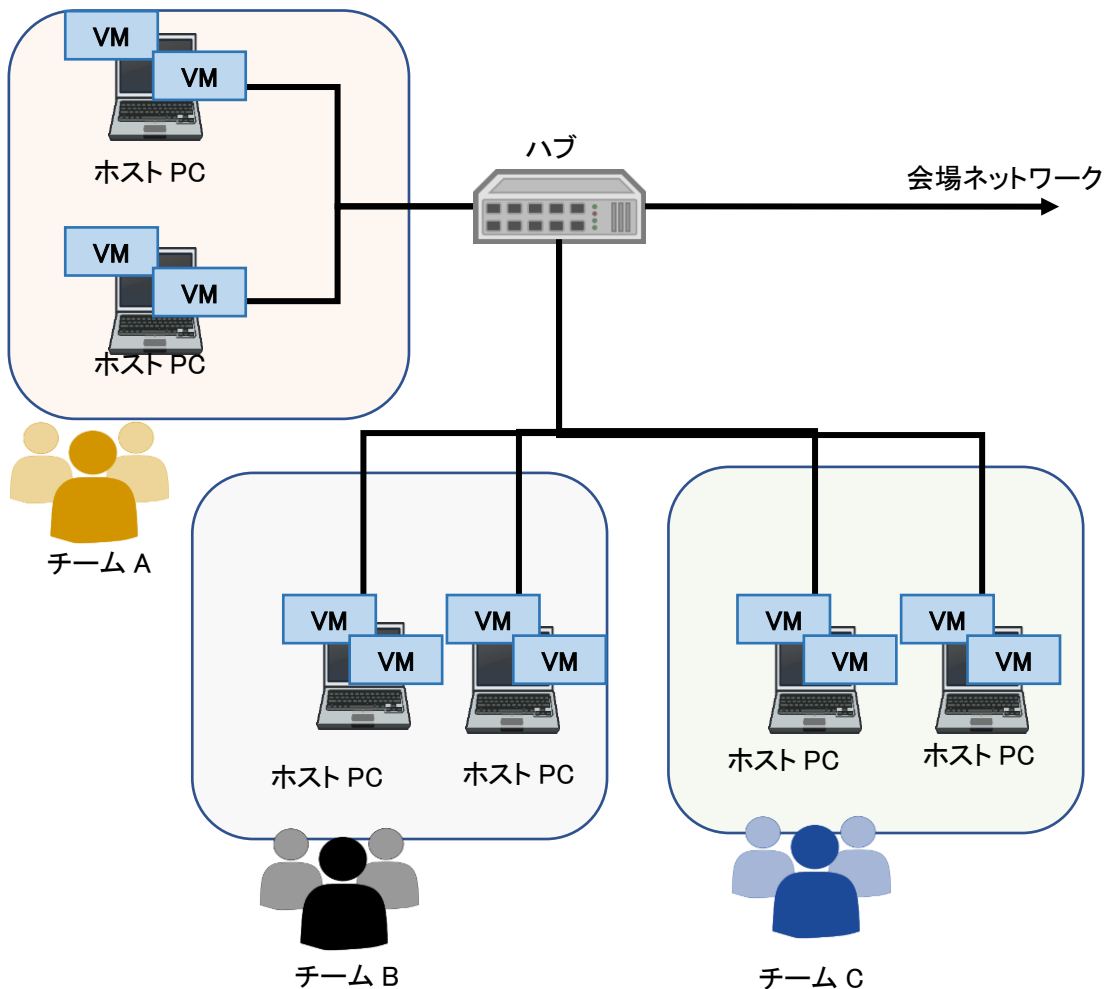


仮想化 演習資料

演習の前に

仮想化では、以下のネットワークポロジで演習を行います。



数人で1つのチームを組み、クライアントPCとサーバを操作します。サーバにはWindows 10上に仮想化アプリケーションを、クライアントにはWindows10を使用します。サーバには仮想マシン(VM)を構築し、そのVMをクライアントPCから接続して演習を行います。

仮想化演習は手順とその結果を確認しながら行うため、別途模範解答はありません。

演習 1 VirtualBox のインストール

① Oracle VM VirtualBox

Oracle VM VirtualBox は、仮想化ソフトウェアパッケージのひとつです。既存の OS（ホスト OS）上にアプリケーションの一つとしてインストールされ、その中に追加の OS（ゲスト OS）を実行させることができます。

サポートされるホスト OS は、Linux、macOS、Windows OS 等であり、ゲスト OS として FreeBSD、Linux、OpenBSD、OS/2 Warp、Windows、Mac OS X Server、Solaris 等、x86/x64 アーキテクチャの OS であれば基本的に動作します。

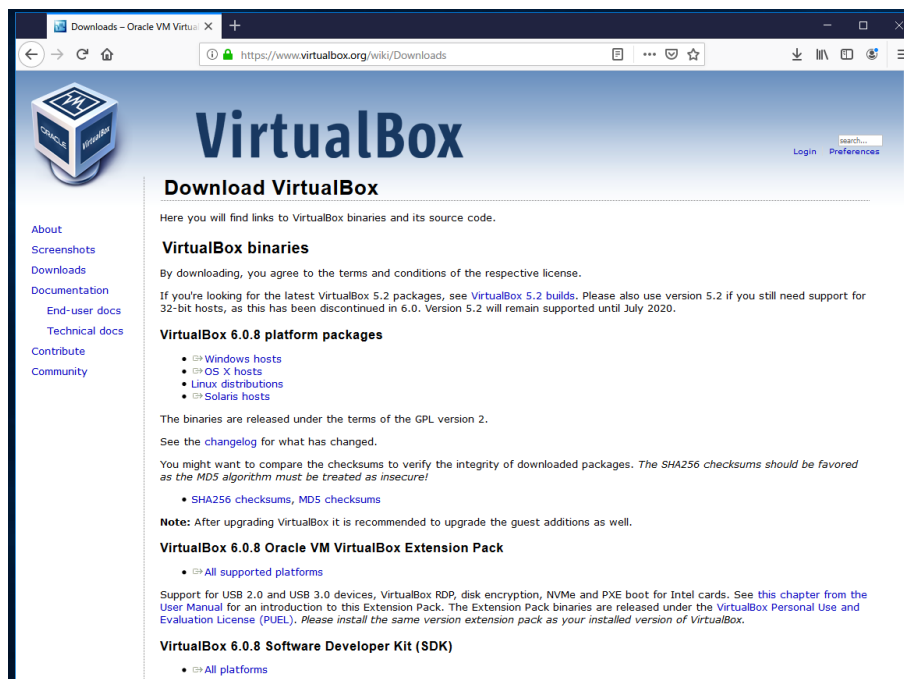
流れとして、ホスト OS の Windows に VirtualBox をインストールし、そこにゲスト OS をインストールしていきます。その後、別の端末からもネットワーク経由で操作できることを体験してもらいます。

② VirtualBox インストール

Windows 10 がインストールされたサーバ機に対して、VirtualBox のインストールイメージをダウンロードします。

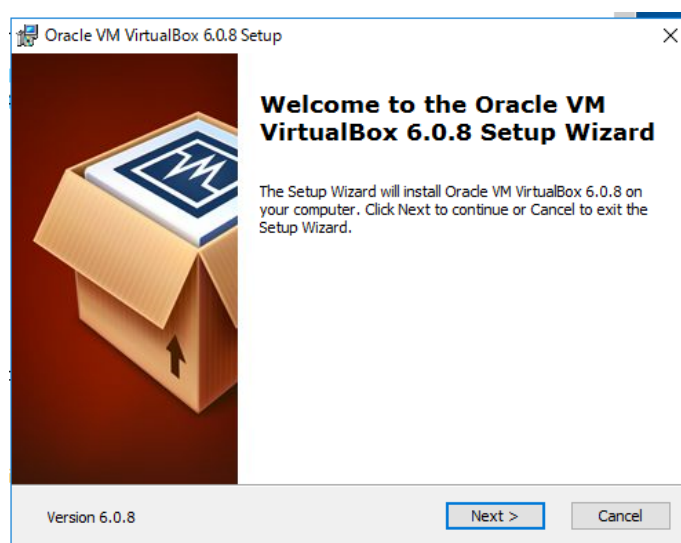
URL: <https://www.virtualbox.org/>

（インターネット検索で virtualbox と検索すれば、該当ページにたどり着くことが可能です）

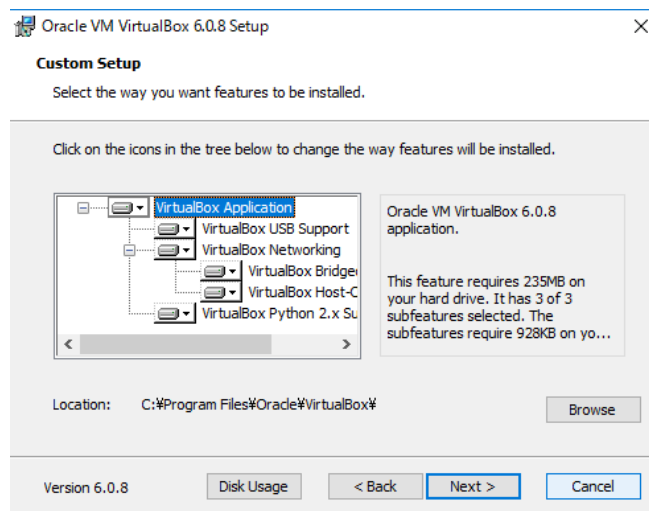


トップページにある、“VirtualBox 6.0.8 platform packages” の欄にある
“Windows hosts” から、Windows 用のインストーラをダウンロードします。
また、後ほど拡張パックも併せてインストールするため、“VirtualBox 6.0.8 Oracle VM
VirtualBox Extension Pack” の欄にある“All supported platforms” も併せてダウンロ
ードしておきます。
※ VirtualBox 6.0.8 は 2019/05/13 にリリースされたものです。

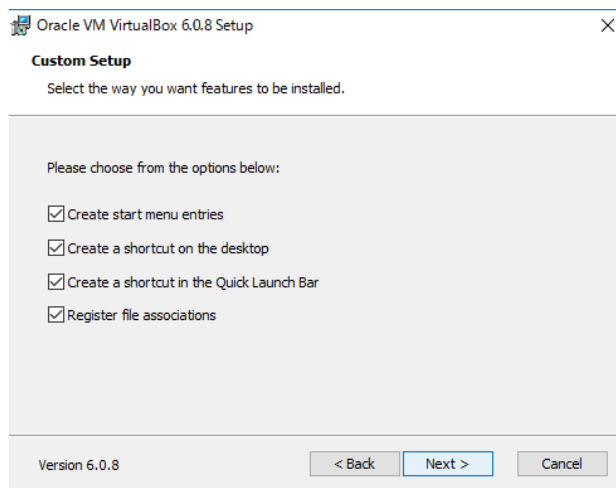
ダウンロードが完了したら、“VirtualBox-6.0.8-130520-Win.exe” を展開し、インスト
ーラを起動します。



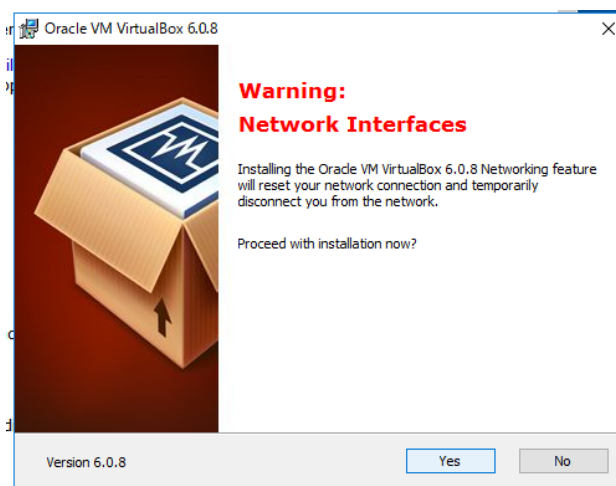
セットアップウィザードに従ってインストールを進めていきます。
特に設定を変更する必要はなく、[Next >] を押して順番に進めていきます。



