

OpenShift Container Platform での外部 IP アドレスの使用は、**cluster** という名前の **Network.config.openshift.io** CR の以下のフィールドで管理されます。

- **spec.externalIP.autoAssignCIDRs** は、サービスの外部 IP アドレスを選択する際にロードバランサーによって使用される IP アドレスブロックを定義します。OpenShift Container Platform は、自動割り当て用の単一 IP アドレスブロックのみをサポートします。これは、ExternalIP をサービスに手動で割り当てる際に、制限された数の共有 IP アドレスのポート領域を管理しなくてはならない場合よりも単純になります。自動割り当てが有効な場合には、**spec.type=LoadBalancer** が設定された **Service** オブジェクトには外部 IP アドレスが割り当てられます。

- **spec.externalIP.policy** は、IP アドレスを手動で指定する際に許容される IP アドレスブロックを定義します。OpenShift Container Platform は、**spec.externalIP.autoAssignCIDRs** で定義される IP アドレスブロックにポリシールールを適用しません。

ルーティングが正しく行われると、設定された外部 IP アドレスブロックからの外部トラフィックは、サービスが公開する TCP ポートまたは UDP ポートを介してサービスのエンドポイントに到達できます。

重要

割り当てる IP アドレスブロックがクラスター内の 1 つ以上のノードで終了することを確認する必要があります。

OpenShift Container Platform は IP アドレスの自動および手動割り当ての両方をサポートしており、それぞれのアドレスは 1 つのサービスの最大数に割り当てられることが保証されます。これにより、各サービスは、ポートが他のサービスで公開されているかによらず、自らの選択したポートを公開できます。

注記

OpenShift Container Platform の **autoAssignCIDRs** で定義された IP アドレスブロックを使用するには、ホストのネットワークに必要な IP アドレスの割り当ておよびルーティングを設定する必要があります。

以下の YAML は、外部 IP アドレスが設定されたサービスについて説明しています。

spec.externalIPs[] が設定された Service オブジェクトの例

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: highport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
```

15.2.2.2. 外部 IP アドレスの割り当ての制限

クラスター管理者は、IP アドレスブロックを指定して許可および拒否できます。

制限は、**cluster-admin** 権限を持たないユーザーにのみ適用されます。クラスター管理者は、サービスの **spec.externalIPs[]** フィールドを任意の IP アドレスに常に設定できます。

spec.ExternalIP.policy フィールドを指定して、**policy** オブジェクトが定義された IP アドレスポリシーを設定します。ポリシーオブジェクトには以下の形があります。

```
{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}
```

ポリシーの制限を設定する際に、以下のルールが適用されます。

- **policy={}** が設定される場合、**spec.ExternalIPs[]** が設定されている **Service** オブジェクトの作成は失敗します。これは OpenShift Container Platform のデフォルトです。
- **policy=null** が設定される場合、**spec.ExternalIPs[]** が IP アドレスに設定される **Service** オブジェクトの作成は許可されます。
- **policy** が設定され、**policy.allowedCIDRs[]** または **policy.rejectedCIDRs[]** のいずれかが設定される場合、以下のルールが適用されます。
  - **allowedCIDRs[]** と **rejectedCIDRs[]** の両方が設定される場合、**rejectedCIDRs[]** が **allowedCIDRs[]** よりも優先されます。
  - **allowedCIDRs[]** が設定される場合、**spec.ExternalIPs[]** が設定されている **Service** オブジェクトの作成は、指定された IP アドレスが許可される場合にのみ正常に実行されます。
  - **rejectedCIDRs[]** が設定される場合、**spec.ExternalIPs[]** が設定されている **Service** オブジェクトの作成は、指定された IP アドレスが拒否されていない場合にのみ正常に実行されます。

15.2.2.3. ポリシーオブジェクトの例

以下に続く例では、複数のポリシー設定の例を示します。

- 以下の例では、ポリシーは OpenShift Container Platform が外部 IP アドレスが指定されたサービスを作成するのを防ぎます。

  Service オブジェクトの spec.externalIPs[] に指定された値を拒否するポリシーの例

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    externalIP:
      policy: {}
    ...
  ```

- 以下の例では、**allowedCIDRs** および **rejectedCIDRs** フィールドの両方が設定されます。

許可される、および拒否される CIDR ブロックの両方を含むポリシーの例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy:
      allowedCIDRs:
      - 172.16.66.10/23
      rejectedCIDRs:
      - 172.16.66.10/24
  ...
```

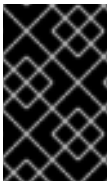
- 以下の例では、**policy** は **null** に設定されます。**null** に設定されている場合、**oc get networks.config.openshift.io -o yaml** を入力して設定オブジェクトを検査する際に、**policy** フィールドは出力に表示されません。

Service オブジェクトの spec.externalIPs[] に指定された値を許可するポリシーの例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: null
  ...
```

15.2.3. ExternalIP アドレスブロックの設定

ExternalIP アドレスブロックの設定は、**cluster** という名前の Network カスタムリソース (CR) で定義されます。ネットワーク CR は **config.openshift.io** API グループに含まれます。



重要

クラスターのインストール時に、Cluster Version Operator (CVO) は **cluster** という名前のネットワーク CR を自動的に作成します。このタイプのその他の CR オブジェクトの作成はサポートされていません。

以下の YAML は ExternalIP 設定について説明しています。

cluster という名前の network.config.openshift.io CR

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
```

```
autoAssignCIDRs: [] (1)
policy: (2)
  ...
```

1. 外部 IP アドレスのサービスへの自動割り当てに使用できる CIDR 形式で IP アドレスブロックを定義します。1 つの IP アドレス範囲のみが許可されます。
2. IP アドレスのサービスへの手動割り当ての制限を定義します。制限が定義されていない場合は、**Service** オブジェクトに **spec.externalIP** フィールドを指定しても許可されません。デフォルトで、制限は定義されません。

以下の YAML は、**policy** スタンザのフィールドについて説明しています。

Network.config.openshift.io policy スタンザ

