ORACLE
Cloud Infrastructure

マルチVNICを使用したベア・メタル・インスタンスへのKVMのインストールと構成

ORACLE WHITE PAPER | 2018年2月

ORACLE

免責事項

下記事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクルの製品に関して記載されている機能の開発、リリース、および時期については、弊社の裁量により決定されます。

改訂履歴

本ホワイトペーパーは、初版の公開後、次の改訂がありました。

- 2018年2月15日：8ページの「ステップ1：ベア・メタル・インスタンスの準備」のステップ4に示しているコマンドの誤りを訂正

目次

概要

Oracle Cloud Infrastructureでは、KVMが事前構成されたパッケージ済のイメージは提供されません。それでも、オラクルとしては、お客様が環境内でKVMの使用を開始して、仮想マシンをクラウドに移行できるようにしたいと考えています。本ホワイトペーパーでは、ベア・メタル・インスタンスでKVMを使用する方法を詳細に説明します。

クラウド環境でKVMを提供することには、2つのメリットがあります。

- 既存のオンプレミスKVM環境をOracle Cloudに拡張できる。
- レガシーのオペレーティング・システムとパッケージ済の仮想マシンをインストールして環境内で使用できる。

KVMハイパーバイザを提供できるのはオラクル独自であり、それはOracle Cloud Infrastructureの複数の機能によって実現します。

- Block Volumeサービス
- ベア・メタルComputeインスタンス
- Networkingサービスにおける複数仮想ネットワーク・インタフェース・カード（複数VNIC）の機能

複数VNICをリリースする以前は、インスタンスが単一のネットワーク・インタフェースに限定されていました。この制限のため、ベア・メタル・インスタンスではハイパーバイザを使用できませんでした。ゲスト・オペレーティング・システムが、仮想クラウド・ネットワーク（VCN）内のサブネットに接続できなかったためです。複数VNIC機能を使用すると、ベア・メタル・インスタンスで最大16個のVNICを割り当てられるようになりました。これは、1つのKVM環境ではゲスト・オペレーティング・システムを16個までしかインストールできないということでもあります。

想定条件

本ホワイトペーパーで説明しているタスクを実行するには、以下の知識と情報が必要です。

- KVMについて十分な知識があり、このハイパーバイザを扱ううえでの基本概念を理解していること。
- ゲストがブロック・ストレージ・デバイスを共有することの影響を理解し、ゲストによるストレージ共有方法を決定できること。
- ゲストとしてオペレーティング・システムをインストールする方法を理解しているか、システム間で仮想ディスク・イメージをコピーする方法を知っていること。
- Linuxシステムの管理について実用的な知識を持ち、Linuxを自由に扱いながらファイルを編集できること。
- 環境内でVCNを作成してあり、そのVCNに1つ以上のサブネットをプロビジョニングしていること。

- Oracle Cloud Infrastructureのベア・メタルComputeインスタンスと、Block Volumeサービスの1つ以上のブロック・ストレージ・デバイスをプロビジョニングしていること（またはプロビジョニングの方法を知っていること）。
- Computeインスタンスがパブリック・サブネット（インターネットにアウトバンド・アクセスできるサブネット）上にあること。
- 仮想マシン・イメージをインポートする場合は、そのイメージがすでにKVM RAWまたはQEMU qcow2形式であると想定しています。本ホワイトペーパーでは、この変換の実行方法については説明しません。この変換の詳細については、qemu-imgのmanページを参照してください。

注意：本ホワイトペーパーの説明は、Oracle Linuxバージョン7.3を対象にしています。それ以外のディストリビューションで発生する問題についてはお答えいたしかねます。オペレーティング・システムのルート・ディスク・ストレージ要件、特にUEFIとBIOS、IDEとSCSIについてあらゆるバリエーションをテストしたわけではありません。オペレーティング・システムは、インストール中にドライバのスリップストリーム・インジェクションを実行する、またはイメージのエクスポート前にドライバをあらかじめインストールするなどの方法で、特定の起動方法をサポートすると想定します。また、ゲストのインストールまたはインポート後にそのゲストの機能をフルに利用するには、追加のドライバおよびKVMゲスト・ツールのインストールが必要になる場合があります。

前提条件

KVMを使用できるようにするには、以下のタスクを実行する必要があります。

- Oracle Linux 7.3でベア・メタルComputeインスタンスをプロビジョニングします。インスタンスにアクセスできることを確認してください。このインスタンスには**BM.Standard**シェイプを選択することをお薦めします。このプロセスではNVMeストレージについて何の要件もないからです。
- 少なくとも1つのブロック・ストレージ・デバイスをプロビジョニングし、プロビジョニングしてあるインスタンスにそれをアタッチします。KVM環境にインストールしたゲスト・オペレーティング・システムが、このブロック・ストレージ・デバイスで保持されます。ゲストをルート・ボリュームにインストールしないでください。ベア・メタル・インスタンスのルート・ボリュームは容量が46Gしかなく、ゲスト・オペレーティング・システムのインストールには不十分だからです。また、ブロック・ストレージはパフォーマンスが高くなります。ゲストに必要なファイルを格納できる十分な容量を確保するために、想定されるゲスト・オペレーティング・システムのサイズより大きいデバイスをプロビジョニングすることをお薦めします。各ゲストを別々のデバイスに格納する場合には、複数のデバイスをプロビジョニングすることもできます。
- 少なくとも1つのサブネットを完全に構成したVCNを作成します。これには、セキュリティ・リスト、インターネット・ゲートウェイ、適切なルート表が含まれます。セキュリティ・リストが、少なくとも1つのサブネット上のインターネットからSSH経由でハイパーバイザに接続できることを確認します。
- ハイパーバイザに使用するサブネット名およびOCIDと、ゲストに使用するサブネット名およびOCIDを利用できることを確認してください。ゲストは、ハイパーバイザと同じサブネット上にある必要はありません。これは、ハイパーバイザへのアクセスとゲストへのアクセスとの間に障壁を作成したい場合に便利です。