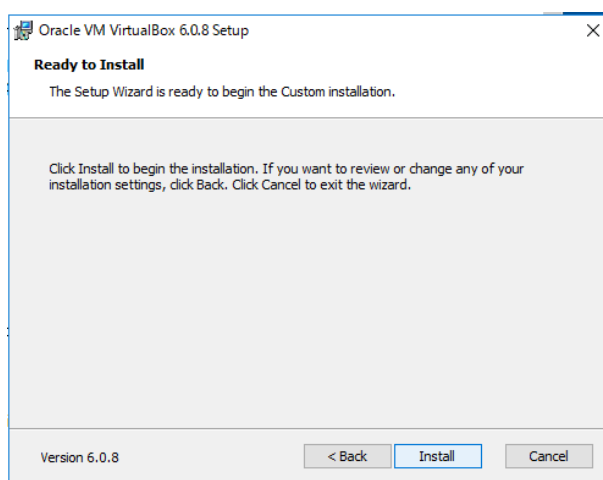
スタートメニューやショートカットをデスクトップに表示する場合は、“Custom Setup”の項目にチェックをいれて、[Next >]に進みます（原則、そのまま進めてもらってOKです）。



“Warning: Network Interfaces”と出ますが、ネットワーク接続がリセットされ、一時的にネットワークが切断されるための警告です。特に問題ありませんので、[Yes]を選択します。



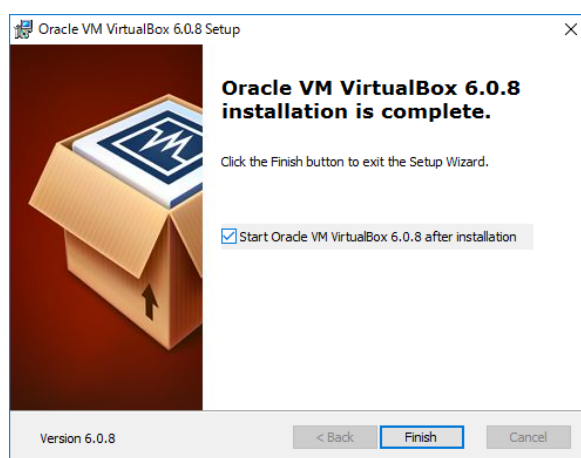
インストールの準備ができると、次の画面になります。[Install]を選んでインストールを開始します。



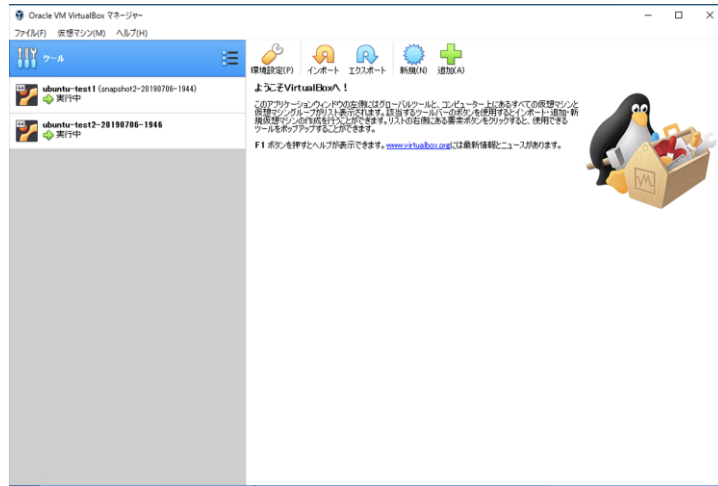
インストールの途中で USB コントローラのインストールが求められることがありますが、続けて[インストール]を選択します。



インストールが完了すると、“Oracle VM VirtualBox 6.0.8 installation is complete.”と表示されます。[Finish]を選択して、インストーラを終了します。



インストールに成功し、VirtualBox を起動すると、下記のような画面が出てきます。
(ペンギンが目印)

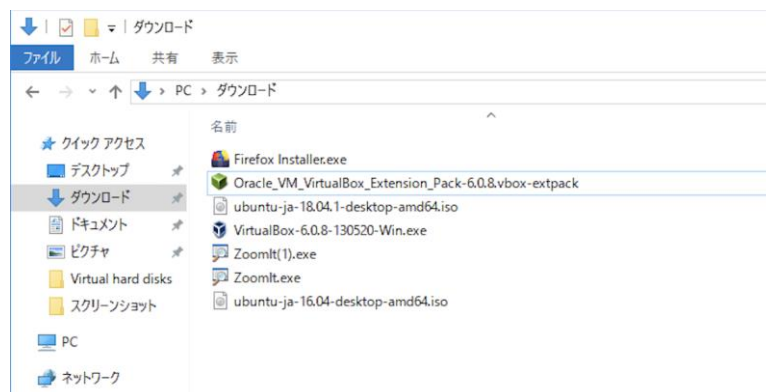


③拡張パックのインストール

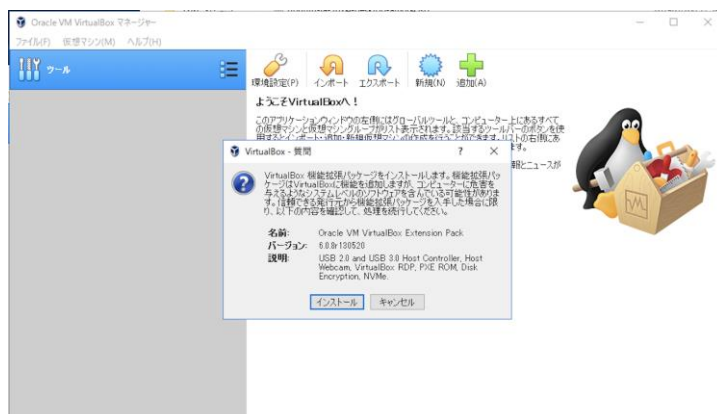
必要最低限の動作を行う場合、拡張パックの導入は必須ではないのですが、利用を便利にする機能が多く備わっていますので、導入しておきましょう。

たとえば、外付けハードディスクの利用やリモートデスクトップによる遠隔制御、ホスト側に接続されている Web カメラの利用、シームレスモード、ゲスト仮想ディスク暗号化のような機能が備わっています。

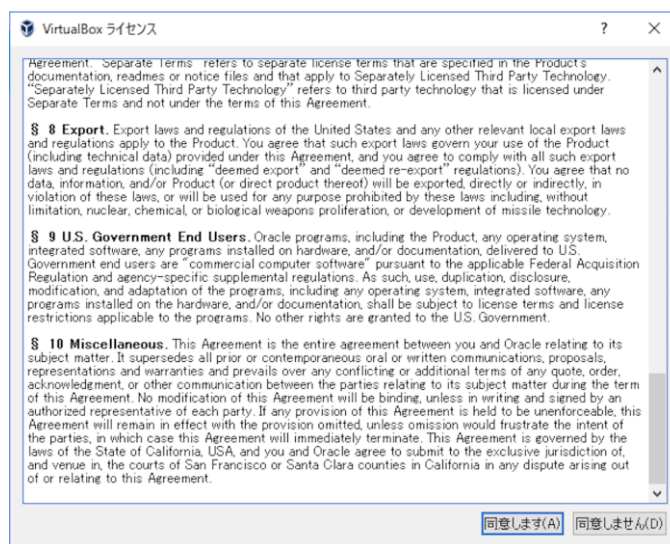
VirtualBox 本体のインストールが完了すると、②でダウンロードした“VirtualBox 6.0.8 Oracle VM VirtualBox Extension Pack”のインストールが可能になっています（ファイルの関連付けが完了しています）。当該フォルダを開き、ダブルクリックして展開します（例ではダウンロードフォルダに入っています）。



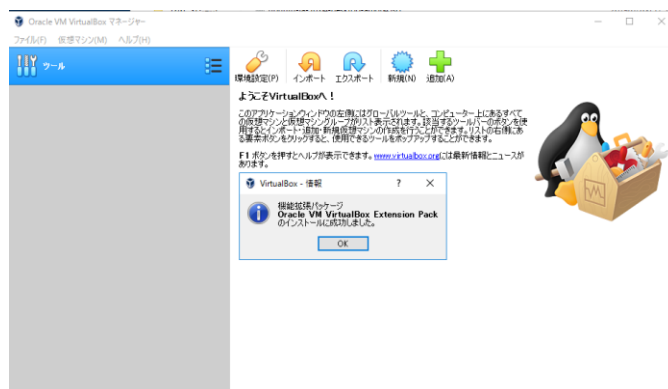
拡張パックのインストールが始まります。内容を確認し、[インストール]を選択します。



ライセンス条項が表示されますので、内容を確認の上、[同意します]を選択します。



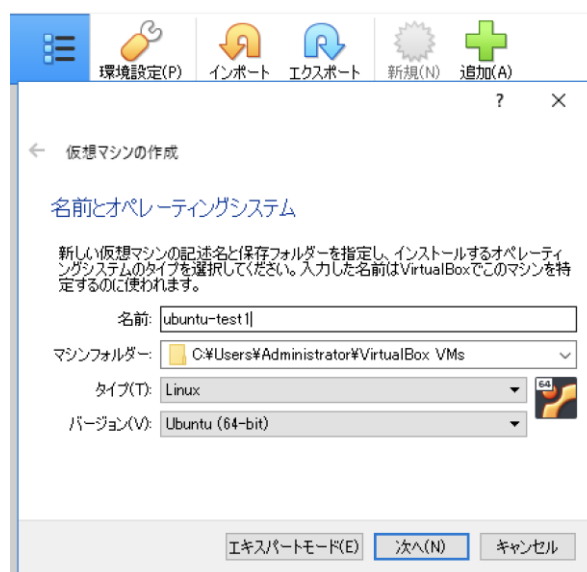
拡張パックのインストールが完了すると、「機能拡張パッケージ Oracle VM VirtualBox Extension Pack のインストールに成功しました。」と表示されます。確認の後、[OK]を選択します。



演習 2 仮想マシンの設定

代表的な Linux ディストリビューションの Ubuntu Desktop による仮想マシンを設定して試します。インストール済のホスト OS に、Ubuntu Desktop 用の仮想マシンの領域を作成します。

VirtualBox メインメニューの「追加」を選択し、仮想マシンの領域を作成します。その後、マシン名等を決定します。

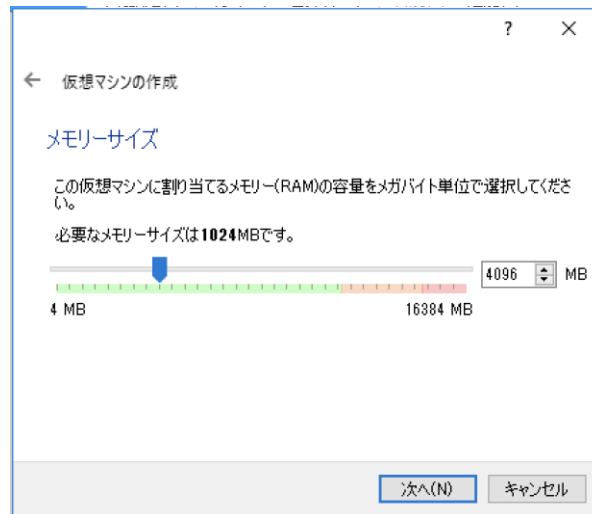


ここでは、以下のように設定します。

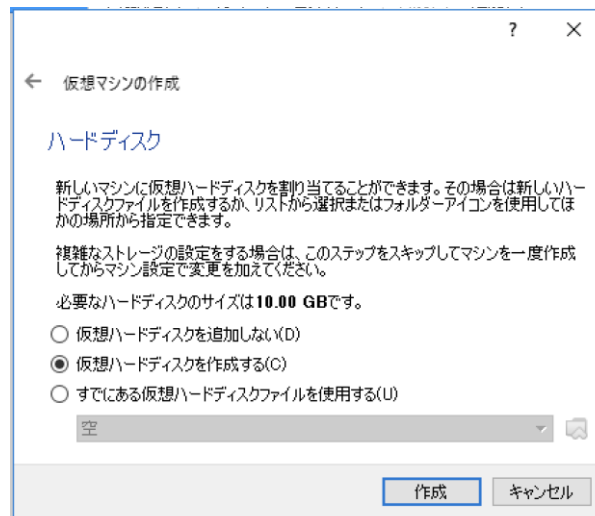
項目	内容
仮想マシンの名前	ubuntu-test1 など (任意)
マシンフォルダー	(任意)
タイプ	Linux
バージョン	Ubuntu (64-bit)