```
policy:
  allowedCIDRs: [] (1)
  rejectedCIDRs: [] (2)
```

1. CIDR 形式の許可される IP アドレス範囲の一覧。
2. CIDR 形式の拒否される IP アドレス範囲の一覧。

外部 IP 設定の例

外部 IP アドレスプールの予想される複数の設定が以下の例で表示されています。

- 以下の YAML は、自動的に割り当てられた外部 IP アドレスを有効にする設定について説明しています。

  spec.externalIP.autoAssignCIDRs が設定された設定例

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    ...
    externalIP:
      autoAssignCIDRs:
      - 192.168.132.254/29
  ```

- 以下の YAML は、許可された、および拒否された CIDR 範囲のポリシールールを設定します。

  spec.externalIP.policy が設定された設定例

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    ...
    externalIP:
  ```

```
policy:
  allowedCIDRs:
  - 192.168.132.0/29
  - 192.168.132.8/29
  rejectedCIDRs:
  - 192.168.132.7/32
```

15.2.4. クラスターの外部 IP アドレスブロックの設定

クラスター管理者は、以下の ExternalIP を設定できます。

- **Service** オブジェクトの **spec.clusterIP** フィールドを自動的に設定するために OpenShift Container Platform によって使用される ExternalIP アドレスブロック。
- IP アドレスを制限するポリシーオブジェクトは **Service** オブジェクトの **spec.clusterIP** 配列に手動で割り当てられます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. オプション: 現在の外部 IP 設定を表示するには、以下のコマンドを入力します。

```
$ oc describe networks.config cluster
```

2. 設定を編集するには、以下のコマンドを入力します。

```
$ oc edit networks.config cluster
```

3. 以下の例のように ExternalIP 設定を変更します。

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: (1)
  ...
```

(1) **externalIP** スタンザの設定を指定します。

4. 更新された ExternalIP 設定を確認するには、以下のコマンドを入力します。

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{{"\n"}}'
```

15.2.5. 次のステップ

- サービスの外部 IP を使用した ingress クラスタートラフィックの設定

15.3. INGRESS コントローラーを使用した INGRESS クラスターの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法は Ingress コントローラーを使用します。

15.3.1. Ingress コントローラーおよびルートの使用

Ingress Operator は Ingress コントローラーおよびワイルドカード DNS を管理します。

Ingress コントローラーの使用は、OpenShift Container Platform クラスターへの外部アクセスを許可するための最も一般的な方法です。

Ingress コントローラーは外部要求を許可し、設定されたルートに基づいてそれらをプロキシー送信するよう設定されます。これは、HTTP、SNI を使用する HTTPS、SNI を使用する TLS に限定されており、SNI を使用する TLS で機能する Web アプリケーションやサービスには十分な設定です。

管理者と連携して Ingress コントローラーを設定します。外部要求を許可し、設定されたルートに基づいてそれらをプロキシー送信するように Ingress コントローラーを設定します。

管理者はワイルドカード DNS エントリーを作成してから Ingress コントローラーを設定できます。その後は管理者に問い合わせることなく edge Ingress コントローラーと連携できます。

デフォルトで、クラスター内のすべての Ingress コントローラーはクラスター内の任意のプロジェクトで作成されたすべてのルートを許可します。

Ingress コントローラー:

- デフォルトでは 2 つのレプリカがあるので、これは 2 つのワーカーノードで実行する必要があります。
- 追加のノードにレプリカを組み込むためにスケールアップすることができます。

注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

15.3.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが 1 名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- OpenShift Container Platform クラスターを、1 つ以上のマスターと 1 つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブ

セットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

15.3.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. サービスの新規プロジェクトを作成します。

```
$ oc new-project <project_name>
```

以下は例になります。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。以下は例になります。

```
$ oc new-app \
    -e MYSQL_USER=admin \
    -e MYSQL_PASSWORD=redhat \
    -e MYSQL_DATABASE=mysqldb \
    registry.redhat.io/rhscl/mysql-80-rhel7
```

3. 以下のコマンドを実行して新規サービスが作成されていることを確認します。

```
$ oc get svc -n myproject
```

出力例

```
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7   ClusterIP   172.30.63.31   <none>        3306/TCP   4m55s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

15.3.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。

2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project myproject
```

3. 以下のコマンドを実行してルートを公開します。

```
$ oc expose service <service_name>
```

以下は例になります。

```
$ oc expose service mysql-80-rhel7
```

出力例

```
route "mysql-80-rhel7" exposed
```

4. cURL などのツールを使用し、サービスのクラスター IP アドレスを使用してサービスに到達できることを確認します。

```
$ curl <pod_ip>:<port>
```

以下に例を示します。

```
$ curl 172.30.131.89:3306
```

このセクションの例では、クライアントアプリケーションを必要とする MySQL サービスを使用しています。**Got packets out of order** のメッセージと共に文字ストリングを取得する場合は、このサービスに接続されていることになります。

MySQL クライアントがある場合は、標準 CLI コマンドでログインします。

```
$ mysql -h 172.30.131.89 -u admin -p
```

出力例

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

15.3.5. ルートラベルを使用した Ingress コントローラーのシャード化の設定

ルートラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーがルートセレクターによって選択される任意 namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

```
# cat router-internal.yaml
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    routeSelector:
      matchLabels:
        type: sharded
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace のルートを選択します。

15.3.6. namespace ラベルを使用した Ingress コントローラーのシャード化の設定

namespace ラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーが namespace セレクターによって選択される任意の namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

```
# cat router-internal.yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
```

```
    kind: IngressController
    metadata:
      name: sharded
      namespace: openshift-ingress-operator
    spec:
      domain: <apps-sharded.basedomain.example.net>
      nodePlacement:
        nodeSelector:
          matchLabels:
            node-role.kubernetes.io/worker: ""
      namespaceSelector:
        matchLabels:
          type: sharded
    status: {}
  kind: List
  metadata:
    resourceVersion: ""
    selfLink: ""
```

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

```
# oc apply -f router-internal.yaml
```

Ingress コントローラーは、**type: sharded** というラベルのある namespace セレクターによって選択される namespace のルートを選択します。

15.3.7. 関連情報

- Ingress Operator はワイルドカード DNS を管理します。詳細は、OpenShift Container Platform の Ingress Operator 、Installing a cluster on bare metal 、および Installing a cluster on vSphere を参照してください。

