

高度ITを活用したビジネス創造プログラム 講師マニュアル

－仮想化講座 Vol.2－

プログラム概要

高度IT 技術を活用したビジネス創造プログラムの概要

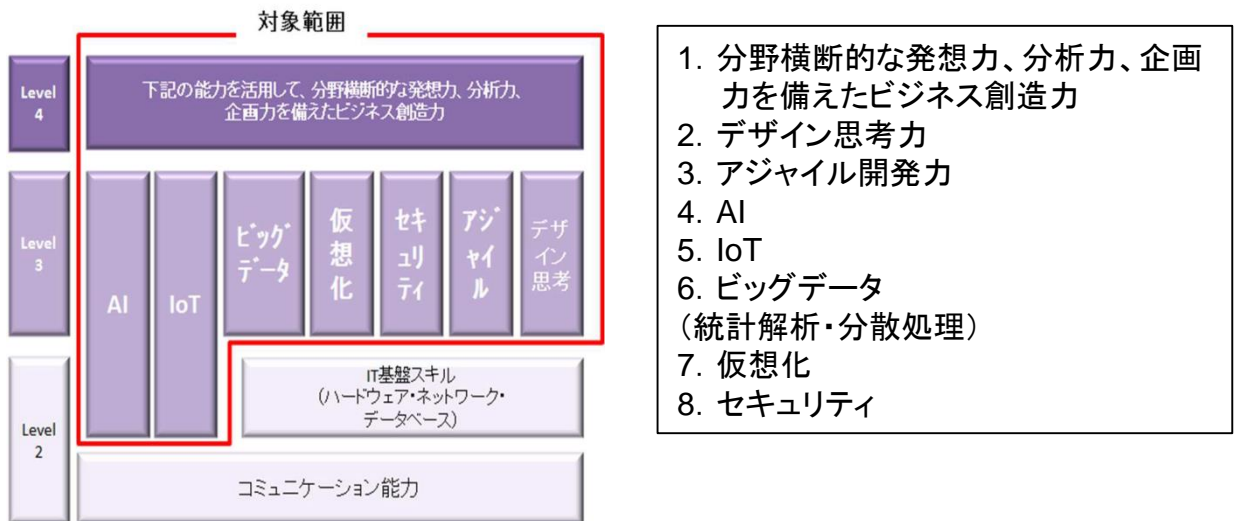
1. プログラム開発の目的と背景

■プログラムの目的

第4 次産業革命において必須であるIoT、AI やビッグデータに代表されるIT 系の技術を駆使し、新たな発想（サービス企画・デザイン思考）でビジネスを創造できる高度IT エンジニアを育成する。

■修得すべき能力とその理由

修得すべき能力を図で表すと下記のようなイメージになる。今回は一定のスキル（レベル2～3程度）を修得しているエンジニアを受講者に想定しているので、学習対象範囲の各能力についての教育訓練プログラムを作成する。



(図1)

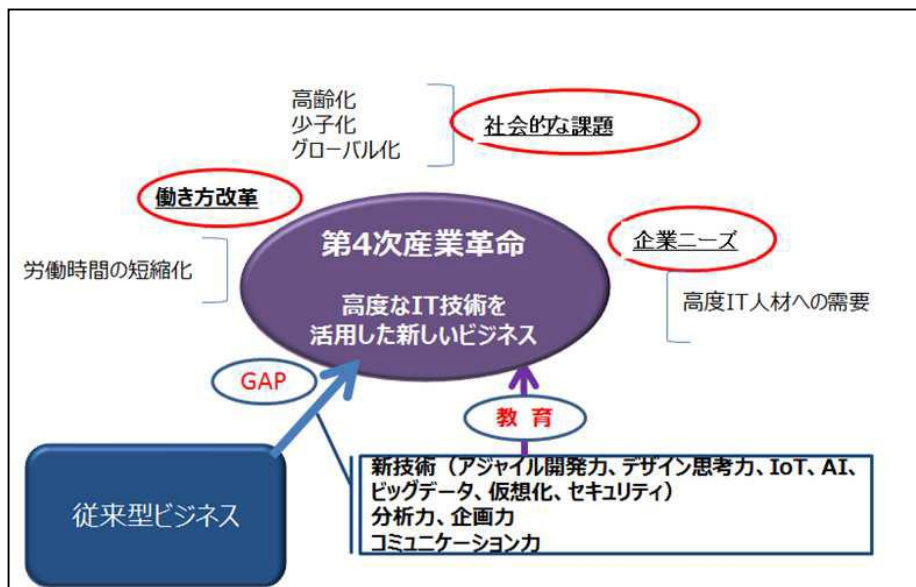
2. プログラム開発の背景

1) 「第4次産業革命」における必要性

現代社会において高齢化・少子化・グローバル化等の社会的な課題が山積する中で、それらの社会的な課題を解決することが求められている。それらの課題の解決には、新しい発想のビジネスが求められていて、それを実現するためには、高度なIT技術の活用が不可欠である。企業も社会的な課題を解決するための新たなビジネスチャンスを探ってはいるが、現状はほとんどの技術者が従来型ビジネスに対応した人材であり、高度なIT技術をもった人材へのニーズがある。