※ バージョンを設定する際に 64bit が選択できない可能性があります。これは、ホスト OS 側が 64bitOS であっても、ホスト端末の BIOS レベルで仮想化オプションを OFF にしている場合が考えられます。64bit が選択できない場合は、ホスト OS を再起動し、BIOS (UEFI) を立ち上げて、拡張メニューから、仮想化オプションを有効にしてください。(主な箇所としては、VT-d (Virtualization Technology) 等の名称が相当します。)

ここでは、メモリーサイズの割り当てを 4096MB (4GB) に設定します。



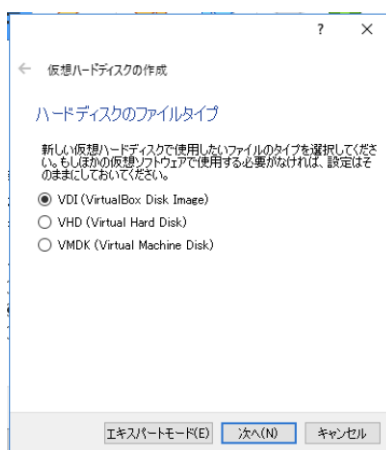
仮想マシンのハードディスクを 10GB に設定します。設定項目を「仮想ハードディスクを作成する」にします。



ハードディスクのタイプを指定します。

- ・ VDI (VirtualBox Disk Image) … VirtualBox 専用のディスクイメージ形式です。
- ・ VHD (Virtual Hard Disk) … Hyper-V (Windows での仮想化技術)用のディスクイメージ形式です。
- ・ VMDK (Virtual Machine Disk) … VMware の vSphere で利用するディスクイメージ形式です。

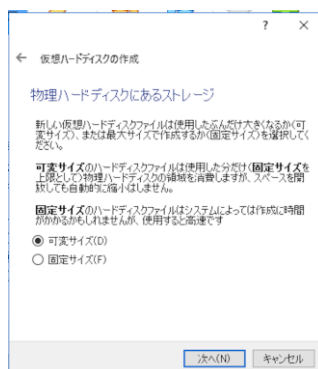
これらはいずれも互換性があります。今回は、“VDI”を選択します。



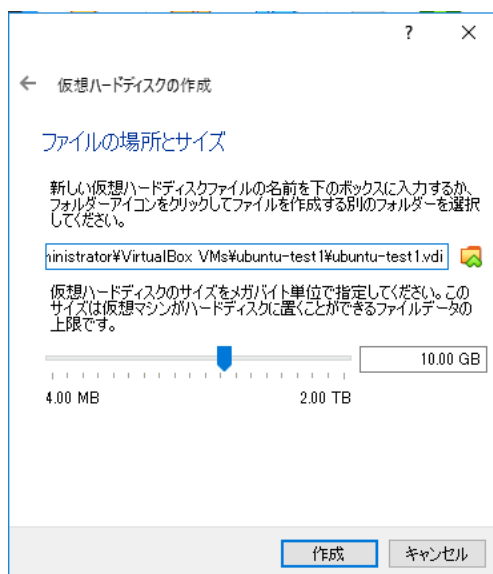
ストレージのタイプを選択します。

- ・ 可変サイズ … ハードディスクファイルは固定サイズを上限として、使用した文だけの領域を消費します。
- ・ 固定サイズ … 必要分を未使用であっても確保しますが、高速にアクセスできます。

ここでは「可変サイズ」を選択し、[次へ]に進みます。



仮想ハードディスクファイルの設置場所を選択します。特に希望がなければ、そのまま [作成] を選択して次に進みます。



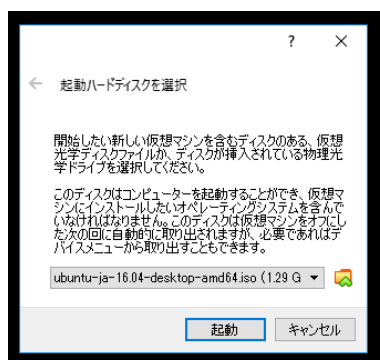
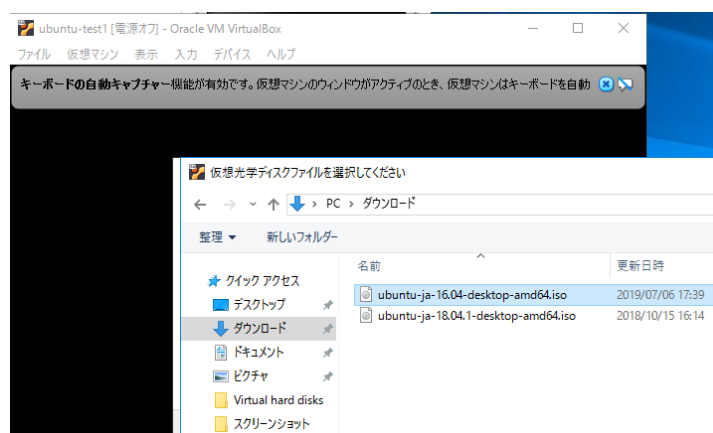
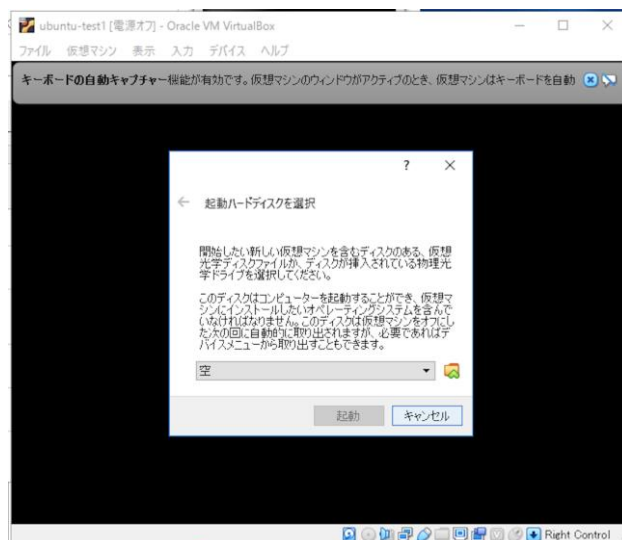
完了すると、仮想マシンの領域を作成することができます。いわゆる、OS を未インストール状態のマシンが出来上がっている状態です。初回は、何かしらのイメージから起動し、インストール作業を行います。予め [設定] からディスクイメージを認識させる方法もありますが、ディスクイメージを認識させていない場合は、起動ハードディスクをどのようにするか聞かれますので、そのまま [起動] を選択し、仮想マシンを起動させます。



起動ディスクを問われますので、事前に準備している Ubuntu Desktop のイメージファイルを利用して、インストールを実施します。

ここでは Ubuntu 16.04 “ubuntu-ja-16.04-desktop-amd64.iso” を利用します。

(ディスクイメージがない場合は、インターネットからもダウンロード可能です。)



インストール終了後、再起動しますが、シャットダウンに失敗して無反応になることがあります。その時はVirtualBox から強制的に仮想マシンの電源を切って再起動してかまいません。再起動後、Ubuntu Desktop が正常に使えることを確認し、ネットワークが接続できることを下記の項目で確認します。

- VM のターミナルから、ホストに ping が成功する
- ホストから VM に ping が成功する
- VM のブラウザから Google などの外部 Web サイトへアクセスできる

また、標準のインストールでは、ゲスト OS がホスト OS から孤立した環境として存在しますが、たとえば、ホスト OS の (Windows の) フォルダとゲスト OS の (Ubuntu の) ディレクトリの紐づけや、マウスポインタの統合、クリップボードの共有、自動ログイン等が可能となります。

ここでは、ファイル操作を簡単にするため、ゲスト OS とホスト OS のフォルダを紐付ける作業を行います。Ubuntu 側で、Terminal を開き、次のコマンドを入力します。

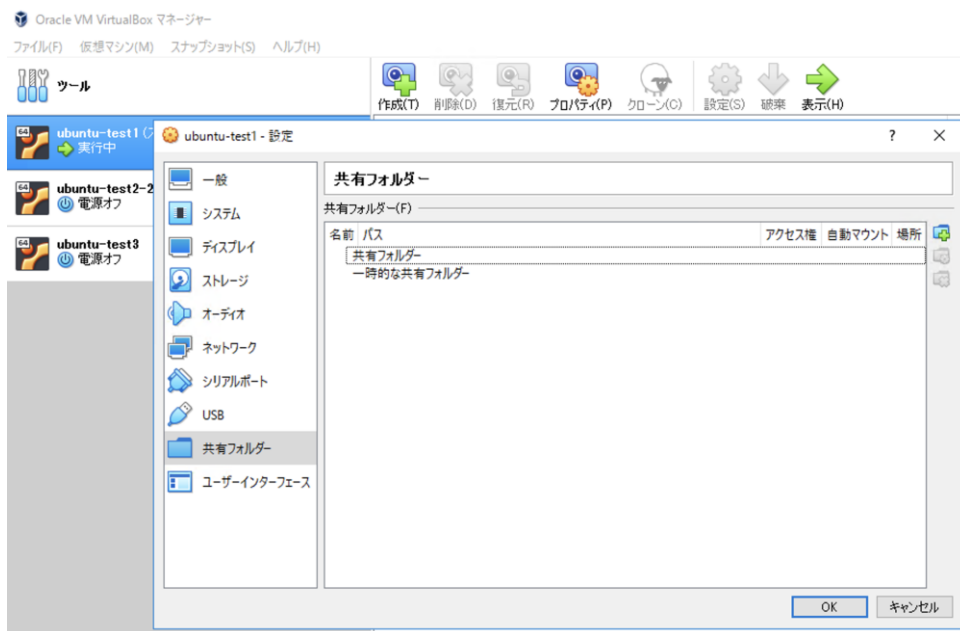
(VirtualBox Guest Additions をインストール)

```
$ sudo apt install virtualbox-guest-dkms
```

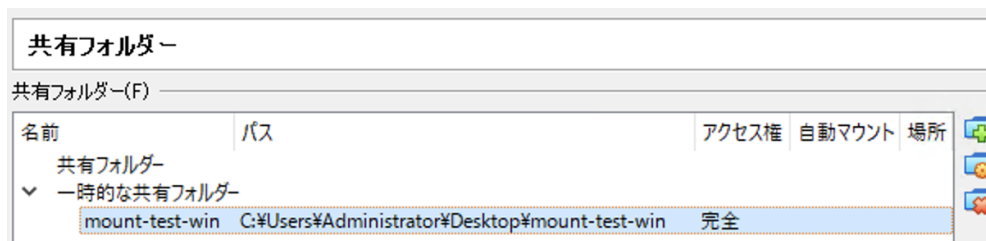
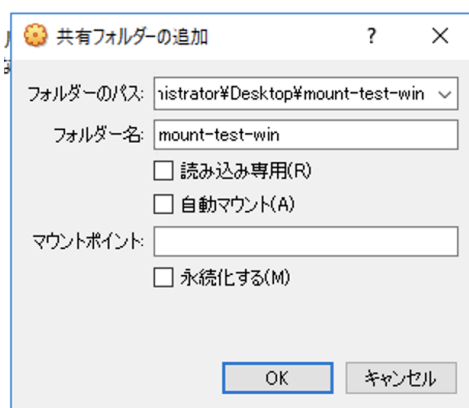
```
ubuntu@ubuntu1:~$ sudo apt install virtualbox-guest-dkms
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
以下の追加パッケージがインストールされます:
  dkms virtualbox-guest-utils
```

```
DKMS: install completed.
systemd (229-4ubuntu21.21) のトリガを処理しています ...
ureadahead (0.100.0-19) のトリガを処理しています ...
ubuntu@ubuntu1:~$ █
```