15.4. ロードバランサーを使用した INGRESS クラスターの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法では、ロードバランサーを使用します。

15.4.1. ロードバランサーを使用したトラフィックのクラスターへの送信

特定の外部 IP アドレスを必要としない場合、ロードバランサーサービスを OpenShift Container Platform クラスターへの外部アクセスを許可するよう設定することができます。

ロードバランサーサービスは固有の IP を割り当てます。ロードバランサーには単一の edge ルーター IP があります (これは仮想 IP (VIP) の場合もありますが、初期の負荷分散では単一マシンになります。

注記

プールが設定される場合、これはクラスター管理者によってではなく、インフラストラクチャーレベルで実行されます。

注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

15.4.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが 1 名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- OpenShift Container Platform クラスターを、1 つ以上のマスターと 1 つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

15.4.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. サービスの新規プロジェクトを作成します。

```
$ oc new-project <project_name>
```

以下は例になります。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。以下は例になります。

```
$ oc new-app \
    -e MYSQL_USER=admin \
    -e MYSQL_PASSWORD=redhat \
    -e MYSQL_DATABASE=mysqldb \
    registry.redhat.io/rhscl/mysql-80-rhel7
```

3. 以下のコマンドを実行して新規サービスが作成されていることを確認します。

```
$ oc get svc -n myproject
```

出力例

```
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7   ClusterIP   172.30.63.31   <none>       3306/TCP   4m55s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

15.4.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project myproject
```

3. 以下のコマンドを実行してルートを公開します。

```
$ oc expose service <service_name>
```

以下は例になります。

```
$ oc expose service mysql-80-rhel7
```

出力例

```
route "mysql-80-rhel7" exposed
```

4. cURL などのツールを使用し、サービスのクラスター IP アドレスを使用してサービスに到達できることを確認します。

```
$ curl <pod_ip>:<port>
```

以下に例を示します。

```
$ curl 172.30.131.89:3306
```

このセクションの例では、クライアントアプリケーションを必要とする MySQL サービスを使用しています。**Got packets out of order** のメッセージと共に文字ストリングを取得する場合は、このサービスに接続されていることになります。

MySQL クライアントがある場合は、標準 CLI コマンドでログインします。

```
$ mysql -h 172.30.131.89 -u admin -p
```

出力例

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
```

```
MySQL [(none)]>
```

15.4.5. ロードバランサーサービスの作成

以下の手順を使用して、ロードバランサーサービスを作成します。

前提条件

- 公開するプロジェクトとサービスがあること。

手順

ロードバランサーサービスを作成するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトを読み込みます。

```
$ oc project project1
```

3. マスターノードでテキストファイルを開き、以下のテキストを貼り付け、必要に応じてファイルを編集します。

ロードバランサー設定ファイルのサンプル

```
apiVersion: v1
kind: Service
metadata:
  name: egress-2 (1)
spec:
  ports:
  - name: db
    port: 3306 (2)
  loadBalancerIP:
  type: LoadBalancer (3)
  selector:
    name: mysql (4)
```

(1) ロードバランサーサービスの説明となる名前を入力します。

(2) 公開するサービスがリッスンしている同じポートを入力します。

(3) タイプに **loadbalancer** を入力します。

(4) サービスの名前を入力します。

4. ファイルを保存し、終了します。
5. 以下のコマンドを実行してサービスを作成します。

```
$ oc create -f <file-name>
```

以下に例を示します。

```
$ oc create -f mysql-lb.yaml
```

6. 以下のコマンドを実行して新規サービスを表示します。

```
$ oc get svc
```

出力例

```
NAME       TYPE           CLUSTER-IP      EXTERNAL-IP                          PORT(S)
AGE
egress-2   LoadBalancer   172.30.22.226   ad42f5d8b303045-487804948.example.com
3306:30357/TCP   15m
```

有効にされたクラウドプロバイダーがある場合、サービスには外部 IP アドレスが自動的に割り当てられます。

7. マスターで cURL などのツールを使用し、パブリック IP アドレスを使用してサービスに到達できることを確認します。

```
$ curl <public-ip>:<port>
```

以下に例を示します。

```
$ curl 172.29.121.74:3306
```

このセクションの例では、クライアントアプリケーションを必要とする MySQL サービスを使用しています。**Got packets out of order** のメッセージと共に文字ストリングを取得する場合は、このサービスに接続していることになります。

MySQL クライアントがある場合は、標準 CLI コマンドでログインします。

```
$ mysql -h 172.30.131.89 -u admin -p
```

出力例

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MySQL [(none)]>
```

15.5. サービスの外部 IP を使用した INGRESS クラスタートラフィックの設定

外部 IP アドレスをサービスに割り当てることで、これをクラスター外のトラフィックで使用できるようにします。通常、これはベアメタルハードウェアにインストールされているクラスターの場合にのみ役立ちます。外部ネットワークインフラストラクチャーは、トラフィックをサービスにルーティングするように正しく設定される必要があります。

15.5.1. 前提条件

- クラスターは ExternalIP が有効にされた状態で設定されます。詳細は、サービスの ExternalIP の設定 について参照してください。

15.5.2. ExternalIP のサービスへの割り当て

ExternalIP をサービスに割り当てることができます。クラスターが ExternalIP を自動的に割り当てするように設定されている場合、ExternalIP をサービスに手動で割り当てる必要がない場合があります。

手順

1. オプション: ExternalIP で使用するために設定される IP アドレス範囲を確認するには、以下のコマンドを入力します。

```
$ oc get networks.config cluster -o jsonpath='{.spec.externalIP}{"\n"}'
```

autoAssignCIDRs が設定されている場合、**spec.externalIPs** フィールドが指定されていない場合、 OpenShift Container Platform は ExternalIP を新規 **Service** オブジェクトに自動的に割り当てます。

2. ExternalIP をサービスに割り当てます。

   a. 新規サービスを作成する場合は、**spec.externalIPs** フィールドを指定し、1 つ以上の有効な IP アドレスの配列を指定します。以下に例を示します。

```
apiVersion: v1
kind: Service
metadata:
  name: svc-with-externalip
spec:
  ...
  externalIPs:
  - 192.174.120.10
```

   b. ExternalIP を既存のサービスに割り当てる場合は、以下のコマンドを入力します。**<name>** をサービス名に置き換えます。**<ip_address>** を有効な ExternalIP アドレスに置き換えます。コンマで区切られた複数の IP アドレスを指定できます。

```
$ oc patch svc <name> -p \
  '{
    "spec": {
      "externalIPs": [ "<ip_address>" ]
    }
  }'
```