- 使用するベア・メタル・インスタンスに、1つ以上のVNICを追加でプロビジョニングします。このVNICは、ゲストが環境内の他のインスタンスと通信できるようにするために使用されます。VNICを（1つでも複数でも）プロビジョニングするとき、返されるIPアドレス/MAC/VLANタグの組合せに注意してください。
- ゲストの初期構成を実行できるように、デスクトップまたはラップトップにVNCクライアントをインストールします。当初、特に新しくインストールしたゲストを作成するときには、特定の構成タスクを実行できるように、ゲストにコンソールからアクセスできる必要があります。ただし、最近のほとんどのオペレーティング・システムでは、アクセス可能なインストール・プロセスにはGUI対応のコンソールを使用できると想定されています。仮想化APIのlibvirtには、ゲストに使用できる基本的なシリアル・コンソールが用意されていますが、GUIインタフェースはありません。この問題に対処するには、インストールの際に、ゲストごとにVNC対応コンソールを構成し、SSHトンネリングとVNCの組合せを使用することをお薦めします。ゲストをインストールするベア・メタル・インスタンスで使用されていないTCPポートを、59xx番台の範囲から特定します。
- ハイパーバイザとの通信およびファイル転送ができるように、SSHおよびSCPクライアントをインストールします。SSHクライアントにポート・トンネリング機能があることを確認してください。
- 既存の仮想マシン・イメージをインポートする場合は、そのイメージをKVM RAW形式またはQEMU qcow2形式に変換できることを確認します。詳細は、qemu-imgのmanページを参照してください。本ホワイトペーパーでは、インポートされるイメージがすでにRAWまたはqcow2形式であると想定しており、変換を実行するプロセスは扱いません。
- 新しいゲスト作成する予定がある場合には、ゲスト・イメージを保持するために作成する仮想ディスクのサイズを検討します。
- ゲストにどのくらいのRAM容量とCPU数を割り当てるのか、既存の仮想マシンをインポートするのか新しいゲストを作成するのかを特定します。テストで使用する**BM.Standard**シェイプは、256GBのRAMと32コアを使用します。
- ゲストとしてインストールするオペレーティング・システムまたはシステム・イメージの制限と機能を把握します。ネットワーク・インタフェースへの接続方法と、ゲストの動作に必要な他のドライバおよびソフトウェアは、オペレーティング・システムとシステム・イメージによって決まります。これは、オペレーティング・システムの多くの部分、CPUおよびRAMドライバ、ストレージ・デバイス、およびネットワークに当てはまります。Windows版のKVM固有ドライバは、次の項で説明するインストール・プロセスの一部として提供されます。ただし、これがネットワーク構成をカバーしていない可能性もあります。ネットワークへのゲスト・アクセスを可能にするにはhostdevと直接との2種類の方法があるため、具体的なネットワーク構成は複雑です。お使いのオペレーティング・システムでIntel 82599 10Gドライバを使用できる場合、またはすでにそれが組み込まれている場合には、hostdev方式をお薦めします。それより古いオペレーティング・システムを使用している場合や、パッケージ済のオペレーティング・システムをインストールする場合には、直接方式をお薦めします。この2つの方法については、「付録B：ネットワーク接続モードの説明」を参照してください。

注意：このネットワークはブリッジ接続やルーティングをネイティブでサポートしていないため、ゲストの標準KVMブリッジ接続は使用しません。ネットワーク上のゲストには、オラクルの構成管理システムによって指定されるMACアドレスが必要です。ブリッジ接続またはルーティングされたネットワークは独自のMACアドレスを割り当てますが、このネットワークではそれが認識されません。そのため、トラフィックがすべて暗黙的に拒否されてしまいます。KVMゲストをネットワークに参加させるには、特定のアドレスを割り当てる必要があります。このアドレスは、すべてのネットワーク・アクティビティについて、ゲスト・オペレーティング・システムのレベルで使用しなければなりません。ゲスト・オペレーティング・システムについて理解したうえでこの環境にインストールするようにしてください。

ステップ1：ベア・メタル・インスタンスの準備

前提条件のタスクがすべて完了したら、以下の手順でベア・メタル・インスタンスを準備します。

1. システムを更新し、KVMその他のソフトウェアをインストールします。

```
yum update -q -y
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-
7.noarch.rpm
wget https://fedorapeople.org/groups/virt/virtio-win/virtio-win.repo -O
/etc/yum.repos.d/virtio-win.repo
yum -q -y install virtio-win
yum -q -y --nogpgcheck localinstall epel-release-latest-7.noarch.rpm
yum -q -y install qemu-kvm qemu-img virt-manager libvirt libvirt-python
libvirt-client virt-install virt-viewer bridge-utils wget jq
```

注意：ここでは最新のEPELリリースと標準パッチをインストールします。必要に応じて追加の機能もインストールできます。

2. `/etc/default/grub`を開き、`GRUB_CMDLINE_LINUX`エントリの最後に次の行を追加します。

```
intel_iommu=on
```

ファイルは次のようになります。

```
GRUB_CMDLINE_LINUX="crashkernel=auto LANG=en_US.UTF-8 console=tty0
console=ttyS0,9600 rd.luks=0 rd.lvm=0 rd.md=0 rd.dm=0 ip=dhcp
netroot=iscsi:169.254.0.2::::iqn.2015-02.oracle.boot:uefi
iscsi_param=node.session.timeo.replacement_timeout=6000 intel_iommu=on"
```

注意：Oracle Linux 7以外のオペレーティング・システムを使用する場合、grub構成の変更と、その変更のコミット方法は異なります。カーネルに対するブート・オプションとして、SR-IOVを有効にする方法は、オペレーティング・システムのドキュメントを参照してください。