また、「働き方改革」という労働時間の短縮化という中で、労働時間が削減されても経済成長を促すには、単位時間あたりの労働生産性の向上が欠かせない。労働生産性の向上には、より高度なIT技術の修得が不可欠である。

このように、「社会的必要性」「企業ニーズ」「働き方改革」という3つの要因で上記に挙げた8つの能力が必要である。上述の内容を表したのが図2である。

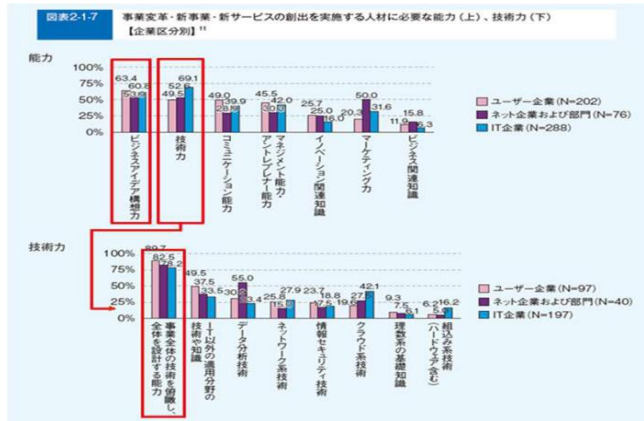


(図2)

2.プログラム開発の背景

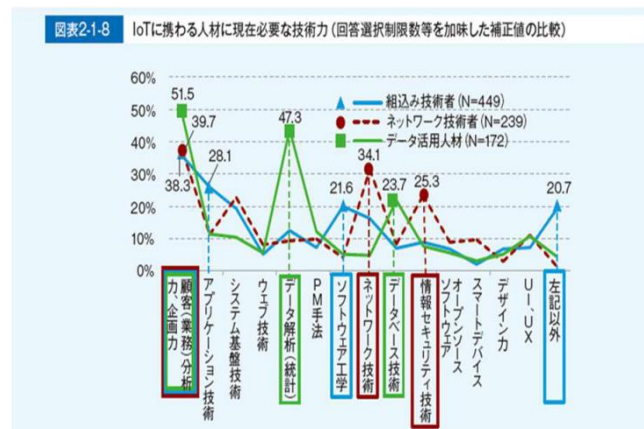
2)8 つの能力の根拠

図2のように、企業が新しいビジネスを創造するためには、従来型ビジネスとのGAPを埋めるための能力が必要であり、IPAの「IT人材白書2016」(資料1)によると「ビジネスアイデア構想力」「技術力」が求められる