以下に例を示します。

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
```

出力例

```
"mysql-55-rhel7" patched
```

3. ExternalIP アドレスがサービスに割り当てられていることを確認するには、以下のコマンドを入力します。新規サービスに ExternalIP を指定した場合、まずサービスを作成する必要があります。

```
$ oc get svc
```

出力例

```
NAME            CLUSTER-IP      EXTERNAL-IP     PORT(S)   AGE
mysql-55-rhel7   172.30.131.89   192.174.120.10  3306/TCP  13m
```

15.5.3. 関連情報

- サービスの ExternalIP の設定

15.6. NODEPORT を使用した INGRESS クラスタートラフィックの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法は **NodePort** を使用します。

15.6.1. NodePort を使用したトラフィックのクラスターへの送信

NodePort-type **Service** リソースを使用して、クラスター内のすべてのノードの特定のポートでサービスを公開します。ポートは **Service** リソースの **.spec.ports[*].nodePort** フィールドで指定されます。

重要

ノードポートを使用するには、追加のポートリソースが必要です。

NodePort は、ノードの IP アドレスの静的ポートでサービスを公開します。**NodePort** はデフォルトで **30000** から **32767** の範囲に置かれます。つまり、 **NodePort** はサービスの意図されるポートに一致しないことが予想されます。たとえば、ポート **8080** はノードのポート **31020** として公開できます。

管理者は、外部 IP アドレスがノードにルーティングされることを確認する必要があります。

NodePort および外部 IP は独立しており、両方を同時に使用できます。

注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

15.6.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが 1 名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
```

- OpenShift Container Platform クラスターを、1 つ以上のマスターと 1 つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

15.6.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. サービスの新規プロジェクトを作成します。

```
$ oc new-project <project_name>
```

以下は例になります。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。以下は例になります。

```
$ oc new-app \
    -e MYSQL_USER=admin \
    -e MYSQL_PASSWORD=redhat \
    -e MYSQL_DATABASE=mysqldb \
    registry.redhat.io/rhscl/mysql-80-rhel7
```

3. 以下のコマンドを実行して新規サービスが作成されていることを確認します。

```
$ oc get svc -n myproject
```

出力例

```
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql-80-rhel7  ClusterIP   172.30.63.31   <none>        3306/TCP   4m55s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

15.6.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project myproject
```

3. アプリケーションのノードポートを公開するには、以下のコマンドを入力します。OpenShift Container Platform は **30000-32767** 範囲の利用可能なポートを自動的に選択します。

```
$ oc expose dc mysql-80-rhel7 --type=NodePort --name=mysql-ingress
```

4. オプション: サービスが公開されるノードポートで利用可能なことを確認するには、以下のコマンドを入力します。

```
$ oc get svc -n myproject
```

出力例

```
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
mysql-80-rhel7   ClusterIP   172.30.217.127   <none>        3306/TCP         9m44s
mysql-ingress    NodePort    172.30.107.72    <none>        3306:31345/TCP   39s
```

5. オプション: **oc new-app** コマンドによって自動的に作成されたサービスを削除するには、以下のコマンドを入力します。

```
$ oc delete svc mysql-80-rhel7
```

第16章 クラスター全体のプロキシーの設定

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシーを使用することができます。既存クラスターのプロキシーオブジェクトを変更 するか、または新規クラスターの **install-config.yaml** ファイルでプロキシー設定を行うことにより、OpenShift Container Platform をプロキシーを使用するように設定できます。

重要

クラスター全体のプロキシーは、ユーザーによってプロビジョニングされるインフラストラクチャーのインストールを使用している場合や、サポートされるプロバイダーに、仮想プライベートクラウドや仮想ネットワークなどの独自のネットワークを提供する場合にのみサポートされます。

16.1. 前提条件

- クラスターがアクセスする必要のあるサイト を確認し、プロキシーをバイパスする必要があるかどうかを判断します。デフォルトで、すべてのクラスターシステムの egress トラフィック (クラスターをホストするクラウドのクラウドプロバイダー API に対する呼び出しを含む) はプロキシーされます。システム全体のプロキシーは、ユーザーのワークロードではなく、システムコンポーネントにのみ影響を与えます。プロキシーオブジェクトの **spec.noProxy** フィールドにサイトを追加し、必要に応じてプロキシーをバイパスします。

注記

Proxy オブジェクトの **status.noProxy** フィールドには、インストール設定の **networking.machineNetwork[].cidr**、 **networking.clusterNetwork[].cidr**、および **networking.serviceNetwork[]** フィールドの値が設定されます。

Amazon Web Services (AWS)、Google Cloud Platform (GCP)、Microsoft Azure、および Red Hat OpenStack Platform (RHOSP) へのインストールの場合、**Proxy** オブジェクトの **status.noProxy** フィールドには、インスタンスメタデータのエンドポイント (**169.254.169.254**) も設定されます。

16.2. クラスター全体のプロキシーの有効化

プロキシーオブジェクトは、クラスター全体の egress プロキシーを管理するために使用されます。プロキシーを設定せずにクラスターがインストールまたはアップグレードされると、プロキシーオブジェクトは引き続き生成されますが、**spec** は設定されません。以下に例を示します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

クラスター管理者は、この **cluster** プロキシーオブジェクトを変更して OpenShift Container Platform のプロキシーを設定できます。

注記

cluster という名前のプロキシーオブジェクトのみがサポートされ、追加のプロキシーは作成できません。

前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる ConfigMap を作成します。

   注記

   プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名される場合は、これを省略できます。

   a. 以下の内容で **user-ca-bundle.yaml** というファイルを作成して、PEM でエンコードされた証明書の値を指定します。

   ```
   apiVersion: v1
   data:
     ca-bundle.crt: | (1)
       <MY_PEM_ENCODED_CERTS> (2)
   kind: ConfigMap
   metadata:
     name: user-ca-bundle (3)
     namespace: openshift-config (4)
   ```