VirtualBox マネージャーで設定したいゲスト OS を選択し、[設定]→[共有フォルダー]を選択します。



ここでは、例として Windows のデスクトップに準備したフォルダを共有フォルダーに指定します。



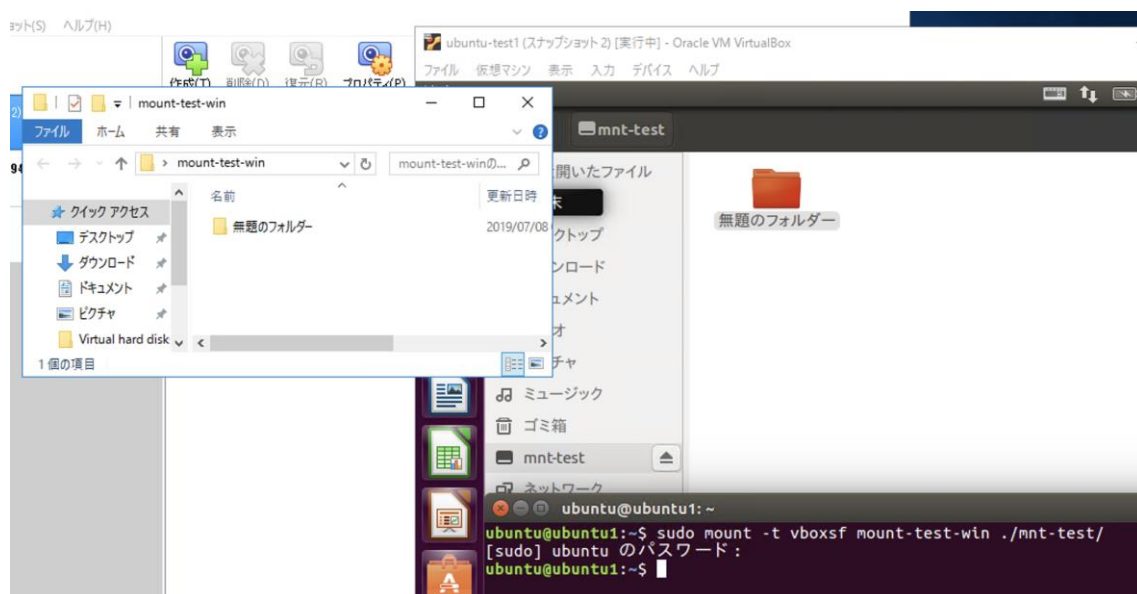
マウントポイントを更新するため、ホスト OS (Ubuntu) を再起動します。その後、ホスト OS 側の Terminal で次のコマンドを入力します。

(VirtualBox Guest Additions をインストール)

```
$ mkdir mnt-test
```

```
$ sudo mount -t vboxsf mount-test-win mnt-test
```

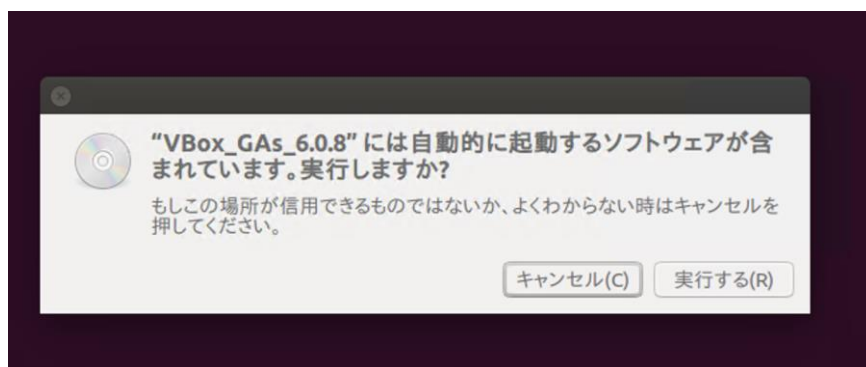
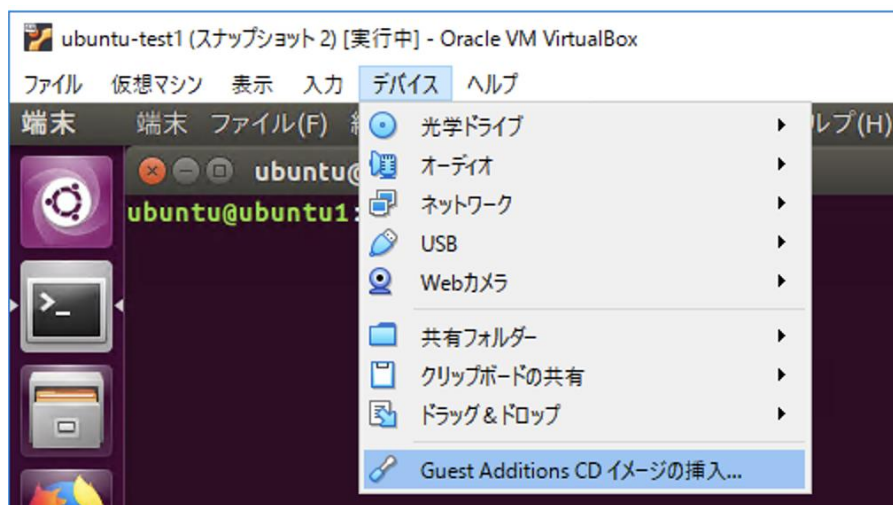
(sudo mount -t vboxsf [Windows 側の共有フォルダ名 (VirtualBox で設定したもの)] [Ubuntu 側のディレクトリ名])



すると、ホスト OS 側の共有フォルダとゲスト OS 側のディレクトリを連携させることが可能となります。

また、クリップボードの共有の設定が可能です。

[Guest Additions CD イメージの挿入]を選択し、イメージをマウントします。



VBox_Gas_6.0.8 を実行するために、ホスト OS 管理者のパスワードを入力します。



動作すると、プログラムを実行させて良いか問われるので、 y と入力して続行します。

```
× ◯ ◻ 端末
Verifying archive integrity... All good.
Uncompressing VirtualBox 6.0.8 Guest Additions for Linux.....
VirtualBox Guest Additions installer
This system appears to have a version of the VirtualBox Guest Additions
already installed. If it is part of the operating system and kept up-to-date,
there is most likely no need to replace it. If it is not up-to-date, you
should get a notification when you start the system. If you wish to replace
it with this version, please do not continue with this installation now, but
instead remove the current version first, following the instructions for the
operating system.

If your system simply has the remains of a version of the Additions you could
not remove you should probably continue now, and these will be removed during
installation.

Do you wish to continue? [yes or no]
y
```

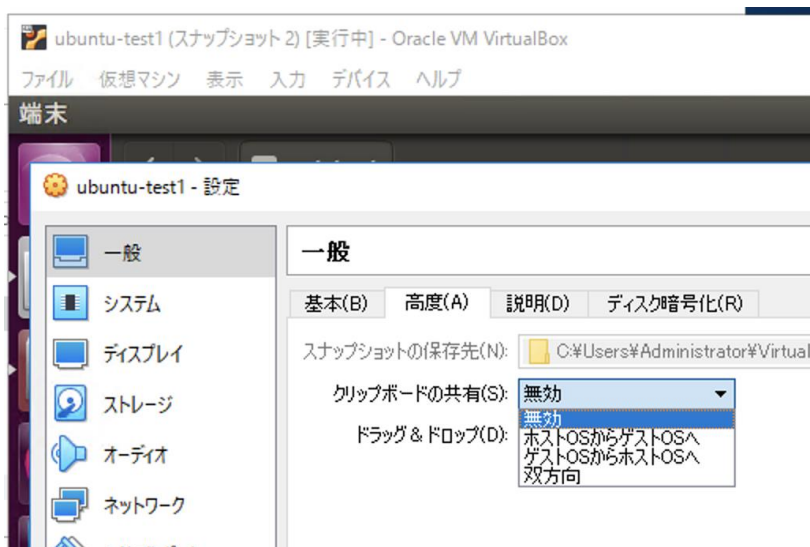
最終、 Press Return to close this window... と表示されれば成功です。

```
× ◯ ◻ 端末

If your system simply has the remains of a version of the Additions you could
not remove you should probably continue now, and these will be removed during
installation.

Do you wish to continue? [yes or no]
y
touch: '/var/lib/VBoxGuestAdditions/skip-4.4.0-21-generic' に touch できません:
そのようなファイルやディレクトリはありません
Copying additional installer modules ...
Installing additional modules ...
VirtualBox Guest Additions: Starting.
VirtualBox Guest Additions: Building the VirtualBox Guest Additions kernel
modules. This may take a while.
VirtualBox Guest Additions: To build modules for other installed kernels, run
VirtualBox Guest Additions: /sbin/rcvboxadd quicksetup <version>
VirtualBox Guest Additions: or
VirtualBox Guest Additions: /sbin/rcvboxadd quicksetup all
VirtualBox Guest Additions: Building the modules for kernel 4.4.0-21-generic.
update-initramfs: Generating /boot/initrd.img-4.4.0-21-generic
VirtualBox Guest Additions: Running kernel modules will not be replaced until
the system is restarted
Press Return to close this window...
```

最後に、対象となるゲスト OS の[仮想マシン]メニューから[一般]→[高度]と進むことで、こちらもゲスト OS を再起動することで有効となります。



演習 3 仮想マシンの複製

仮想マシンは単なるファイルなので、そのままコピーすればバックアップできます。しかし、同じ仮想マシンを複製したい場合、ファイル名の情報が内部に書き込まれていて、表層の名前を書き換えただけでは複製できません。ここでは、仮想マシンの複製手順について説明します。

① スナップショットの作成

スナップショットは、ある時点での仮想マシンの状態を抜き出したもののことを指します。スナップショットを作成することで、その時の状態を保存することができます。

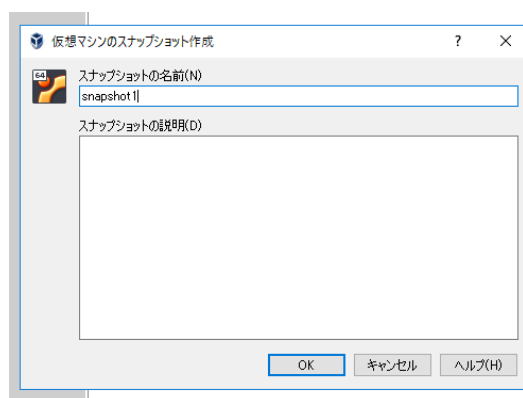
VirtualBox メニューバーにある[仮想マシン]→[ツール]→[スナップショット]と進みます。



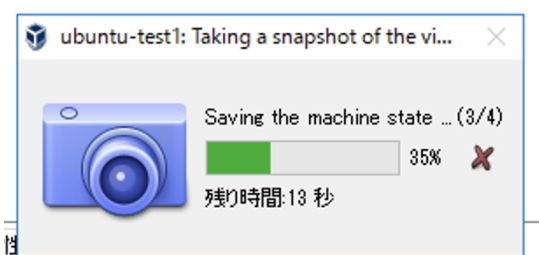
何も手を加えないうちは、最新の状態のみが存在します。[作成]を選択肢、スナップショットを作成します。