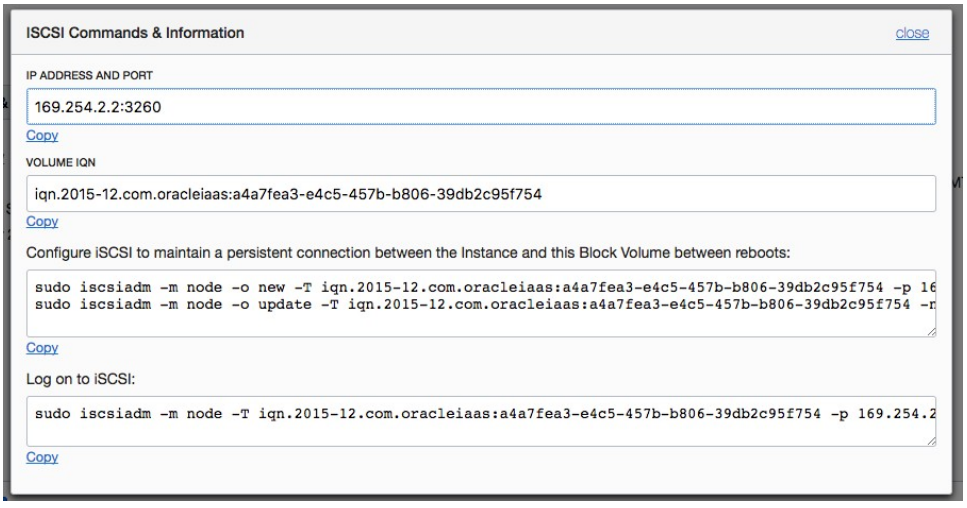
3. `tuned`を有効にし、`virtual-host`のパフォーマンス最適化を設定します。

```
systemctl enable tuned
systemctl start tuned
tuned-adm profile virtual-host
```

4. ブート時に常にこの構成が使用されるように、変更をコミットします。

```
cp /boot/efi/EFI/redhat/grub.cfg /boot/efi/EFI/redhat/grub.cfg.orig
grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

5. ブロック・ストレージ・デバイスがベア・メタル・インスタンスにアタッチされ、使用準備もできていることを確認します。次のいずれかの方法を使用できます。
   - コンソールの「**iSCSIコマンドおよび情報**」ページを開き、ブロック・ストレージ・デバイスに関連するコマンドをコピーします。



   - iscsiattach.shスクリプト（付録C）を使用して、アタッチされ構成されているすべてのブロック・ボリュームを自動的にスキャンします。

6. ブロック・ボリュームの永続マウントを作成します。詳細は、Oracle Cloud Infrastructureのドキュメントで「ボリュームへの接続」を参照してください。

7. 再起動して変更を実装します。

ステップ2：ネットワークの構成

このKVM実装では、SR-IOV仮想機能（FV）と、Oracle Cloud Infrastructureの複数VNIC機能を使用します。ゲスト・オペレーティング・システムからネットワークへの接続をサポートするために、すでに1つ以上のVNICを作成しています。VNICには、Oracle Cloud Infrastructureネットワークで認識されるMACアドレスと、ゲストが必要とするIPアドレス情報との2つがあります。

注意：この構成は再起動後まで永続しません。付録で説明しているスクリプトを応用するか、独自のスクリプトを作成して再起動後も構成が永続するようにするといいでしょう。ただし、ゲストを追加または削除するたびに、最新のネットワーク構成を反映するように構成を変更する必要があります。

ネットワークを構成するには、以下のタスクをrootとして（`sudo`または`sudo su`を利用して）実行します。

1. ハイパーバイザのプライマリ・インタフェースを表すデバイスを特定します。例：

```
URL=http://169.254.169.254/opc/v1/vnics/
baseAddress=`curl -s ${URL} | jq -r '.[0] | .privateIp'`
PHYSDEV=`ip -o -4 addr show | grep ${baseAddress} | awk -F: '{gsub(/^[
\t]|[ \t]$/,"",$2);split($2,out,/[ \t]+/);print out[1]}'`
echo ${PHYSDEV}
```

   注意：この例のURLは、Oracle Cloud Infrastructure Computeメタデータ・サービスの一部です。メタデータ・サービスは、インスタンスの構成に関する情報を提供します。この情報には、URLでGETを実行してアクセスできます。ここでは、このシステムやアタッチされているVNICに関する情報を取得しています。

2. オペレーティング・システムでVFを有効にします。必要なVFの数を`sysfs`に含まれるファイルにエコーします。VNICの現在の最大数に一致するVF数、つまり16が推奨値です。

```
echo "16" > /sys/class/net/${PHYSDEV}/device/sriov_numvfs
```

   このコマンドが完了するまでには、10～20秒かかります。

3. イーサネット・デバイスのブリッジ接続モードを`vepa`に設定します。Intelのイーサネット・カードには、メインのデバイスと全VFとの間を自動的にブリッジする機能があります。

```
bridge link set dev ${PHYSDEV} hwmode vepa
```

   注意：構成する各ゲスト・オペレーティング・システムで使用する追加のインタフェースを構成するには、以下の手順を繰り返します。

4. このゲストに使用するVNICを選択し、VNICに関する次の情報を探します。
   - IPアドレス（自身で選択したもの）
   - MACアドレス
   - VLANタグ

5. 使用するVFを特定します。既存の構成を追加する場合、すでに使用している既存のVFを*上書きしない*ようにしてください。VFのMACアドレスは通常、00:00:00:00:00:00です。

```
ip link show ${PHYSDEV} | grep -v {PHYSDEV} | grep -i vf
```

   hostdevモードを使用してゲストを外部に接続している場合は、使用するVFとポート、スロット、機能番号を指定する以上のネットワーク・デバイス情報は不要です。直接モードでインタフェースを構成する必要がある場合は、次の手順に従う必要がありますが、必要な情報をVFから取得することもできます。

6. VFと、対応するネットワーク・デバイスの両方を構成します。

```
ip link set ${PHYSDEV} vf <VF_number> mac <MAC_address> spoofchk off
```

   ゲストをサポートするために作成したVNICに対応するMACアドレス（`<MAC_address>`）を使用します。大文字小文字は問いません。

   次の手順で、VF情報を検索します。

   A. プライマリ・ネットワーク・デバイスのPCIアドレスを検索します。