   (1) このデータキーは **ca-bundle.crt** という名前にする必要があります。

   (2) プロキシーのアイデンティティー証明書に署名するために使用される 1 つ以上の PEM でエンコードされた X.509 証明書。

   (3) プロキシーオブジェクトから参照される ConfigMap 名。

   (4) ConfigMap は **openshift-config** namespace になければなりません。

   b. このファイルから ConfigMap を作成します。

   ```
   $ oc create -f user-ca-bundle.yaml
   ```

2. **oc edit** コマンドを使用してプロキシーオブジェクトを変更します。

   ```
   $ oc edit proxy/cluster
   ```

3. プロキシーに必要なフィールドを設定します。

   ```
   apiVersion: config.openshift.io/v1
   kind: Proxy
   ```

```
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> (1)
  httpsProxy: http://<username>:<pswd>@<ip>:<port> (2)
  noProxy: example.com (3)
  readinessEndpoints:
  - http://www.google.com (4)
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle (5)
```

1. クラスター外の HTTP 接続を作成するために使用するプロキシー URL。URL スキームは **http** である必要があります。

2. クラスター外で HTTPS 接続を作成するために使用するプロキシー URL。これが指定されていない場合、HTTP および HTTPS 接続の両方に **httpProxy** が使用されます。

3. プロキシーを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のコンマ区切りの一覧。

   サブドメインのみと一致するように、ドメインの前に **.** を付けます。たとえば、**.y.com** は **x.y.com** に一致しますが、 **y.com** には一致しません。* を使用し、すべての宛先のプロキシーをバイパスします。インストール設定で **networking.machineNetwork[].cidr** フィールドで定義されるネットワークに含まれていないワーカーをスケールアップする場合、それらをこの一覧に追加し、接続の問題を防ぐ必要があります。

   httpProxy または **httpsProxy** フィールドのいずれも設定されていない場合に、このフィールドは無視されます。

4. **httpProxy** および **httpsProxy** の値をステータスに書き込む前の readiness チェックに使用するクラスター外の 1 つ以上の URL。

5. HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる、**openshift-config** namespace の ConfigMap の参照。ここで参照する前に ConfigMap が存在している必要があります。このフィールドは、プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

4. 変更を適用するためにファイルを保存します。

注記

URL スキームは **http** である必要があります。**https** スキームは現在サポートされていません。

16.3. クラスター全体のプロキシーの削除

cluster プロキシーオブジェクトは削除できません。クラスターからプロキシーを削除するには、プロキシーオブジェクトからすべての **spec** フィールドを削除します。

前提条件

- クラスター管理者のパーミッション。

- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. **oc edit** コマンドを使用してプロキシーを変更します。

```
$ oc edit proxy/cluster
```

2. プロキシーオブジェクトからすべての **spec** フィールドを削除します。以下に例を示します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec: {}
status: {}
```

3. 変更を適用するためにファイルを保存します。

第17章 カスタム PKI の設定

Web コンソールなどの一部のプラットフォームコンポーネントは、通信にルートを使用し、それらと対話するために他のコンポーネントの証明書を信頼する必要があります。カスタムのパブリックキーインフラストラクチャー (PKI) を使用している場合は、プライベートに署名された CA 証明書がクラスター全体で認識されるようにこれを設定する必要があります。

プロキシー API を使用して、クラスター全体で信頼される CA 証明書を追加できます。インストール時またはランタイム時にこれを実行する必要があります。

- **インストール** 時に、クラスター全体のプロキシーを設定します。プライベートに署名された CA 証明書は、**install-config.yaml** ファイルの **additionalTrustBundle** 設定で定義する必要があります。
  インストールプログラムは、定義した追加の CA 証明書が含まれる **user-ca-bundle** という名前の ConfigMap を生成します。次に Cluster Network Operator は、これらの CA 証明書を Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージする **trusted-ca-bundle** ConfigMap を作成し、この ConfigMap はプロキシーオブジェクトの **trustedCA** フィールドで参照されます。

- **ランタイム** 時に、デフォルトのプロキシーオブジェクトを変更して、プライベートに署名された CA 証明書を追加 します (これは、クラスターのプロキシー有効化のワークフローの一部です)。これには、クラスターで信頼される必要があるプライベートに署名された CA 証明書が含まれる ConfigMap を作成し、次にプライベートに署名された証明書の ConfigMap を参照する **trustedCA** でプロキシーリソースを変更することが関係します。

注記

インストーラー設定の **additionalTrustBundle** フィールドおよびプロキシーリソースの **trustedCA** フィールドは、クラスター全体の信頼バンドルを管理するために使用されます。 **additionalTrustBundle** はインストール時に使用され、プロキシーの **trustedCA** がランタイム時に使用されます。

trustedCA フィールドは、クラスターコンポーネントによって使用されるカスタム証明書とキーのペアを含む **ConfigMap** の参照です。

17.1. インストール時のクラスター全体のプロキシーの設定

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシーを使用することができます。プロキシー設定を **install-config.yaml** ファイルで行うことにより、新規の OpenShift Container Platform クラスターをプロキシーを使用するように設定できます。

前提条件

- 既存の **install-config.yaml** ファイルが必要です。

- クラスターがアクセスする必要のあるサイトを確認し、プロキシーをバイパスする必要があるかどうかを判別します。デフォルトで、すべてのクラスター egress トラフィック (クラスターをホストするクラウドについてのクラウドプロバイダー API に対する呼び出しを含む) はプロキシーされます。**Proxy** オブジェクトの **spec.noProxy** フィールドにサイトを追加し、必要に応じてプロキシーをバイパスします。

注記

Proxy オブジェクトの **status.noProxy** フィールドには、インストール設定の **networking.machineNetwork[].cidr**、 **networking.clusterNetwork[].cidr**、および **networking.serviceNetwork[]** フィールドの値が設定されます。

Amazon Web Services (AWS)、Google Cloud Platform (GCP)、Microsoft Azure、および Red Hat OpenStack Platform (RHOSP) へのインストールの場合、**Proxy** オブジェクトの **status.noProxy** フィールドには、インスタンスメタデータのエンドポイント (**169.254.169.254**) も設定されます。

手順

1. **install-config.yaml** ファイルを編集し、プロキシー設定を追加します。以下に例を示します。

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> (1)
  httpsProxy: http://<username>:<pswd>@<ip>:<port> (2)
  noProxy: example.com (3)
additionalTrustBundle: | (4)
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
...
```