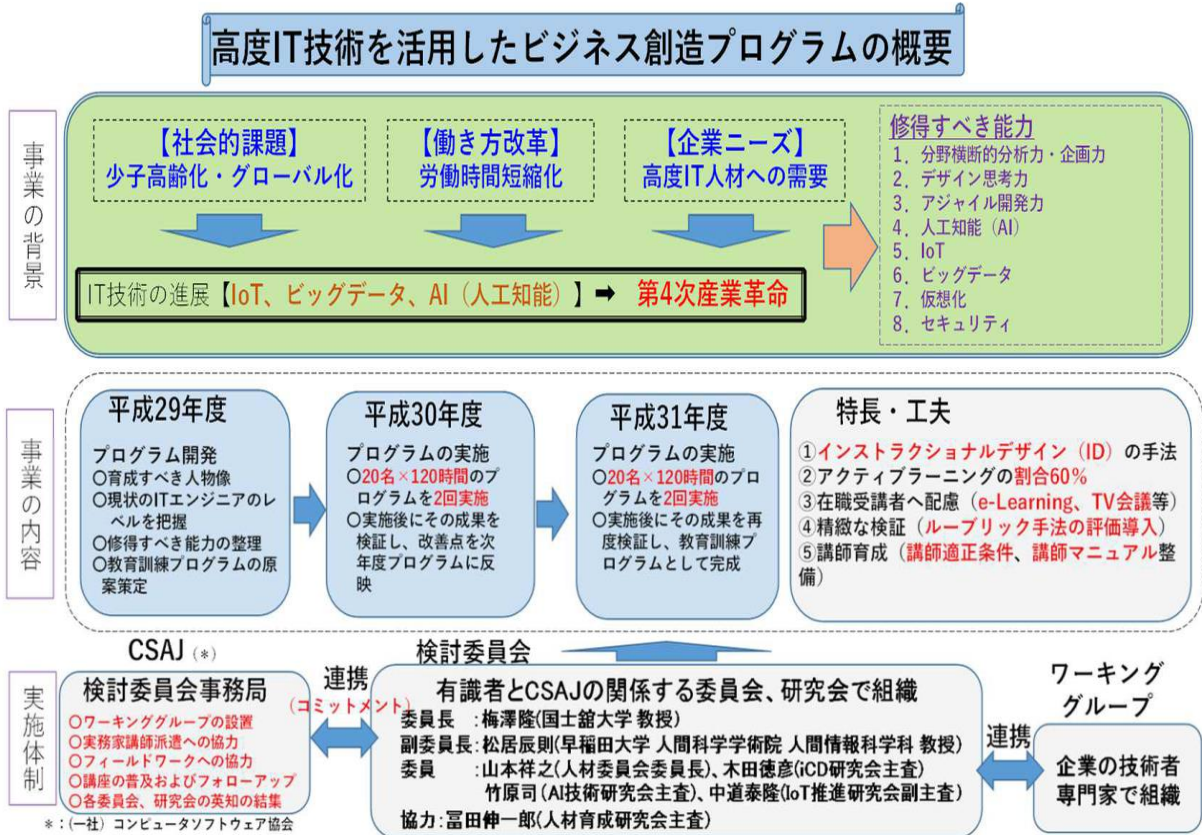
(資料1)
『新事業・新サービス創出に必要な能力・技術力とは？』
(IPA「IT人材白書2016」)

「ビジネスアイデア構想力」「技術力」とは何かを詳細に見てみると、例えば「IoTに関わる人材に必要な能力は」というIPAのアンケート調査「IT人材白書2016」(資料2)では「顧客分析・企画力」「データ解析」「ソフトウェア工学」「ネットワーク技術」「データベース技術」「情報セキュリティ」があげられる。



(資料2)
『IoT人材に必要な技術力とは？』
(IPA「IT人材白書2016」)

3.実施組織



4.高度IT 技術を活用したビジネス創造プログラムの構成

■各講座の時間割

コース名	講義	演習	小計
1. オリエンテーション	2:00	-	2:00
2. デザイン思考講座	4:00	6:00	10:00
3. 仮想化講座	4:00	4:00	8:00
4. ビッグデータ講座	7:10	7:50	15:00
5. AI基礎講座	8:35	7:25	16:00
6. IoT活用講座	8:50	7:10	16:00
7. セキュリティ講座	6:30	4:30	11:00
8. アジャイル講座	5:40	6:20	12:00
9. 顧客分析・企画力養成講座	4:50	13:10	18:00
10. フィールドワーク (セキュリティ・アジャイル開発・AI)	-	12:00	12:00