新規にスナップショットを作成します。スナップショットの名称を決めて、適切に説明を書き加えます。どのようなときのスナップショットなのか、あるいはバージョン情報や日時情報を加えると良いでしょう。ここでは“snapshot1”としておきます。



スナップショットが作成されるのを待ちます。



複数のスナップショットが存在する場合、過去の作成成分をたどることができます。



② クローンの作成

VirtualBox では、仮想イメージと完全に同じものをクローンとして複製することができます。基本的にはスナップショットを元に、クローンを作成します。

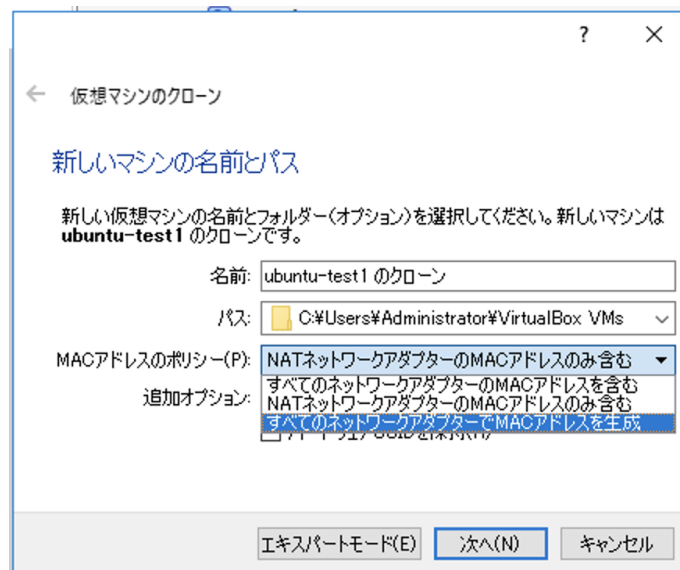
なお、ゲスト OS の電源が ON のままクローンを作成することもできるのですが、MAC アドレスの重複等が発生してしまうため、クローンを作成する場合には、ゲスト OS の電源を OFF にしてから作業を行います。

作成する状態を選択肢、[クローン]（羊のアイコン）をクリックします。



新しい仮想マシンを作成する際に、名前等を入力します。

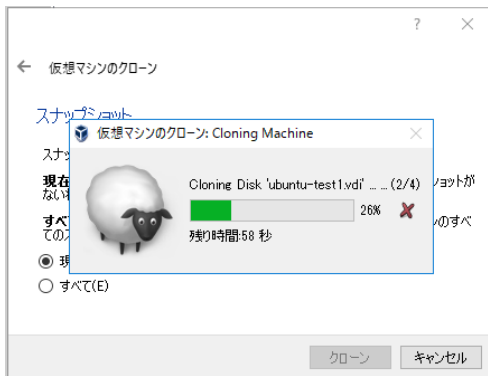
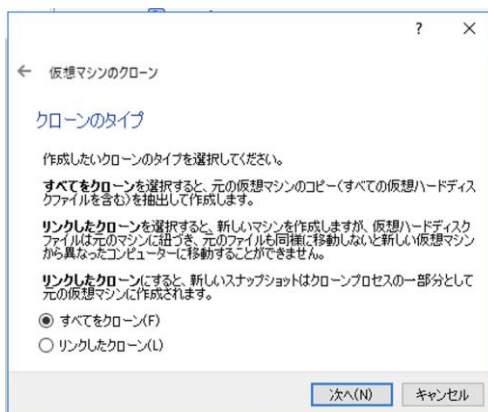
既存の仮想マシンと MAC アドレスの重複を避けるために、[MAC アドレスのポリシー]については[すべてのネットワークアダプターで MAC アドレスを生成]を選択してください。



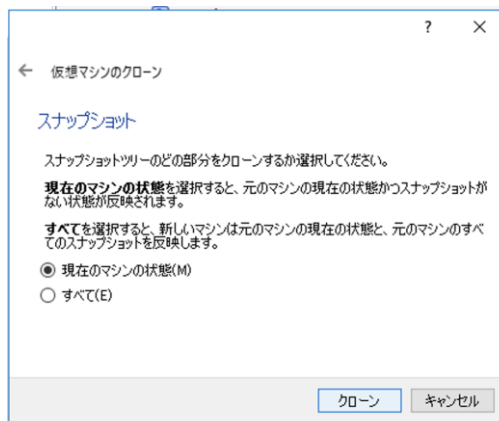
クローンのタイプを選択します。

- ・ すべてをクローン … もとの仮想マシンのコピーを抽出してクローンを作成します。
- ・ リンクしたクローン… 新しいマシンを制作する際に、仮想ハードディスクを既存の仮想マシンに紐付いて、クローンを作成します。そのため、既存の仮想マシンの部分的なプロセスとして動作・作成することになります。

今回は独立した運用を試みますので、[すべてをクローン]を選択します。



スナップショットのどの部分をクローンとするのか選択します。これまでのスナップショットの履歴をすべて選択する場合はすべてを選択します。今回は[現在のマシンの状態]を選択します。その後、[クローン]に進みます。



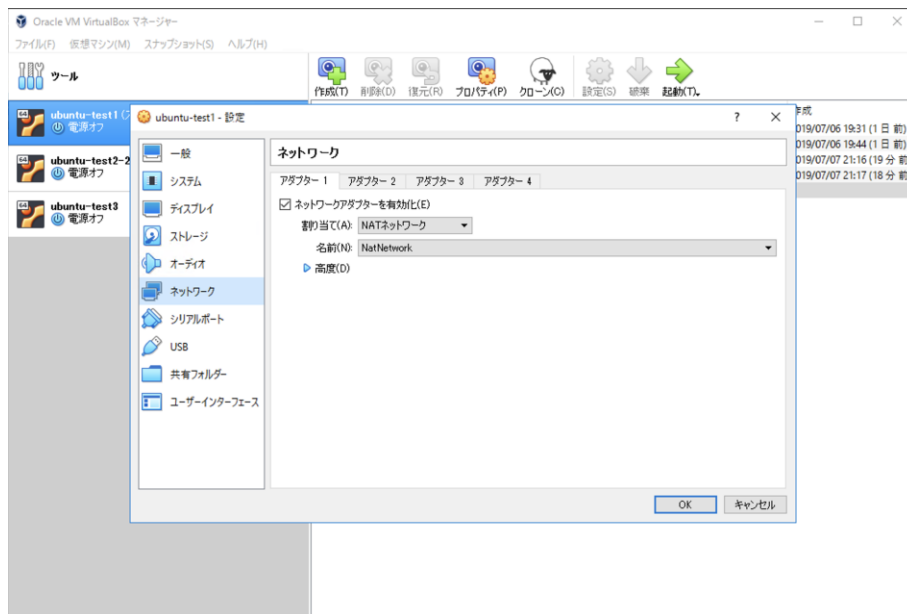
クローン生成が成功すると、左側のペインに、新しい項目が増えます。



演習 3 仮想ネットワークの設定変更

VirtualBox では基本的に仮想ネットワーク機器は、ホストオンリー、NAT、NAT ネットワーク、ブリッジアダプター、内部ネットワークといった複数種類のネットワークから選択することができます。この演習では NNAT ネットワーク、ホストオンリーおよびブリッジの違いを学習します。

VirtualBox の基本メニューから [設定] を選択し、[ネットワーク] を選択します。

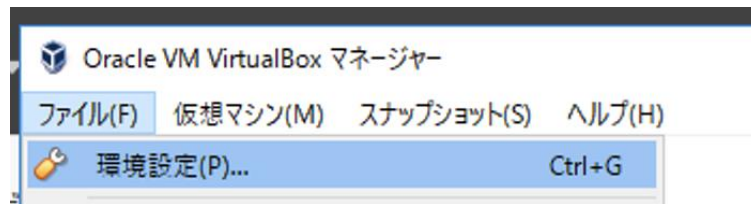


[アダプター1] (ホスト OS が利用しているネットワークアダプタ) に対して、どのようなネットワークにするかを決めます。

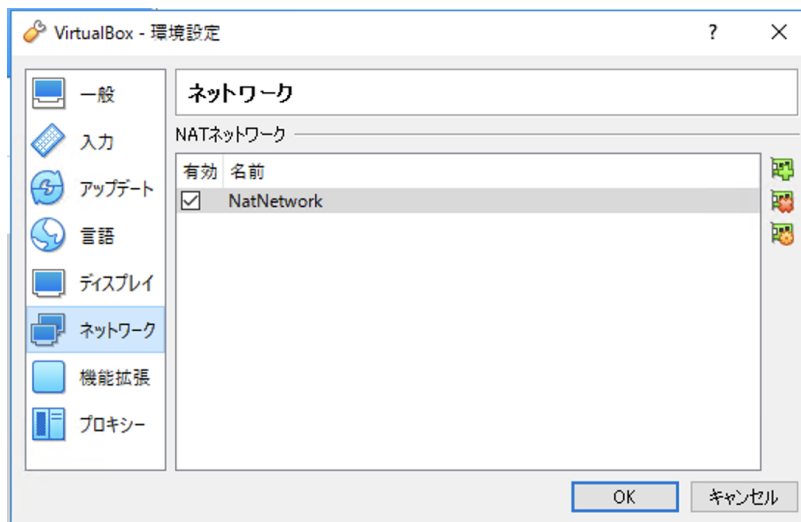



① 仮想スイッチ（NAT ネットワーク）の設定

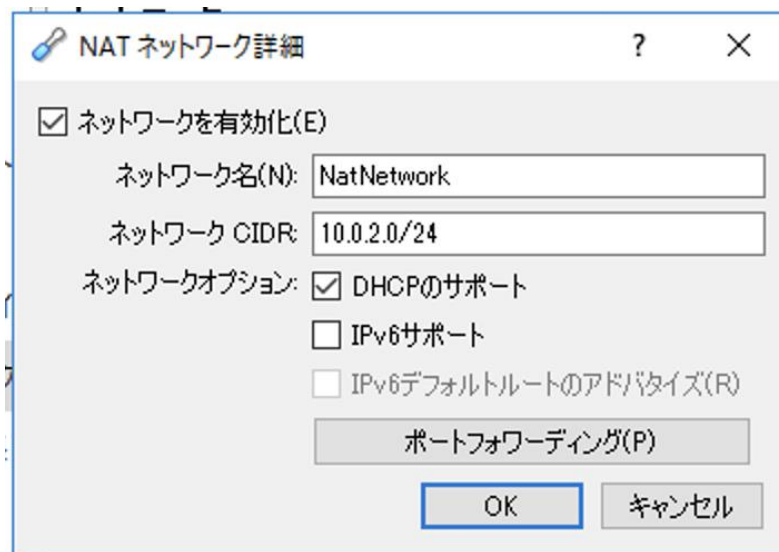
[ファイル]→[環境設定]と進み、[ネットワーク]の項目に進みます。



右端の「+」マークを押すと、NAT ネットワークを生成することができます。



右端の「歯車マーク」を押すと、NAT ネットワークの設定が可能です。ここでは標準的に与えられたネットワークの設定を行います。



また、ゲスト OS ごとにそれぞれのアダプターの機能を変更することが可能です。
ゲスト OS ごとに、[設定]→[ネットワーク]と進むことで、ネットワークアダプターに対してどのように設定するか選択できます。

例 1) ネットワーク無効



すべてのネットワークアダプターを無効にしてゲスト OS を起動すると、ネットワークインターフェースが存在しなくなります。



例2) ホストオンリーアダプタ

ホストOSとゲストOSでの疎通は可能になりますが、ゲストOS同士での疎通を認めません。



例3) ブリッジアダプター

ホスト OS に接続されているネットワークに対して、仮想スイッチングハブを接続することと同等になります。そのため、ホスト OS とゲスト OS が同一のネットワークに存在することになります。



例4) NAT ネットワーク (Network Address Transfer)

ホスト OS に接続されているネットワークに対して NAT ネットワークを構築します。そのため、複数台ゲスト OS が起動している場合、NAT 配下の同一ネットワークにゲスト OS を存在させることが可能になります。どのようなネットワークとするかは、冒頭の「NAT ネットワーク」をどのような設定にするかで変化します。