(1) クラスター外の HTTP 接続を作成するために使用するプロキシー URL。URL スキームは **http** である必要があります。追加のプロキシー設定が必要ではなく、追加の CA を必要とする MITM の透過的なプロキシーネットワークを使用する場合には、**httpProxy** 値を指定することはできません。

(2) クラスター外で HTTPS 接続を作成するために使用するプロキシー URL。このフィールドが指定されていない場合、HTTP および HTTPS 接続の両方に **httpProxy** が使用されます。追加のプロキシー設定が必要ではなく、追加の CA を必要とする MITM の透過的なプロキシーネットワークを使用する場合には、**httpsProxy** 値を指定することはできません。

(3) プロキシーを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のコンマ区切りの一覧。サブドメインのみと一致するように、ドメインの前に **.** を付けます。たとえば、**.y.com** は **x.y.com** に一致しますが、 **y.com** には一致しません。* を使用し、すべての宛先のプロキシーをバイパスします。

(4) 指定されている場合、インストールプログラムは HTTPS 接続のプロキシーに必要な 1 つ以上の追加の CA 証明書が含まれる **user-ca-bundle** という名前の設定マップを **openshift-config** namespace に生成します。次に Cluster Network Operator は、これらのコンテンツを Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージする **trusted-ca-bundle** 設定マップを作成し、この設定マップは **Proxy** オブジェクトの **trustedCA** フィールドで参照されます。**additionalTrustBundle** フィールドは、プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。追加のプロキシー設定が必要ではなく、追加の CA を必要とする MITM の透過的なプロキシーネットワークを使用する場合には、MITM CA 証明書を指定する必要があります。

注記

インストールプログラムは、プロキシーの **readinessEndpoints** フィールドをサポートしません。

2. ファイルを保存し、OpenShift Container Platform のインストール時にこれを参照します。

インストールプログラムは、指定の **install-config.yaml** ファイルのプロキシー設定を使用する **cluster** という名前のクラスター全体のプロキシーを作成します。プロキシー設定が指定されていない場合、**cluster Proxy** オブジェクトが依然として作成されますが、これには **spec** がありません。

注記

cluster という名前の **Proxy** オブジェクトのみがサポートされ、追加のプロキシーを作成することはできません。

17.2. クラスター全体のプロキシーの有効化

プロキシーオブジェクトは、クラスター全体の egress プロキシーを管理するために使用されます。プロキシーを設定せずにクラスターがインストールまたはアップグレードされると、プロキシーオブジェクトは引き続き生成されますが、**spec** は設定されません。以下に例を示します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

クラスター管理者は、この **cluster** プロキシーオブジェクトを変更して OpenShift Container Platform のプロキシーを設定できます。

注記

cluster という名前のプロキシーオブジェクトのみがサポートされ、追加のプロキシーは作成できません。

前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる ConfigMap を作成します。

注記

プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名される場合は、これを省略できます。

a. 以下の内容で **user-ca-bundle.yaml** というファイルを作成して、PEM でエンコードされた証明書の値を指定します。

```
apiVersion: v1
data:
  ca-bundle.crt: | (1)
    <MY_PEM_ENCODED_CERTS> (2)
kind: ConfigMap
metadata:
  name: user-ca-bundle (3)
  namespace: openshift-config (4)
```

1. このデータキーは **ca-bundle.crt** という名前にする必要があります。
2. プロキシーのアイデンティティー証明書に署名するために使用される 1 つ以上の PEM でエンコードされた X.509 証明書。
3. プロキシーオブジェクトから参照される ConfigMap 名。
4. ConfigMap は **openshift-config** namespace になければなりません。

b. このファイルから ConfigMap を作成します。

```
$ oc create -f user-ca-bundle.yaml
```

2. **oc edit** コマンドを使用してプロキシーオブジェクトを変更します。

```
$ oc edit proxy/cluster
```

3. プロキシーに必要なフィールドを設定します。

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port> (1)
  httpsProxy: http://<username>:<pswd>@<ip>:<port> (2)
  noProxy: example.com (3)
  readinessEndpoints:
  - http://www.google.com (4)
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle (5)
```

1. クラスター外の HTTP 接続を作成するために使用するプロキシー URL。URL スキームは **http** である必要があります。
2. クラスター外で HTTPS 接続を作成するために使用するプロキシー URL。これが指定されていない場合、HTTP および HTTPS 接続の両方に **httpProxy** が使用されます。
3. プロキシーを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のコンマ区切りの一覧。

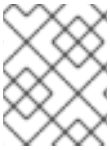
サブドメインのみと一致するように、ドメインの前に **.** を付けます。たとえば、**.y.com** は **x.y.com** に一致しますが、 **y.com** には一致しません。***** を使用し、すべての宛先のプロキシーをバイパスします。インストール設定で **networking.machineNetwork[].cidr** フィールドで定義されるネットワークに含まれていないワーカーをスケールアップする場合、それらをこの一覧に追加し、接続の問題を防ぐ必要があります。

httpProxy または **httpsProxy** フィールドのいずれも設定されていない場合に、このフィールドは無視されます。

4. **httpProxy** および **httpsProxy** の値をステータスに書き込む前の readiness チェックに使用するクラスター外の 1 つ以上の URL。

5. HTTPS 接続のプロキシーに必要な追加の CA 証明書が含まれる、**openshift-config** namespace の ConfigMap の参照。ここで参照する前に ConfigMap が存在している必要があります。このフィールドは、プロキシーのアイデンティティー証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

4. 変更を適用するためにファイルを保存します。



注記

URL スキームは **http** である必要があります。**https** スキームは現在サポートされていません。

17.3. OPERATOR を使用した証明書の挿入

カスタム CA 証明書が ConfigMap 経由でクラスターに追加されると、Cluster Network Operator はユーザーによってプロビジョニングされる CA 証明書およびシステム CA 証明書を単一バンドルにマージし、信頼バンドルの挿入を要求する Operator にマージされたバンドルを挿入します。

Operator は、以下のラベルの付いた空の ConfigMap を作成してこの挿入を要求します。