```
ethtool -i ${PHYSDEV} | grep bus-info
```

      例：

```
ethtool -i ${PHYSDEV} | grep bus
bus-info: 0000:13:00.0
```

      `bus-info`の行が、プライマリ・ネットワーク・デバイスのPCIアドレスを指定しています。これを使用すると、このデバイスに関連付けられているすべてのVFのアドレスを特定できます。

   B. ベース・デバイスに関連付けられているVFを検索します。

```
lspci -nn | grep -i virtual
```

例：

```
lspci -nn | grep -i virtual
13:10.0 Ethernet controller [0200]: Intel Corporation 82599 Ethernet
Controller Virtual Function [8086:10ed] (rev 01)
13:10.2 Ethernet controller [0200]: Intel Corporation 82599 Ethernet
Controller Virtual Function [8086:10ed] (rev 01)
13:10.4 Ethernet controller [0200]: Intel Corporation 82599 Ethernet
Controller Virtual Function [8086:10ed] (rev 01)
13:10.6 Ethernet controller [0200]: Intel Corporation 82599 Ethernet
Controller Virtual Function [8086:10ed] (rev 01)
13:11.0 Ethernet controller [0200]: Intel Corporation 82599 Ethernet
Controller Virtual Function [8086:10ed] (rev 01)
13:11.2 Ethernet controller [0200]: Intel Corporation 82599 Ethernet
Controller Virtual Function [8086:10ed] (rev 01)
```

PCIアドレスは`port_number:slot_number.function_number`です。たとえば、最初のエントリはアドレスが13:10.0です。

- ポート番号：13
- スロット番号：10
- 機能番号：0

注意：デバイスのアドレス指定では、偶数で終わるベース・デバイスに関連付けられているVFには、偶数のPCIアドレスが割り当てられ、奇数で終わるベース・デバイスに関連付けられているVFには、奇数のPCIアドレスが割り当てられます。上の例ではPCIアドレスが13:00.0、つまり偶数です。

C. 16進のベース・アドレスを10進法に変換します。たとえば、13.10.0は19.16.0になります。

D. ベース・アドレスを、VFに関連付けられているネットワーク・デバイス・ファイルに対するIntelの命名標準でフォーマットします。

```
enp<port_number>s<slot_number>[f<function_number>]
```

例：

```
enp19s16f0
```

注意：デバイスの命名標準では、VFの機能番号が0の場合、`f`エントリが切り捨てられて使用されません。そのため、この例では実際のデバイス・ファイルは`enp19s16`です。

E. VFデバイスをdown、up、down、upに設定して、適切なMACアドレスでデバイスを構成します。

```
ip link set <VF_device_name> down
ip link set <VF_device_name > up
ip link set <VF_device_name > down
ip link set <VF_device_name > up
```

これで、VFネットワーク・デバイスのMACアドレスが、VF自体に設定したアドレスと一致し、どちらも選択したVNICに一致するはずです。

7. VLANタグを使用するインタフェースを作成します。先ほど検索したデバイス・ファイルが使用できますが、正しいVLAN上にありません。VF上で直接VLANを設定することも可能ですが、VLANをクリアまたは変更するために、ベア・メタル・インスタンス全体を再起動してネットワークを完全にリセットしなければならなくなります。これを避けるには、VNIC VLANに固有のリンク付きインタフェースをVFネットワーク・デバイスに作成します。新しいインタフェースを作成するだけです。

```
ip link add link <VF_network_device> name vlan<VLAN_tag> type vlan id <VLAN_tag>
ip link set vlan<VLAN_tag> up
```

VLAN（`<VLAN_tag>`）デバイスは、ゲストのインストール中に使用されるデバイスです。メモを取っておいてください。

注意：この手順はサンプル・スクリプトに統合されており、使用するVNICを選択して、指定された情報に基づいて構成を実行することができます。結果はインベントリ・ファイルに格納され、起動時にそれを使用するとネットワーク・スタック全体を正しく起動することができます。ただし、提供しているすべてのスクリプトと同様、このスクリプトが特定のニーズに適するかどうかは保証できません。

ステップ3：KVMゲストのインストール

インスタンスの作成に使用する方法は、インスタンスをどのように作成したいかによって決まります。

- まったく新しいオペレーティング・システムで新しいゲストをインストールし、元のISOからコピーする
- 事前構成済のオペレーティング・システムがあらかじめインストールされた状態で、既存のイメージをインポートする

インストール用ISOからゲストOSをインストール

この方法では、インストール用ISOから新しいゲストを作成します。この手順を実行する場合には、使用するドライバがすべてそろっており、VNCクライアントがゲストのインストール時にゲストに接続できるようになっている必要があります。この手順では、このゲストによって使用されるネットワーク・インタフェースをすでに構成しており、任意のマウント・ポイントにブロック・デバイスをマウントしている、そして再起動時にはそのブロック・デバイスが再マウントされるように、`/etc/fstab`ファイルに正しいエントリがあると想定しています。

次のコマンドをrootとして実行します。

1. SSHを使用してベア・メタル・インスタンスに接続し、使用するVNC用にSSHトンネルを開いて、ゲストのコンソールに接続します。これは、前提条件で指定したポート（59XX）です。Mac/Linuxでポート・トンネルを開く方法を次の例に示します。Windowsの場合、ローカル・ポート・トンネルの構成方法はSSHクライアント・ソフトウェアで確認してください。

```
ssh -L <VNC_port>:localhost:<VNC_port> opc@<public_IP_of_BM_instance>
```

2. オペレーティング・システムのインストール用ISOとドライバのISOを、ベア・メタル・インスタンスにコピーします。コピー先は、ゲストを保持するブロック・デバイスか、明示的にISOイメージを保持するように作成した別個のディレクトリ構造です。ここでは、ISOのすべてに対して`/usr/local/share/images`を作成しました。

3. ゲストのブート・ドライブとなる仮想ディスクを作成します。

```
qemu-img create -f raw <path_to_block_mount_point_with_file_name>.img
<size of virtual disk in GB>G
```

   最後にある`G`は、測定単位としてGBを使用するというqemu-imgへの指定です。詳細は、qemu-imgのmanページを参照してください。

4. ゲストを含むドメインを作成します。ドメインは、KVM内でゲストを保持するコンテナです。ドメインを作成すると、仮想ハードウェアについてゲストのパラメータがどうなっているかをKVMに指定することになります。自分にとって意味のあるドメイン名を選択します。ゲストのホスト名に一致する必要はありません。