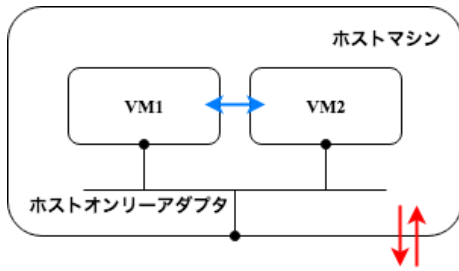
NAT ネットワーク配下の IP アドレスが割り振られていますが、DNS サーバ等は、上位のネットワークで設定されたものを継承しています。



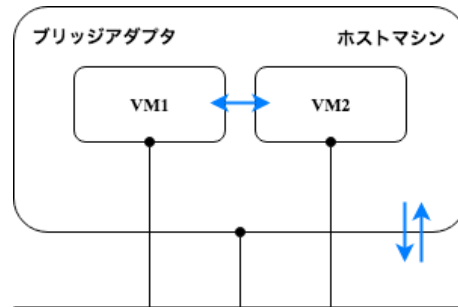
(参考) ネットワークアダプタの違い

- 通信可能
- 通信不可能

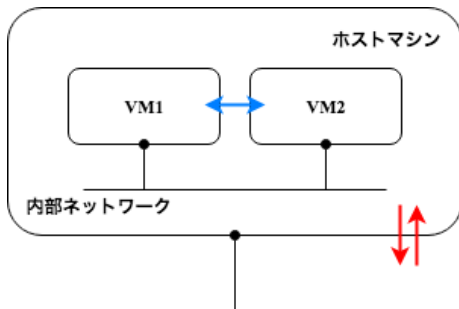
・ ホストオンリーアダプタ



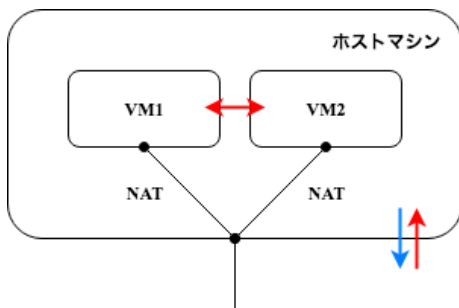
・ ブリッジアダプタ



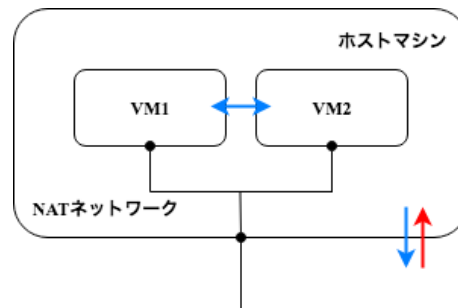
・ 内部ネットワーク



・ NAT



・ NAT ネットワーク



② 仮想マシンを仮想 NIC に接続して確認（ブリッジ接続）

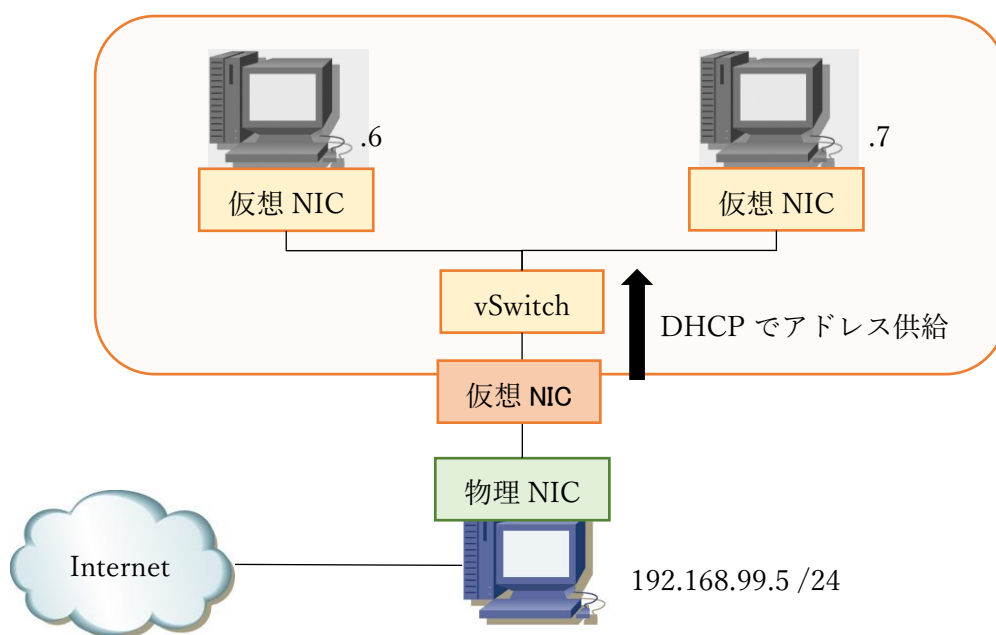
ここで、仮想マシンをもう 1 台作成し、サーバと 2 台の仮想マシンの間でネットワーク通信が可能かどうか確認します。

まず、外部に設定された仮想スイッチに接続した場合です。次の各項目をチェックしましょう。

- VM のターミナルから、ホスト同士に ping が成功する
- ホストから各 VM に ping が成功する
- 各 VM が同じネットワークに接続されている
- 他グループの各 VM への ping が成功する
- 各 VM のブラウザから Google などの外部 Web サイトへアクセスできる

この時、ネットワークは次のようになっています。

※この例ではネットワークは 192.168.99.0/24



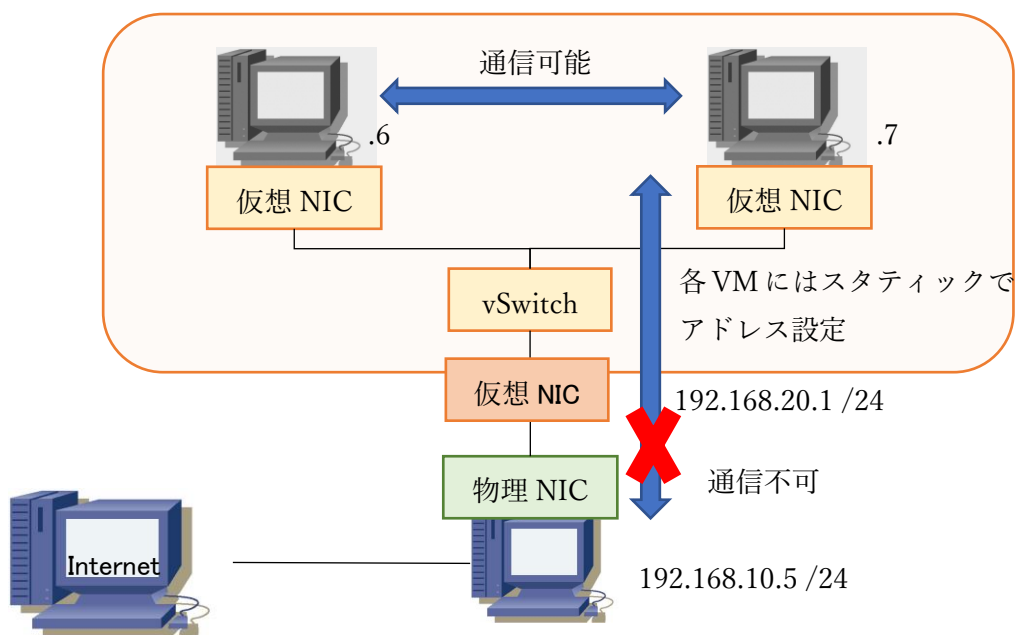
③ 仮想マシンを仮想NICに接続して確認（内部ネットワーク）

次に内部ネットワークに切り替えます。内部ネットワークはVM同士で通信できても、VMと物理ホストは通信できません。そのため、仮想ネットワークの影響を物理ネットワークに極力与えないような設計になっています。IPアドレス等は内部ネットワークと同じくスタティックで設定します。演習としては、接続先スイッチを変更するだけで、仮想NICの設定を変更する必要はありません。

ここで、次の項目を確認しましょう。

- VMのターミナルから、ホスト同士にpingが成功する
- ホストと各VM同士でpingが失敗する
- 各VMが同じネットワークに接続されている
- 他グループの各VMへのpingは成功しない
- 各VMのブラウザからGoogleなどの外部Webサイトへアクセスできない

この時、ネットワークは次のようになっています。



④ 仮想マシンを仮想 NIC に接続して確認 (NAT ネットワーク)

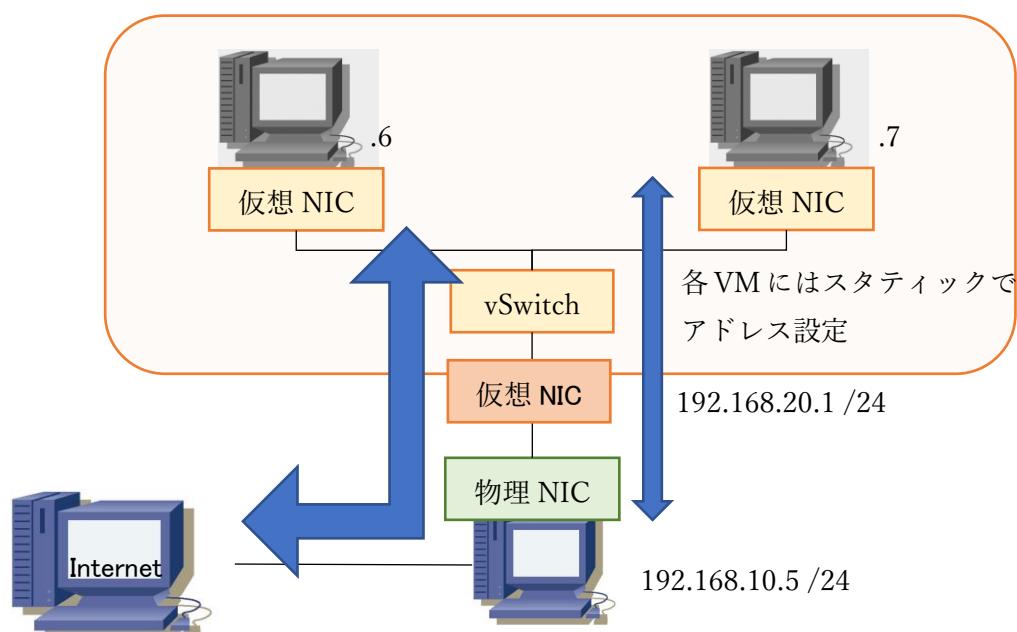
最後に NAT ネットワークです。NAT ネットワークに設定し、それぞれの設定について検証します。各仮想 NIC の IP アドレス設定は次のようになります。

物理ホストの内部用仮想 NIC	仮想マシンの仮想 NIC
IP アドレス: 192.168.20.1/24	IP アドレス: 192.168.20.5/24 など
デフォルトゲートウェイ: なし	デフォルトゲートウェイ: 192.168.20.1
DNS: なし	DNS: 物理ネットワークの DNS

以上の設定が完了したら次の項目について確認しましょう。

- VM のターミナルから、ホスト同士に ping が成功する
- ホストから各 VM に ping が成功する
- 各 VM が同じネットワークに接続されている
- 他グループの各 VM への ping は失敗する
- 各 VM のブラウザから Google などの外部 Web サイトへアクセスできる

また、この時のネットワークは次のようになります。



演習 4 Docker のインストールと基本設定

(準備)

コンテナソフトウェアの Docker をインストールします。基本的に、オフィシャル Web サイトのインストール方法の通りにコマンドを実行すればインストールできます。

Get Docker CE for Ubuntu

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

(Docker 用のリポジトリを追加)

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates curl software-properties-
common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository ¥
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu ¥
    $(lsb_release -cs) ¥
    stable"
```

(Docker のインストール)