```
config.openshift.io/inject-trusted-cabundle="true"
```

Operator は、この ConfigMap をコンテナーのローカル信頼ストアにマウントします。

注記

信頼された CA 証明書の追加は、証明書が Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルに含まれない場合にのみ必要になります。

証明書の挿入は Operator に制限されません。Cluster Network Operator は、空の ConfigMap が **config.openshift.io/inject-trusted-cabundle=true** ラベルを使用して作成される場合に、すべての namespace で証明書を挿入できます。

ConfigMap はすべての namespace に置くことができますが、ConfigMap はカスタム CA を必要とする Pod 内の各コンテナーに対してボリュームとしてマウントされる必要があります。以下は例になります。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
```

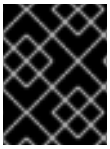
```
  namespace: my-example-custom-ca-ns
spec:
  ...
    spec:
      ...
      containers:
        - name: my-container-that-needs-custom-ca
          volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            readOnly: true
      volumes:
      - name: trusted-ca
        configMap:
          name: trusted-ca
          items:
            - key: ca-bundle.crt (1)
              path: tls-ca-bundle.pem (2)
```

1. **ca-bundle.crt** は ConfigMap キーとして必要になります。
2. **tls-ca-bundle.pem** は ConfigMap パスとして必要になります。

第18章 RHOSP での負荷分散

18.1. KURYR SDN を使用した OCTAVIA OVN ロードバランサープロバイダードライバーの使用

OpenShift Container Platform クラスターが Kuryr を使用し、これが後に RHOSP 16 にアップグレードされた Red Hat OpenStack Platform (RHOSP) 13 クラウドにインストールされている場合、これを Octavia OVN プロバイダードライバーを使用するように設定できます。



重要

Kuryr はプロバイダードライバーの変更後に既存のロードバランサーを置き換えます。このプロセスにより、ダウンタイムが生じます。

前提条件

- RHOSP CLI の **openstack** をインストールします。
- OpenShift Container Platform CLI の **oc** をインストールします。
- RHOSP の Octavia OVN ドライバーが有効になっていることを確認します。

ヒント

利用可能な Octavia ドライバーの一覧を表示するには、コマンドラインで **openstack loadbalancer provider list** を入力します。

ovn ドライバーはコマンドの出力に表示されます。

手順

Octavia Amphora プロバイダードライバーから Octavia OVN に変更するには、以下を実行します。

1. **kuryr-config** ConfigMap を開きます。コマンドラインで、以下を実行します。

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

2. ConfigMap で、**kuryr-octavia-provider: default** が含まれる行を削除します。以下に例を示します。

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: default (1)
...
```

(1) この行を削除します。クラスターは、**ovn** を値としてこれを再生成します。

Cluster Network Operator が変更を検出し、**kuryr-controller** および **kuryr-cni** Pod を再デプロイするのを待機します。このプロセスには数分の時間がかかる可能性があります。

3. **kuryr-config** ConfigMap アノテーションで **ovn** をその値として表示されていることを確認します。コマンドラインで、以下を実行します。

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

ovn プロバイダーの値は出力に表示されます。

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: ovn
...
```

4. RHOSP がそのロードバランサーを再作成していることを確認します。

   a. コマンドラインで、以下を実行します。

```
$ openstack loadbalancer list | grep amphora
```

   単一の Amphora ロードバランサーが表示されます。以下に例を示します。

```
a4db683b-2b7b-4988-a582-c39daaad7981 | ostest-7mbj6-kuryr-api-loadbalancer  |
84c99c906edd475ba19478a9a6690efd | 172.30.0.1     | ACTIVE              | amphora
```

   b. 以下を入力して **ovn** ロードバランサーを検索します。

```
$ openstack loadbalancer list | grep ovn
```

   ovn タイプの残りのロードバランサーが表示されます。以下に例を示します。

```
2dffe783-98ae-4048-98d0-32aa684664cc | openshift-apiserver-operator/metrics |
84c99c906edd475ba19478a9a6690efd | 172.30.167.119 | ACTIVE              | ovn
0b1b2193-251f-4243-af39-2f99b29d18c5 | openshift-etcd/etcd                 |
84c99c906edd475ba19478a9a6690efd | 172.30.143.226 | ACTIVE              | ovn
f05b07fc-01b7-4673-bd4d-adaa4391458e | openshift-dns-operator/metrics      |
84c99c906edd475ba19478a9a6690efd | 172.30.152.27  | ACTIVE              | ovn
```

18.2. OCTAVIA を使用したアプリケーショントラフィック用のクラスターのスケーリング

Red Hat OpenStack Platform (RHOSP) で実行される OpenShift Container Platform クラスターでは、Octavia 負荷分散サービスを使用して、複数の仮想マシン (VM) または Floating IP アドレスにトラフィックを分散することができます。この機能は、単一マシンまたはアドレスが生じさせるボトルネックを軽減します。

クラスターで Kuryr を使用する場合、Cluster Network Operator はデプロイメント時に内部 Octavia ロードバランサーを作成していました。アプリケーションネットワークのスケーリングには、このロードバランサーを使用できます。

クラスターで Kuryr を使用しない場合、アプリケーションのネットワークのスケーリングに使用する独自の Octavia ロードバランサーを作成する必要があります。

18.2.1. Octavia を使用したクラスターのスケーリング

複数の API ロードバランサーを使用する場合や、クラスターが Kuryr を使用しない場合、Octavia ロードバランサーを作成してから、クラスターをこれを使用するように設定します。

前提条件

- Octavia は Red Hat OpenStack Platform (RHOSP) デプロイメントで利用できます。

手順

1. コマンドラインから、Amphora ドライバーを使用する Octavia ロードバランサーを作成します。

```
$ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
<id_of_worker_vms_subnet>
```

   API_OCP_CLUSTER の代わりに、任意の名前を使用することができます。

2. ロードバランサーがアクティブになったら、リスナーを作成します。

```
$ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
HTTPS--protocol-port 6443 API_OCP_CLUSTER
```