```
virt-install --arch=x86_64 --name=<domain_name> --ram=<RAM_in_MB> --cpu
Haswell-noTSX \
--vcpus <number_of_CPUs> --hvm --video qxl --nonetwork --os-type
<linux_or_windows> \
--noautoconsole --boot cdrom,hd \
--disk <ISO_location>,device=cdrom,bus=ide \
--disk <ISO_location #1>,device=cdrom,bus=ide \
--disk <virtual_disk_path>,format=raw,bus=virtio \
--graphics vnc,port=<VNC_port>,listen=0.0.0.0,password=<password>
```

   パラメータに関する注意事項：

   - 最初のISO_locationタグは、ブートISOのパスを含む必要があります。

- 2番目のISO_locationタグは、初期オペレーティング・システム・インストールの一部としてインストールする必要があるドライバのパスを含む必要があります。Windowsの場合、これは`/usr/share/virtio-win/virtio-win.iso`とする必要があります。この特定のISOには、ストレージ、メモリー、ネットワーク・デバイスに必要なドライバがすべて含まれます。
- 追加のドライバが必要な場合は、3行目のISOを追加し、次のCDROMイメージをマウントできます。これらのデバイスは、インストール後に削除できます。このようなデバイスをドメインから削除する手順については、QEMUとlibvirtのドキュメントを参照してください。
- `--nonetwork`を指定していることに注意してください。ネットワークの実装方法はコマンドラインでは指定できず、ドメインの作成後にデバイスを追加する必要があります。
- VNCポートは、前提条件で指定したポートです。コンソールにはパスワードを設定することもお薦めします。
- オペレーティング・システムに`virtio`ストレージ・バスのドライバがない場合は、この構成の行を適切なバス・タイプに置き換えます。使用できるバス・タイプは多数あり、virt-installのmanページに記載されています。

5. 次の「ステップ4：ドメインへのネットワーク・デバイスのアタッチ」までスキップします。

既存のイメージからゲストOSをインポート

事前構成済のイメージをインポートする際には、オペレーティング・システムをどのようにインストールしたかと、KVM環境の要件に応じてイメージに対する変更が可能かどうかによって、制限事項があります。この手順では、VNICをすでに作成しており、どのネットワーク・インタフェース方式およびタイプを使用するか理解していて、成功に必要な前提条件（ブロック・デバイスのマウントなど）がすべて完了していると想定します。また、イメージ・ファイルがすべて正しい形式（KVM RAWまたはqcow2）であること、あるいはファイルをベア・メタル・インスタンスにコピーした後で変換プロセスが実行されることも前提です。

次のコマンドをrootとして実行します。

1. SSHを使用してベア・メタル・インスタンスに接続し、使用するVNC用にSSHトンネルを開いて、ゲストのコンソールに接続します。これは、前提条件で指定したポート（59XX）です。Mac/Linuxでポート・トンネルを開く方法を次の例に示します。Windowsの場合、ローカル・ポート・トンネルの構成方法はSSHクライアント・ソフトウェアで確認してください。

```
ssh -L <VNC_port>:localhost:<VNC_port> opc@<public_IP_of_BM_instance>
```

2. 仮想ディスクのイメージ・ファイルをブロック・ストレージ・デバイスにコピーします。適切ではない可能性があるため、ルート・デバイスには*コピーしない*でください。

3. ゲストを含むドメインを作成します。

```
virt-install --arch=x86_64 --name=<domain_name> --ram=<RAM_in_MB> --cpu
Haswell-noTSX \
--vcpus <number_of_CPUs> --hvm --video qxl --nonetwork --os-type
<linux_or_windows> --noautoconsole \
--disk <virtual_disk_path>,format=raw,bus=virtio \
--graphics vnc,port=<VNC_port>,listen=0.0.0.0,password=<password> --import
```

パラメータに関する注意事項：

- なんらかの理由でISOをイメージにアタッチする必要がある場合は、次のような行を構成に追加します。

  ```
  --disk <ISO_location>,device=cdrom,bus=ide \
  ```

- イメージに複数の仮想ディスクがある場合は、構成コマンドで複数の`virtual disk path`行を指定して順番にリストします。
- `--nonetwork`を指定していることに注意してください。ネットワークの実装方法はコマンドラインでは指定できず、ドメインの作成後にデバイスを追加する必要があります。
- VNCポートは、前提条件で指定したポートです。コンソールにはパスワードを設定することもお薦めします。
- オペレーティング・システムに`virtio`ストレージ・バスのドライバがない場合は、この構成の行を適切なバス・タイプに置き換えます。使用できるバス・タイプは多数あり、virt-installのmanページに記載されています。

4. 次の「ステップ4：ドメインへのネットワーク・デバイスのアタッチ」までスキップします。

ステップ4：ドメインへのネットワーク・デバイスのアタッチ

ドメインを作成したら、結果のネットワーク・デバイスをアタッチします。ネットワーク・タイプに応じて適切な方法に従ってください。

hostdevネットワーク

1. 必要なVNIC情報を収集します。
   - MACアドレス
   - VLANタグ
   - VFのPCIポート番号、スロット番号、および機能番号（ここまでの手順で収集済）

2. 以下のテンプレートを使用して、ネットワーク・インタフェースの詳細を指定したファイルを作成します。各プレースホルダを該当する情報に置き換え、XMLファイル（attach.xmlなど）として保存します。

```
<interface type='hostdev' managed='yes'>
  <source>
    <address type='pci' domain='0x0000' bus='[port_number_in_hex]'
slot='[slot_number_in_hex]' function='[VF_number_in_hex]'/>
  </source>
  <vlan>
    <tag id='[VLAN_tag]'/>
  </vlan>
  <mac address='[VNIC_MAC_address]'/>
</interface>
```

3. ドメインにネットワーク・デバイスをアタッチします。