```
$ sudo apt-get update
$ sudo apt-get install docker-ce
```

(Docker のバージョン確認)

```
$ sudo docker version
Client:
 Version:      18.03.1-ce
 API version:  1.37
(略)

Server:
 Version:      18.03.1-ce
 API version:  1.37 (minimum version 1.12)
(略)
```

一般ユーザで Docker が動作するように設定します。

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

※すでに docker グループが存在しているとエラーが出る場合がありますが支障ありません。

一度グループポリシーを反映させるために、現在のシェルを抜け出して、再度ログインを行います。

① コンテナの起動

テスト用コンテナ hello-world の起動実験を行います。 docker run --rm hello-world を実行し、下記のようなメッセージが出れば成功です。

```
$ docker run --rm hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:445b2fe9afea8b4aa0b2f27fe49dd6ad130dfe7a8fd0832be5de99625dad47cd
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://cloud.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/engine/userguide/>

② Ubuntu コンテナの使用

docker のコマンドで、Ubuntu イメージコンテナを起動します。

```
$ docker run -i -t ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
```

```
latest: Pulling from library/ubuntu
50aff78429b1: Pull complete
f6d82e297bce: Pull complete
275abb2c8a6f: Pull complete
9f15a39356d6: Pull complete
fc0342a94c89: Pull complete
Digest: sha256:ec0e4e8bf2c1178e025099eed57c566959bb408c6b478c284c1683bc4298b683
Status: Downloaded newer image for ubuntu:latest
root@fc5fc9c960fb:/#
```

コマンド入力が可能なように、`-i -t` オプションで標準入出力と擬似端末を有効にします。`/bin/bash` を明示的に記載することで、`bash` を利用できます (Ubuntu イメージの標準コマンドは `/bin/bash` なのですが、他の Docker イメージの場合、デフォルト指定がない場合があるのであえて明記します)。また、`-i -t` オプションは `-it` とつなげることも可能です。

確実に Ubuntu が動作しているか、`/etc/lsb-release` ファイルを表示して確認します。

```
root@fc5fc9c960fb:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04 LTS"
```

この Ubuntu イメージでは Ubuntu 18.04 LTS が動作していました。

なお、ネットワークの設定を確認すると、`link-local` しか設定されておらず、異なるホストを起動させていることが確認できます。

```
root@fc5fc9c960fb:/# ifconfig
bash: ifconfig: command not found
root@fc5fc9c960fb:/# cat /etc/networks
# symbolic names for networks, see networks(5) for more information
link-local 169.254.0.0
```

次にテキストファイルを作成してみましょう。ここでは、`hello.txt` を生成します。

```
root@fc5fc9c960fb:/# echo Hello > hello.txt
root@fc5fc9c960fb:/# ls
bin boot dev etc hello.txt home lib lib64 media mnt opt proc root run sbin srv
sys tmp usr var
```

```
root@fc5fc9c960fb:/# cat hello.txt
Hello
```

この時点で、Hello と書き込まれた hello.txt の存在を確認できます。

再度コンテナを起動させて、作成したファイルの存在を確認します。exit コマンドでログアウトします。

```
root@fc5fc9c960fb:/# exit
exit
$
```

その後、再び docker run -i -t ubuntu /bin/bash コマンドで Ubuntu イメージを立ち上げ、hello.txt の中身を確認します。

```
$ docker run -i -t ubuntu /bin/bash
root@6f32f125be9a:/# cat hello.txt
cat: hello.txt: No such file or directory
root@6f32f125be9a:/#
```

すると、先ほど作成した hello.txt が確認できません。docker run コマンドは、実行するたびに別の新しいコンテナを作成し実行するためです。

exit コマンドでコンテナを終了します。

それではここでこれまで使用したコンテナのサイズを見てみましょう。コンテナのサイズは docker images コマンドで確認できます。

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	22aebb614c1c	11 days ago	111MB
hello-world	latest	f2a91732366c	5 weeks ago	1.85kB

③ コンテナの管理

docker ps コマンドで現在動作しているコンテナの確認をします。

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	

すべてのコンテナが終了している場合は何も出力されません（ラベルのみの出力）。

②で実施した Ubuntu イメージを立ち上げたまま、別のシェルでコンテナを確認してみます。

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
22aebb614c1c       ubuntu             "/bin/bash"        47 seconds ago
Up 37 seconds      amazing_heyrovsky
```

コンテナ ID 22aebb614c1c で Ubuntu イメージのコンテナが動作していることが確認できます。 docker kill コマンドでコンテナ ID を指定してコンテナを終了します。

```
$ docker kill 22aebb614c1c
22aebb614c1c
```

docker kill の場合、コンテナは終了しましたがコンテナ自体はまだ残っており、コンテナ内で作成したファイル等は削除されていません。終了したコンテナも含めたコンテナの一覧を確認するためには、 docker ps -a コマンドを使います。

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
22aebb614c1c       ubuntu             "/bin/bash"        6 minutes ago
Exited (137) 57 seconds ago      amazing_heyrovsky
```

完全にコンテナを削除する場合には、 docker rm コマンドを使います。ここでは、すべてのコンテナをまとめて削除するように指定します。

```
$ docker rm -f $(docker ps -a -q)
22aebb614c1c
```

削除の際、削除したコンテナの ID が出力されます。

④ Web サーバコンテナの使用

docker のコマンドで、Web サーバコンテナを起動してみましょう。ここでは軽量 Web サーバアプリケーションとして有名な nginx (engine x、 エンジンエックス)を利用します。nginx イメージで、Web サーバを起動します。

```
$ docker run -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
e7bb522d92ff: Pull complete
0f4d7753723e: Pull complete
91470a14d63f: Pull complete
Digest: sha256:edc8182581fdaa985a39b3021836aa09a69f9b966d1a0ff2f338be6f2fbfe238
```

```
Status: Downloaded newer image for nginx:latest
```

プロンプトが返ってきませんが、成功すると、nginx が起動します。Docker ホストの 8080 番ポートに接続しましょう。

```
http://Docker ホストの IP アドレス:8080/  
(例) http://192.168.99.99:8080/
```

ブラウザに下図のように表示され、Web サーバに接続できたことが確認できます。

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

終了するときは、Ctrl+C で終了します。

1 台のマシンの上にコンテナを複数個作成することもできます。以下のコマンドで、Web サーバ (nginx) を 100 個同時起動してみます。

```
$ for port in $(seq 8001 8100); do docker run -d -p $port:80 nginx; done
```

わずかな時間で 100 個起動させることが可能です。ポート番号を 8001~8100 で設定しているので、それぞれのポート番号に対して、どこでもアクセスすることが可能です。同じ Docker イメージを利用しているため、ディスク消費が少ないです。

演習 5 コンテナのデータ保存と独自コンテナの作成

① Docker データの永続化

Docker は起動とコンテナを作成、終了すると破棄します。そのため、コンテナ内に作成したデータは破棄され保存されません。そのため、もしデータを生成して保存したい場合、データの永続化を行う必要があります。データ永続化の手法はいくつかありますが、ここでは最も手軽なローカルファイルのマウントを紹介します。

まずローカルに保存用のディレクトリを作成し、確認のためデータを書き込みます。

```
$ mkdir data
$ echo dockertest > data/test.txt
$ cat data/test.txt
Dockertest
```

次に、このディレクトリを-vオプションで紐付けてubuntuコンテナを起動します。起動後、ディレクトリ一覧を表示させると、マウントしたdataディレクトリが現れます。

```
$ docker run -it -v $HOME/data:/data ubuntu /bin/bash
root@aa27f97dcfef:/# ls
bin  data  etc   lib   media opt   root sbin sys  usr
boot dev  home lib64 mnt  proc run  srv  tmp  var
```

ローカル側で作成したファイルを表示させると、ちゃんと中味が見えます。

```
root@aa27f97dcfef:/# cat data/test.txt
dockertest
```

ではコンテナ内でファイルを生成して書き込み、コンテナを終了します。

```
root@aa27f97dcfef:/# echo dockertest2 > /data/test2.txt
root@aa27f97dcfef:/# cat /data/test2.txt
dockertest2
root@aa27f97dcfef:/# exit
exit
```

コンテナ終了後も、マウントしていたdataディレクトリ内にコンテナ内で作成したファイルが残っていて、内容も確認できました。

```
$ ls data/
test2.txt test.txt
```

```
$ cat data/test2.txt
dockertest2
```

② 独自の Docker イメージの作成と配布

独自の Docker イメージを作成し、Docker Hub を通じて配布できるようにします。そのため、あらかじめ Docker Hub (<https://hub.docker.com/>) にアカウントを作成しておきます。適当な作業用ディレクトリを作成し、そこで Dockerfile というファイルを作成します。

```
$ mkdir dockertmp
$ cd dockertmp
$ vim Dockerfile
```

Dockerfile 内にコンテナイメージの設定を書いていきます。ここでは以下の内容とします。FROM は元になるコンテナイメージで、ここでは非常に軽量な Alpine Linux を指定しています。実行するコマンドは CMD で指定しますが、複数のコマンドを実行する場合は、この例のように && で結びます。ここでは、Hello! World! と表示した後、ps コマンドでプロセスを表示し、最後に Good! と表示します。

```
FROM alpine
MAINTAINER 名前 <メールアドレス>
RUN echo "now build!"
CMD echo "Hello! world!" && ps && echo "Good!"
```

Dockerfile を保存したら、Docker build コマンドでコンテナイメージをビルドします。この時、アカウント名が test1、イメージ名が world-echo なら、コマンドは docker build -t test1/world-echo:1 . になります。最後のピリオドを忘れないようにしましょう。これは、Dockerfile のパスを表すもので、ここではカレントディレクトリで作業をしているので、同じ場所を表すピリオドにします。

```
$ docker build -t [アカウント名]/[イメージ名]:1 .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM alpine
latest: Pulling from library/alpine
2fdfe1cd78c2: Pull complete
Digest: sha256:ccba511b1d6b5f1d83825a94f9d5b05528db456d9cf14a1ea1db892c939cda64
Status: Downloaded newer image for alpine:latest
----> e21c333399e0
```

```
Step 2/4 : MAINTAINER 名前 <メールアドレス>
--> Running in d748d224ba5a
--> ae7da6f7cb6b
Removing intermediate container d748d224ba5a
Step 3/4 : RUN echo "now build!"
--> Running in 8556af03b7a5
now build!
--> f6fdedce2481
Removing intermediate container 8556af03b7a5
Step 4/4 : CMD echo "Hello! world!" && ps && echo "Good!"
--> Running in 286bf9723a3f
--> 493ed881f5f6
Removing intermediate container 286bf9723a3f
Successfully built 493ed881f5f6
Successfully tagged [アカウント名]/[イメージ名]:1
```

無事ビルドできたら、テストします。

```
$ docker run [アカウント名]/[イメージ名]:1
Hello! world!
PID   USER    TIME    COMMAND
   1  root      0:00  /bin/sh -c echo "Hello! world!" && ps && echo "Good!"
   6  root      0:00  ps
Good!
```

それでは、この Docker イメージを Docker Hub にアップロードしましょう。まずこのイメージにタグ付けをします。docker images でイメージ ID を調べ、そのイメージ ID に対して docker tag コマンドでタグ付けを行います。tag 付けは任意のものでかまいませんが、ここでは最新版を表す latest とします。

```
$ docker images
REPOSITORY          TAG    IMAGE ID          CREATED          SIZE
[アカウント名]/[イメージ名] 1      493ed881f5f6     9 minutes ago   4.14MB
```

これからこのイメージを Docker Hub にアップロードしますが、その前に Docker Hub に docker login コマンドでログインしておきます。

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
```

have a Docker ID、 head over to <https://hub.docker.com> to create one.

Username: [アカウント名]

Password:

Login Succeeded

いよいよアップロードです。