   注記

   ロードバランサーのステータスを表示するには、**openstack loadbalancer list** と入力します。

3. ラウンドロビンアルゴリズムを使用し、セッションの永続性が有効にされているプールを作成します。

```
$ openstack loadbalancer pool create --name API_OCP_CLUSTER_pool_6443 --lb-
algorithm ROUND_ROBIN --session-persistence type=<source_IP_address> --listener
API_OCP_CLUSTER_6443 --protocol HTTPS
```

4. コントロールプレーンマシンが利用可能であることを確認するには、ヘルスモニターを作成します。

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --type
TCP API_OCP_CLUSTER_pool_6443
```

5. コントロールプレーンマシンをロードバランサープールのメンバーとして追加します。

```
$ for SERVER in $(MASTER-0-IP MASTER-1-IP MASTER-2-IP)
do
  openstack loadbalancer member create --address $SERVER  --protocol-port 6443
API_OCP_CLUSTER_pool_6443
done
```

6. オプション: クラスター API の Floating IP アドレスを再利用するには、設定を解除します。

```
$ openstack floating ip unset $API_FIP
```

7. 設定を解除された **API_FIP**、または新規アドレスを、作成されたロードばランサー VIP に追加します。

```
$ openstack floating ip set  --port $(openstack loadbalancer show -c <vip_port_id> -f value
API_OCP_CLUSTER) $API_FIP
```

クラスターは、負荷分散に Octavia を使用するようになりました。

注記

Kuryr が Octavia Amphora ドライバーを使用する場合、すべてのトラフィックは単一の Amphora 仮想マシン (VM) 経由でルーティングされます。

この手順を繰り返して追加のロードバランサーを作成します。これにより、ボトルネックを軽減することができます。

18.2.2. Octavia の使用による Kuryr を使用するクラスターのスケーリング

クラスターで Kuryr を使用する場合は、クラスターの API Floating IP アドレスを既存の Octavia ロードバランサーに関連付けます。

前提条件

- OpenShift Container Platform クラスターは Kuryr を使用します。
- Octavia は Red Hat OpenStack Platform (RHOSP) デプロイメントで利用できます。

手順

1. オプション: コマンドラインからクラスター API の Floating IP アドレスを再利用するには、この設定を解除します。

```
$ openstack floating ip unset $API_FIP
```

2. 設定を解除された **API_FIP**、または新規アドレスを、作成されたロードばランサー VIP に追加します。

```
$ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value
${OCP_CLUSTER}-kuryr-api-loadbalancer) $API_FIP
```

クラスターは、負荷分散に Octavia を使用するようになりました。

注記

Kuryr が Octavia Amphora ドライバーを使用する場合、すべてのトラフィックは単一の Amphora 仮想マシン (VM) 経由でルーティングされます。

この手順を繰り返して追加のロードバランサーを作成します。これにより、ボトルネックを軽減することができます。

18.3. RHOSP OCTAVIA を使用した INGRESS トラフィックのスケーリング

Octavia ロードバランサーを使用して、Kuryr を使用するクラスターで Ingress コントローラーをスケーリングできます。

前提条件

- OpenShift Container Platform クラスターは Kuryr を使用します。
- Octavia は RHOSP デプロイメントで利用できます。

手順

1. 現在の内部ルーターサービスをコピーするには、コマンドラインで以下を入力します。

```
$ oc -n openshift-ingress get svc router-internal-default -o yaml > external_router.yaml
```

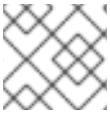
2. **external_router.yaml** ファイルで、**metadata.name** および **spec.type** の値を **LoadBalancer** に変更します。

ルーターファイルの例

```
apiVersion: v1
kind: Service
metadata:
  labels:
    ingresscontroller.operator.openshift.io/owning-ingresscontroller: default
  name: router-external-default (1)
  namespace: openshift-ingress
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: http
  - name: https
    port: 443
    protocol: TCP
    targetPort: https
  - name: metrics
    port: 1936
    protocol: TCP
    targetPort: 1936
  selector:
    ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
  sessionAffinity: None
  type: LoadBalancer (2)
```

(1) この値は **router-external-default** のように記述的であることを確認します。

(2) この値は **LoadBalancer** であることを確認します。



注記

ロードバランシングと関連性のないタイムスタンプやその他の情報を削除できます。

1. コマンドラインで、**external_router.yaml** ファイルからサービスを作成します。

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

2. サービスの外部 IP アドレスがロードバランサーに関連付けられているものと同じであることを確認します。

   a. コマンドラインで、サービスの外部 IP アドレスを取得します。

```
$ oc -n openshift-ingress get svc
```

出力例

```
NAME                      TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)
AGE
router-external-default   LoadBalancer   172.30.235.33    10.46.22.161
80:30112/TCP,443:32359/TCP,1936:30317/TCP   3m38s
router-internal-default   ClusterIP      172.30.115.123   <none>
80/TCP,443/TCP,1936/TCP                     22h
```

   b. ロードバランサーの IP アドレスを取得します。

```
$ openstack loadbalancer list | grep router-external
```

出力例

```
| 21bf6afe-b498-4a16-a958-3229e83c002c | openshift-ingress/router-external-default |
66f3816acf1b431691b8d132cc9d793c | 172.30.235.33  | ACTIVE | octavia |
```

   c. 直前のステップで取得したアドレスが、Floating IP の一覧で相互に関連付けられていることを確認します。

```
$ openstack floating ip list | grep 172.30.235.33
```

出力例

```
| e2f80e97-8266-4b69-8636-e58bacf1879e | 10.46.22.161 | 172.30.235.33 | 655e7122-
806a-4e0a-a104-220c6e17bda6 | a565e55a-99e7-4d15-b4df-f9d7ee8c9deb |
66f3816acf1b431691b8d132cc9d793c |
```

EXTERNAL-IP の値を新規 Ingress アドレスとして使用できるようになりました。

注記

Kuryr が Octavia Amphora ドライバーを使用する場合、すべてのトラフィックは単一の Amphora 仮想マシン (VM) 経由でルーティングされます。

この手順を繰り返して追加のロードバランサーを作成します。これにより、ボトルネックを軽減することができます。