```
virsh attach-device <your_domain_name> ./attach.xml --config
```

ファイル名をattach.xml以外にした場合は、それに合わせてファイル名を変更してください。

4. ドメインを強制的に再起動します。次のコマンドは、ドメインを強制的に停止（destroyの部分）してから再起動します。これを実行する必要があるのは、インストール・プロセスの一環としてネットワーク・デバイスを選択するためです。

```
virsh destroy <your_domain_name>
virsh start <your_domain_name>
```

5. ドメインが再起動したら、ゲストに設定するオペレーティング・システムに対応するインストール・プロセスを続行します。デスクトップまたはラップトップにインストールしたVNCソフトウェアを使用して、localhost:59XX（XXは前のステップで選択したポート番号）でゲスト・コンソールに接続し、オペレーティング・システム構成タスクを実行します。

hostdevネットワーク

1. 必要なVNIC情報を収集します。
   - MACアドレス
   - 前の手順で作成したVLANネットワーク・デバイスの名前
   - エミュレートするネットワーク・カード・モデル（e1000、virtioなど）

2. 以下のテンプレートを使用して、ネットワーク情報の詳細を指定したファイルを作成します。各プレースホルダを該当する情報に置き換え、XMLファイル（`attach.xml`など）として保存します。

```
<interface type='direct'>
  <mac address='[VNIC_MAC_address]'/>
  <source dev='[VLAN_network_device_name]' mode='passthrough'/>
  <model type='[NIC_model_for_guest]'/>
</interface>
```

3. ドメインにネットワーク・デバイスをアタッチします。

```
virsh attach-device <your_domain_name> ./attach.xml --config
```

ファイル名を`attach.xml`以外にした場合は、それに合わせてファイル名を変更してください。

4. ドメインを強制的に再起動します。次のコマンドは、ドメインを強制的に停止（`destroy`の部分）してから再起動します。これを実行する必要があるのは、インストール・プロセスの一環としてネットワーク・デバイスを選択するためです。

```
virsh destroy <your_domain_name>
virsh start <your_domain_name>
```

5. ドメインが再起動したら、ゲストに設定するオペレーティング・システムに対応するインストール・プロセスを続行します。デスクトップまたはラップトップにインストールしたVNCソフトウェアを使用して、`localhost:59XX`（`XX`は前のステップで選択したポート番号）でゲスト・コンソールに接続し、オペレーティング・システム構成タスクを実行します。

ゲスト・ネットワーク構成

ゲスト用に作成するVNICに関連付けられているネットワーク情報は、この時点ではDHCPを通じてゲストに提供されません。そのため、ゲストのIP情報は、インストールする特定のオペレーティング・システムまたはイメージに対応する個々の手順に従って、手動で構成する必要があります。この情報は、ゲストがVNIC経由でアタッチされている特定のサブネットのIPアドレス、ネットマスク、およびデフォルト・ルートなどです。

イメージをインストールする場合、ほぼすべてのイメージでIP情報を静的に設定できます。この場合、IPアドレス（VNIC構成から）とデフォルト・ゲートウェイの*両方*を永続構成として静的に設定します。

まとめ

おめでとうございます。ベア・メタル・インスタンスで稼働し、Oracle Cloud Infrastructureの残りの部分と対話できるKVMに、ゲストを正常にインストールできました。

付録A：インストールのトラブルシューティング

- ゲスト・オペレーティング・システムに作成した仮想ドライブが表示されない場合、たいていはオペレーティング・システムのインストール・プロセスの一環として`virtio`ドライバの追加が必要です（必要なのは通常、Windowsの場合です）。適切なドライバを使用してISOをアタッチ/マウントして再試行してください。その方法については、KVMとlibvirtのページを参照してください。
- インストール中にネットワーク・インタフェースが使用できなくなるのも、インストール中に適切なドライバがない場合の兆候です。同じ手順で、仮想ドライブ用に指示されたドライブをインストールする必要があります。ただし、オペレーティング・システムのインストール中にネットワークが必要ない場合、ネットワーク・インタフェースは後からいつでも構成できます。
- VNICの作成中に割り当てた、または作成したIPアドレスは、DHCPによって提供されます。ベア・メタル・インスタンスがインスタンス化されるかぎり、そのアドレスは常にそのVNICに割り当てられます。したがって常に、直前に作成したゲストに割り当てられます。必然的な理由がないかぎり、ゲスト内からIPアドレス情報を静的に割り当てる必要はありません。ただし、オペレーティング・システム内では静的なデフォルト・ルートの定義が必要です。すべてのIPアドレス情報を入力する必要がある場合、DHCPは無視され、IP情報はゲスト・オペレーティング・システム内で適用されます。

付録B：ネットワーク接続モードの説明

hostdev

パフォーマンス的にもゲスト分離の面でも、hostdev方式をお薦めします。ゲストから、ハイパーバイザ上でSR-IOVを構成する一部として作成されるPCIデバイスに直接アクセスできるからです。PCIデバイスは、仮想機能（VF）と呼ばれ、ハイパーバイザ（ベア・メタル・インスタンス）のハードウェアに対する実際のインタフェースになります。そのため、ゲストでは最大のスループットと分離度が確保されます。

- スループットが最大になるのは、ゲストとネットワークの間にオペレーティング・システムが介在しないためです。
- 分離度が最大になるのは、ハイパーバイザのオペレーティング・システムが、ハードウェア・インタフェースを提供する以上には関与しないためです。

hostdev方式の欠点は、別のデバイス・タイプをエミュレートできないことにあります。そのため、ゲスト・オペレーティング・システムでは、ハイパーバイザによって提供されるハードウェアのタイプに一致するドライバが必要になります。Oracleは、ベア・メタル・インスタンスにIntel 82599 10Gイーサネット・インタフェースを使用します。各ゲスト・オペレーティング・システムには、このデバイスをサポートするデバイス・ドライバをインストールするか、ゲストのインストール後またはインポートの完了後になんらかの方法でそのドライバをインストールする必要があります。テストされたレガシー・オペレーティング・システムの多くに使用されたドライバ・セットは、Intel® Ethernet Adapter Connections CDにあります。完全なドライバ・セットをダウンロードし、そこからISOを作成することをお薦めします。ネットワーク接続の方式を選択する前に、ドキュメントでドライバ・セットについて参照し、オペレーティング・システムでこの製品のドライバが利用できることを確認してください。

直接

直接方式は、ハイパーバイザ構成のネットワーク・インタフェースを利用してゲスト・オペレーティング・システムに接続します。ただし、ハイパーバイザによって提供されるネットワーク構成は最小限です。DHCPおよび関連の高度なネットワーク管理機能はゲスト・オペレーティング・システムが引き続きすべて発行し、ハイパーバイザは単にゲストが動作するインタフェースを提供するだけです。

直接方式では、最新およびレガシーのオペレーティング・システムでよくある一般的なネットワーク・インタフェース・タイプをKVMでネイティブにエミュレートできます。e1000（Intel FastEthernetドライバ）およびvirtio（KVMネイティブ）デバイス・タイプのエミュレーションは動作することが確認されていますが、virtioドライバでは引き続き、Windowsオペレーティング・システムにドライバをインジェクトする必要があります。これは、構成が通常は静的であり、特定のハードウェア・タイプを必要とするため、パッケージ済の仮想マシンに便利です。VM用で最も一般的なタイプは、e1000エミュレーションです。直接方式のネットワーク・デバイスについて説明するときには、これを使用します。

付録C：スクリプト・ライブラリ

以下のスクリプトは、ここまでに説明したタスクの一部を自動化するものです。どのスクリプトも、オラクルの環境で動作が確認されていますが、どの環境でも同じように動作するかどうかは保証できません。

一般的なVFネットワーク構成機能：common.sh