```
$ docker tag 493ed881f5f6 [アカウント名]/[イメージ名]:latest
$ docker push [アカウント名]/[イメージ名]
The push refers to a repository [docker.io/*****]
04a094fe844e: Layer already exists
latest: digest: sha256:e9a6eea923fb895920fcf38c6832746ac3e9a35e1905d7a45384af3
c8999b654 size: 528
```

無事アップロードできれば、Docker Hub にも表示されます。[https://hub.docker.com/u/\[アカウント名\]/](https://hub.docker.com/u/[アカウント名]/)でアクセスし、確認しましょう。また、他のチームの Docker イメージを試してみましょう。docker pull [アカウント名]/[イメージ名] で取得できます。

演習 6 コンテナの応用

(準備) 演習 4 で使用した Docker 環境を確認します。

(Docker のバージョン確認)

```
$ sudo docker version
Client:
 Version:      18.03.1-ce
 API version:  1.37
 (略)

Server:
 Version:      18.03.1-ce
 API version:  1.37 (minimum version 1.12)
 (略)
```

(Docker 関係コマンドの確認)

(イメージの検索)

```
$ docker search ruby
ruby の Docker イメージを Docker Hub から検索する
```

(イメージのダウンロード)

```
$ docker search ruby:2.3.1
Ruby:2.3.1 の Docker イメージを Docker Hub からダウンロードする
```

(イメージの一覧)

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
wordpress           latest             1d3cc82944da      7 days ago        408MB
php                 7.0-apache        7011510f1ff8      7 days ago        367MB
mysql              5.7               66bc0f66b7af      8 days ago        372MB
ubuntu             latest            00fd29ccc6f1      6 months ago     111MB
```

ダウンロードしたイメージの一覧を表示する

(コンテナの一覧)

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
```

11cc1f898379	wordpress:latest	"docker-entrypoint.s..."	About an hour ago
Up About an hour	0.0.0.0:80->80/tcp	dockerwordpress_wordpress_1	
f2fc48620de2	mysql:5.7	"docker-entrypoint.s..."	About an hour ago
Up About an hour	0.0.0.0:3306->3306/tcp	dockerwordpress_db_1	

実行中のコンテナ一覧を表示する。実行中でないものも表示する場合は下記コマンド。

(コンテナの一覧 - 実行中でないものも表示する)

-a: 停止したコンテナも含めて表示する

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
11cc1f898379	wordpress:latest	"docker-entrypoint.s..."	About an hour ago Up
About an hour	0.0.0.0:80->80/tcp	dockerwordpress_wordpress_1	
f2fc48620de2	mysql:5.7	"docker-entrypoint.s..."	About an hour ago Up
About an hour	0.0.0.0:3306->3306/tcp	dockerwordpress_db_1	
4b102f6d049d	php:7.0-apache	"docker-php-entrypoi..."	2 hours ago
Exited (0) About an hour ago		php70-apache	
93bcd3151af1	httpd	"httpd-foreground"	5 months ago
Exited (255) 2 hours ago	0.0.0.0:80->80/tcp	focused_mcclintock	
afd58e477525	httpd	"httpd-foreground"	5 months ago
Exited (0) 5 months ago		quirky_mahavira	

(コンテナの起動・実行)

```
$ docker run [オプション] IMAGE [コマンド] ...
```

- name: コンテナに任意で名前をつけることができます。
- rm: 実行後のコンテナを削除します。指定しない場合はゴミが残り続けます。
- v: ホストのディレクトリをコンテナ内のディレクトリにマウントします。"\$PWD"はカレントディレクトリを意味します。
- w: ワーキングディレクトリを指定します。

デタッチド・モード

- d: デタッチド・モードで起動する (バックグラウンド)

コンテナが実行するルート・プロセスが終了したら、デタッチド・モードで起動したコンテナも終了します。デタッチド・モードのコンテナは停止しても自動的に削除できません。つまり -d オプションでは --rm を指定できません。デタッチド・コンテナに再度アタッチ (接続) するには、docker attach コマンドを使います。

フォアグラウンド・モード

- i STDIN (標準入力) を開きます。
- t tty を割り当てます。

上記 -i -t はセットで使うとターミナルでコンテナを実行することができます。ターミナルを exit するとターミナルを終了し、コンテナも停止します。終了したくない場合は「CTRL + p + q」で抜けます。

(コンテナの起動・実行 - Hello-World)

```
$ docker run --rm hello-world
```

(コンテナの実行 - Ubuntu)

```
$ docker run -i -t ubuntu /bin/bash
```

(コンテナの停止・終了)

```
$ docker kill 22aebb614c1c (コンテナ ID)
22aebb614c1c
```

(コンテナの完全削除)

-f: 強制

```
$ docker rm -f $(docker ps -a -q)
22aebb614c1c
```

削除の際、削除したコンテナの ID が出力されます。

① 複数のサービスを同時に起動

docker のコマンドで、Web サーバコンテナと新しい PHP が一つのコンテナに含まれているものを起動してみましょう。軽量 Web サーバアプリケーションとして有名な Apache と PHP7 を必要最低限の環境で構築する方法を紹介します。

```
$ docker run -d -p 8070:80 --name php70-apache php:7.0-apache
4b102f6d049d8c78bdd7126095398053e92cb1c23418bc67b2ee8ac4e2afdc15
```

成功すると、Apache + PHP7 が起動します。Docker ホストの 8070 番ポートに接続しましょう。(例) <http://192.168.99.99:8070/>

Forbidden

You don't have permission to access / on this server.

Web サーバである Apache は起動しているのですが、表示すべきファイルが設置されていないために”Forbidden”と出力されてしまいます。

そのため、作成した php70-apache コンテナに bash でログインし、閲覧可能なページを作成します。

php-70-apache コンテナにログイン

```
$ docker container exec -ti php70-apache bash
root@0e501e9b25f0:/var/www/html#
```

すると、プロンプトにカレントディレクトリである `/var/www/html` が表示されます。念のためにディレクトリの存在を確認します。

```
root@0e501e9b25f0:/var/www/html# pwd
/var/www/html
```

カレントディレクトリの中身を確認

```
root@0e501e9b25f0:/var/www/html# ls
root@0e501e9b25f0:/var/www/html#
```

ディレクトリ上には何も無いことが確認できます。


`/var/www/html` のディレクトリ上に「`<?php phpinfo();?>`」と記載されたテキストファイルである `index.php` ファイルを作成します。

```
root@0e501e9b25f0:/var/www/html# echo '<?php phpinfo();?>' > index.php
```

`index.php` が出来上がっていることを確認します。


```
root@0e501e9b25f0:/var/www/html# ls
index.php
root@0e501e9b25f0:/var/www/html# cat index.php
<?php phpinfo();?>
```

再び Docker ホストの 8070 番ポートに接続しましょう。(例) `http://192.168.99.99:8070/`

PHP Version 7.0.30


System	Linux 0e501e9b25f0 4.4.0-112-generic #135-Ubuntu SMP Fri Jan 19 11:48:36 UTC 2018 x86_64
Build Date	Jun 28 2018 02:47:30
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS
PHP Extension Build	API20151012,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv2, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v3.0.0, Copyright (c) 1998-2017 Zend Technologies



Configuration

すると、PHP の情報がたくさん表示されます。これは、phpinfo 関数で表示された動作中の PHP に関する情報になります。この画面が見えているということは、PHP と WEB サーバである Apache が確実に動作しているということが確認できます。

② ホストとコンテナ間でディレクトリを同期する

このままでは、コンテナを削除したと同時にコンテナ内にあるファイルが削除されます。そのため、ホスト側とコンテナのディレクトリを同期させて、そこにファイルを書き込む形式を取ります。

コンテナからログアウトする (logout もしくは exit)

```
root@0e501e9b25f0:/var/www/html# exit
exit
```

先ほど作成したコンテナを停止、削除します。

```
$ docker container stop php70-apache
```

```
$ docker container rm php70-apache
```

新しくディレクトリが同期されるコンテナを作成します。ホスト側の同期対象となるディレクトリを「/home/ユーザ名/docker/php70-apache」（ここでは~/docker/php70-apache）、コンテナ側の対象ディレクトリはデフォルトの「/var/www/html」とします。ディレクトリを作成します。

```
$ mkdir ~/docker/php70-apache/
```

ディレクトリが同期されるコンテナを作成します。

```
$ docker run -d -p 8070:80 -v ~/docker/php70-apache:/var/www/html --name php70-apache php:7.0-apache
41daf7f2647bc016db8b3d60ba14b67644b0f33f3d36477ab9e2ff63a70c2e6a
```

~/docker/php70-apache 上に先ほどと同じように index.php ファイルを作成します。

index.php ファイルを作成

```
$ cd ~/docker/php70-apache/
$ echo '<?php phpinfo();?>' > index.php
$ cat index.php
<?php phpinfo();?>
```

再び Docker ホストの 8070 番ポートに接続しましょう。(例) <http://192.168.99.99:8070/> PHP の情報ページが表示されていれば完了となります。

③ docker-compose を利用して、複数のコンテナを同時に起動する

docker-compose を使うと、複数のコンテナから構成されるサービスをひとつに束ねることが可能となり、管理が容易になります。管理には YAML（ヤメル・ヤムル）形式のファイル（拡張子 .yml）を用います。

docker-compose をインストール

```
$ sudo apt-get install docker-compose
```

docker-compose のバージョンを確認します。

```
$ docker-compose --version
docker-compose version 1.8.0, build unknown
```

ファイル例 : docker-compose.yml

```
version: '2'
services:
  db:
    image: mysql:5.7
    volumes:
      - "$PWD/.data/db:/var/lib/mysql"
    ports:
      - "3306:3306"
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
      - "$PWD:/var/www/html"
    links:
      - db
    ports:
      - "8060:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
```

YAML ファイルの書式は docker run のオプションと対応しているので読み替えがききます。ここでは例として CMS (コンテンツ・マネジメント・システム) である WordPress の構築を挙げます。

WordPress の立ち上げに必要なもの :

- ・ WEB サーバ (Apache 等 + PHP 等)
- ・ データベース (MySQL、 PostgreSQL 等)

上記の docker-compose.yml では、データベースとして mysql、アプリケーションとして wordpress を指定しています。なお、ここでは便宜を図るためにデータベース名やパスワードには “wordpress” を用いています。

docker-compose.yml のファイル置き場を作成し、ファイルを作成します。

```
$ mkdir ~/docker/wordpress
$ cd ~/docker/wordpress/
```

docker-compose.yml ファイルを作成します。

```
$ touch docker-compose.yml
```

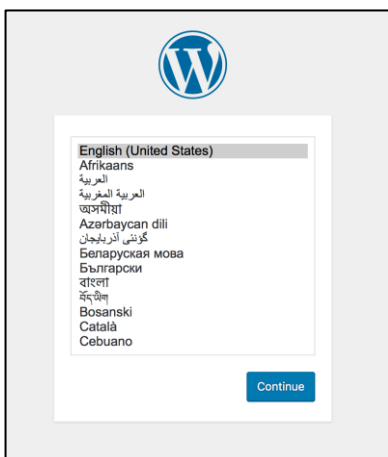
「ファイル例 : docker-compose.yml」に従ってファイルを作成します。適当なテキストエディタで編集してください。

プロジェクトの起動を行います。

```
$ docker-compose up -d
Creating network "dockerwordpress_default" with the default driver
Pulling db (mysql:5.7)...
5.7: Pulling from library/mysql
683abbb4ea60: Already exists
0550d17aeefa: Pull complete
7e26605ddd77: Pull complete
... 途中省略 ...
Digest: sha256:7122e8924cfb8bc1f4bc0d5a01f6df7d8186f5661c385511079c60c4feca5019
Status: Downloaded newer image for wordpress:latest
Creating dockerwordpress_db_1
Creating dockerwordpress_wordpress_1
```

これでページの作成は完了です。Docker ホストの 8060 番ポートに接続しましょう。

(例) <http://192.168.99.99:8060/>



WordPress のページが表示されていれば、インストール成功です。