```
#!/bin/bash
#
# common.sh - common functions
#
# Copyright Oracle, Inc. All rights reserved.
# This is provided "as is" without warrantee, either explicit or implied



function GetPhysDev {
```

```
    URL="http://169.254.169.254/opc/v1/vnics/"
    baseAddress=`curl -s ${URL} | jq -r '.[0] | .privateIp'`
    physdev=`ip -o -4 addr show | grep ${baseAddress} | awk -F: '{gsub(/^[ \t]|[
\t]$/,"",$2);split($2,out,/[ \t]+/);print out[1]}'`
    echo ${physdev}
}

function GetVfInfo {
    physdev=$1
    vfNumber=$2

    declare -a virtArray

    function=`echo "(${vfNumber} % 4) * 2" | bc`
    (( cycle = vfNumber / 4 ))
    basePci=`ethtool -i ${physdev} | grep "bus-info" | awk -F: '{print $3":"$4}'`
    portPci=`echo ${basePci} | awk -F: '{print ":"$1":"}'`

    read -ra virtArray <<< `lspci -nn -s ${portPci} | grep -v ${basePci} | awk '{print
$1}' | grep "\.${function}"`

    vfPci=${virtArray[${cycle}]}

    hexPort=`echo ${vfPci} | awk -F: '{print $1}'`
    hexSlot=`echo ${vfPci} | awk -F: '{print $2}' | sed 's/\..*$//'`
    port=$((0x`echo ${hexPort}`))
    slot=$((0x`echo ${hexSlot}`))

    if [ ${function} -eq 0 ]
    then
        postfix=""
    else
        postfix="f"${function}
    fi

    device="enp"${port}"s"${slot}${postfix}
    echo ${hexPort}","${hexSlot}","${function}","${device}
}

function ConfigVlan {
    physdev=$1
    vlanId=$2
    ip link add link ${physdev} name vlan${vlanId} type vlan id ${vlanId}
    ip link set vlan${vlanId} up
```

```
}

function ConfigVf {
    physdev=$1
    vfNumber=$2
    macAddr=$3
    ip link set ${physdev} vf ${vfNumber} mac ${macAddr} spoofchk off
}

function ConfigDev {
    physdev=$1
    vfNumber=$2
    macAddr=$3

    declare -a vfDev
    IFS="," read -ra vfDev <<< $(GetVfInfo ${physdev} ${vfNumber})
    ip link set ${vfDev[3]} down
    ConfigVf ${physdev} ${vfNumber} ${macAddr}
    ip link set ${vfDev[3]} up
    ip link set ${vfDev[3]} down
    sleep 5
    ip link set ${vfDev[3]} up
    sleep 10
    echo ${vfDev[3]}
}
```

iSCSIボリュームを自動的にアタッチする：iscsiattach.sh

```
#!/bin/bash
# iscsiattach.sh - Scan and automatically attach new iSCSI targets
#
# Author: Steven B. Nelson, Sr. Solutions Architect
#       Oracle Cloud Infrastructure
#
# 20 April 2017
# Copyright Oracle, Inc. All rights reserved.

BASEADDR="169.254.2.2"

# Set a base address incrementor so we can loop through all the
# addresses.
addrCount=0

while [ ${addrCount} -le 32 ]
```

```
do

    CURRADDR=`echo ${BASEADDR} | awk -F\. '{last=$4+'${addrCount}';print
$1"."$2"."$3"."last}'`

    clear
    echo "Attempting connection to ${CURRADDR}"

    mkfifo discpipe
    # Find all the iSCSI Block Storage volumes attached to the instance but
    # not configured for use on the instance. Basically, get a list of the
    # volumes that the instance can see, the loop through the ones it has,
    # and add volumes not already configured on the instance.
    #
    # First get the list of volumes visible (attached) to the instance

    iscsiadm -m discovery -t st -p ${CURRADDR}:3260 | grep -v uefi | awk '{print $2}' >
discpipe 2> /dev/null &

    # If the result is non-zero, that generally means that there are no targets available
or
    # that the portal is reachable but not active. We make no distinction between the two
    # and simply skip ahead.
    result=$?
    if [ ${result} -ne 0 ]
    then
        (( addrCount = addrCount + 1 ))
        continue
    fi

    # Loop through the list (via the named FIFO pipe below)
    while read target
    do
        mkfifo sesspipe
        # Get the list of the currently attached Block Storage volumes
        iscsiadm -m session -P 0 | grep -v uefi | awk '{print $4}' > sesspipe 2> /dev/null
&

        # Set a flag, and loop through the sessions (attached, but not configured)
        # and see if the volumes match. If so, skip to the next until we get
        # through the list. Session list is via the pipe.
        found="false"
        while read session
        do
```

```
        if [ ${target} = ${session} ]
        then
            found="true"
            break
        fi
        done < sesspipe

        # If the volume is not found, configure it. Get the resulting device file.
        if [ ${found} = "false" ]
        then
        iscsiadm -m node -o new -T ${target} -p ${CURRADDR}:3260
        iscsiadm -m node -o update -T ${target} -n node.startup -v automatic
        iscsiadm -m node -T ${target} -p ${CURRADDR}:3260 -l
        sleep 10
        fi
    done < discpipe

    (( addrCount = addrCount + 1 ))
    find . -maxdepth 1 -type p -exec rm {} \;
done
echo "Scan Complete."
```

ゲストに使用するVNICを選択して構成する：selectvnic.sh

```
#!/bin/bash
# selectvnic.sh - Select and configure the vNIC to use for the KVM configuration
#
# Steven B. Nelson, Sr. Solutions Architect
# Oracle, Inc.
#
../common.sh

URL="http://169.254.169.254/opc/v1/vnics/"
CONFIGFILE=./kvmnet.conf
MAXVFS=16

PHYSDEV=$(GetPhysDev)

declare -a privateIp
declare -a macAddr
declare -a vlanTag
declare -a ipLink

numvfs=`cat /sys/class/net/${PHYSDEV}/device/sriov_numvfs`
```

```
if [ ${numvfs} -eq 0 ]
then
    echo "No VFs appear to have been configured. Setting up ${MAXVFS} VFs now. Stand
by."
    echo "${MAXVFS}" > /sys/class/net/${PHYSDEV}/device/sriov_numvfs
    sleep 30
    echo "Setting the bridging mode."
    bridge link set dev ${PHYSDEV} hwmode vepa
fi

read -ra ipLink <<< `ip link show ${PHYSDEV} | grep vf | awk -F, '{print $1}' | sed 's/^
*//g;s/ /,/g'`

echo "Stand by. Gathering link information"
for field in privateIp vlanTag macAddr
do
    read -ra ${field} <<< `curl -s ${URL} | jq -r '.[1:(length)] | .[].'"${field}"''`
done

for link in ${ipLink[@]}
do
    IFS="," read -ra linkInfo <<< ${link}
    count=0
    while [ ${count} -lt ${#macAddr[@]} ]
    do
        uMacvNic=`echo ${macAddr[${count}]} | awk '{print toupper($0)}'`
        uMacLink=`echo ${linkInfo[3]} | awk '{print toupper($0)}'`
        preConfig=`grep ${macAddr[${count}]} ${CONFIGFILE}`
        if [ ${uMacvNic} = ${uMacLink} ] || [ -n "${preConfig}" ]
        then
            echo "IP address ${privateIp[${count}]} appears to be used. Removing from
list"
            unset privateIp[${count}]
            unset macAddr[${count}]
            unset vlanTag[${count}]
            unset ipLink[${count}]
            declare -a temp=( ${privateIp[@]} )
            privateIp=( ${temp[@]} )

            temp=( ${vlanTag[@]} )
            vlanTag=( ${temp[@]} )

            temp=( ${macAddr[@]} )
            macAddr=( ${temp[@]} )
```

```
            temp=( ${ipLink[@]} )
            ipLink=( ${temp[@]} )
        fi
        (( count = count + 1))
    done
done

if [ ${#privateIp[@]} -lt 1 ]
then
    echo "All IP addresses currently assigned are consumed. Please assign a"
    echo "new vNIC to this hypervisor and try again."
    exit 1
fi

if [ ${#ipLink[@]} -lt 1 ]
then
    echo "There are not any Virtual Functions (VFs) available for use"
    echo "Change the VF configuration and try again."
    exit 2
fi

echo "Select the IP address to use for the guest"
select opt in ${privateIp[@]} "quit"
do
    quitIndex=`expr ${#privateIp[@]} + 1`
    if [ ${REPLY} -le ${#privateIp[@]} ]
    then
        break
    elif [ ${REPLY} -eq ${quitIndex} ]
    then
        exit 3
    fi
done
(( index = REPLY - 1 ))

vfNum=`echo ${ipLink[0]} | awk -F, '{print $2}'`
echo ${vfNum}","${macAddr[${index}]}","${vlanTag[${index}]} >> ${CONFIGFILE}
vfDev=$(ConfigDev ${PHYSDEV} ${vfNum} ${macAddr[${index}]})
ConfigVlan ${vfDev} ${vlanTag[${index}]}
```

以前に選択したVFを再構成する：netconfig.sh

```
#!/bin/bash
# netconfig.sh - Configuration of network at boot time for KVM
#
# Steven B. Nelson, Sr. Solutions Architect
# Oracle, Inc.

. ./common.sh

NUMVFS=16
CONFIGFILE=./kvmnet.conf

PHYSDEV=$(GetPhysDev)

currvfs=`cat /sys/class/net/${PHYSDEV}/device/sriov_numvfs`
if [ ${currvfs} -eq 0 ]
then
    echo "Standby - configuring ${NUMVFS} VF devices"
    echo ${NUMVFS} > /sys/class/net/${PHYSDEV}/device/sriov_numvfs
    sleep 30
fi

echo "Setting the bridging mode"
bridge link set dev ${PHYSDEV} hwmode vepa

while IFS="," read -ra netConfig
do
    if grep -q "#" <<< ${netConfig[0]}
    then
        continue
    else
        vfDev=$(ConfigDev ${PHYSDEV} ${netConfig[0]} ${netConfig[1]})
        ConfigVlan ${vfDev} ${netConfig[2]}
    fi
done < ${CONFIGFILE}
```

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax：+1.650.506.7200

マルチVNICを使用したベア・メタル・インスタンスへのKVMのインストールと構成
2018年2月
著者および技術的な連絡先：Steven B. Nelson