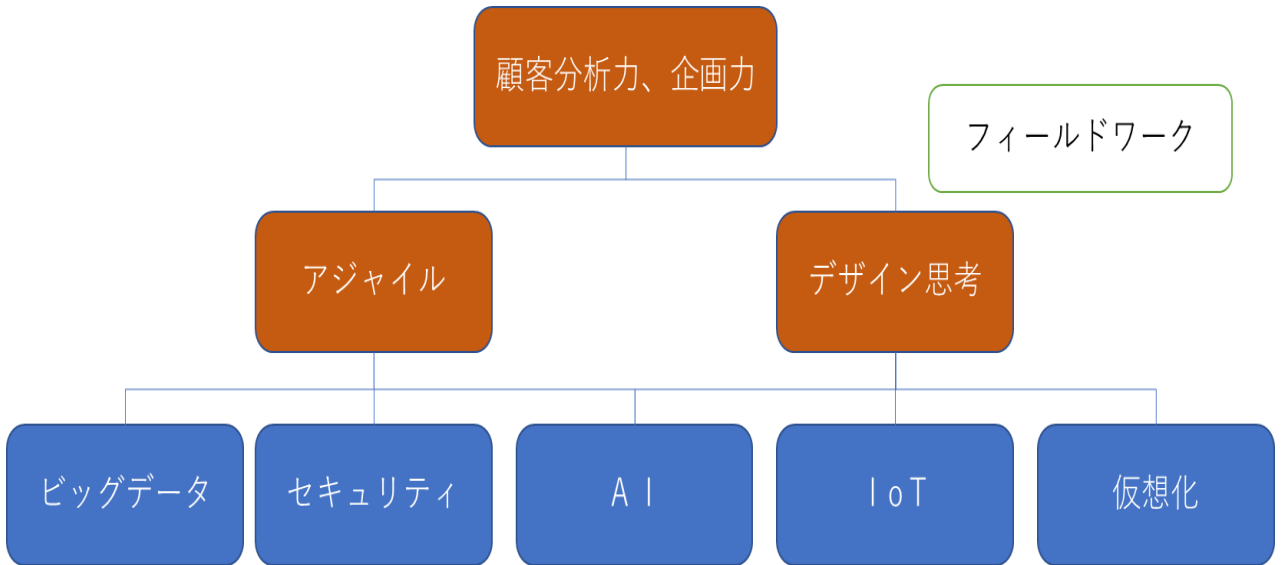
■フィールド・ワーク

	関連講座	訪問先企業 (予定)	学習概要
1	セキュリティ	株式会社ラック様	場所：本社ビル 永田町 内容：1.ラック様のセキュリティ業務の実際 2.JSOCS見学 3.体験ゲーム
2	アジャイル開発	KDDI株式会社様	場所：新宿 内容：1.開発現場の見学 2.KDDI様のアジャイルへの取り組み 3.体感ゲーム
3	Ai基礎	日本電気株式会社様	場所：三田 内容：1. NEC Future Creation Hub 見学 2:AI企画の概要と演習説明 3:【演習】データ・バリューチェーンの作成 4:まとめ

5. 高度IT 技術を活用したビジネス創造プログラムの構成

講座の関係図

- ・ 赤枠：思考法・発想法
- ・ 青枠：IT技術知識・スキル



6. 教育訓練プログラム受講修了及び評価

■ 受講修了条件

● 終了時間：120 時間

● 終了時の能力像：「第4 次産業革命において、IT 系の必須技術を駆使し、新たな発想を持ってビジネスを創造できる知識・スキル」

【各講座の時間数】

No	講座名 (モジュール名)	時間	No	講座名 (モジュール名)	時間
1	オリエンテーション	2	6	IoT活用講座	16
2	デザイン思考講座	10	7	セキュリティ講座	11
3	仮想化講座	8	8	アジャイル講座	12
4	ビッグデータ講座	15	9	顧客分析・企画力養成講座	18
5	AI基礎講座	16	10	フィールドワーク	12
総合計					120

■ 評価指標と基準点

下記の指標に対する基準点をすべてクリアする。

	実施タイミング	指標	合格基準
1	—	出席率	80%以上
2	各講座	e-learning テスト	100%
3	各講座	理解度確認テスト	80%
4	各講座	ルーブリック仕様のアンケート	～ができる(下記に例示)
5	各講座	成果物評価	講師による評価：講師が正常稼働及び理解度を前提に総合的な判定を行う

ルーブリック仕様のアンケート (例)

自己評価		合格基準		
		1	2	3
理解度	内容を理解している	メソッドのメリット・デメリットを理解している。	メリット・デメリットをメンバーに説明できる。	メリット・デメリットについて自分の考えを説明できる
応用力	現場で活用できる	状況に応じてメリット・デメリットを説明できる。	状況にあったメソッドを選択できる。	状況にあったメソッドで課題解決に取り組める。

7.仮想化講座 概要

ねらい	ハイパーバイザ型仮想化とコンテナ型仮想化、両者の違いを学び、用途に応じた仮想化のタイプ、サービスを選択・構築できるようになる。				
開催日程	8 時間 (e-learning 2 時間 含む)				
受講条件	IT 技術者としての経験が 3 年以上、ICT の基礎知識を持っていること				
学習目標	各種仮想化の実装方法を理解し、特徴に応じたシステム構築、サービス選択が出来るようになる。				
	時間	講義	演習	学習概要	学習詳細
カリキュラム 概要	40	40	0:00	仮想化概要	<ul style="list-style-type: none"> ・仮想化の種類 ・仮想化製品 ・サーバ仮想化 ・デスクトップ仮想化 ・仮想化とクラウド ・ネットワークとは ・LAN ・WAN ・IP アドレス ・ネットワーク部とホスト部 ・IP アドレスのクラス
	0:50	0:20	0:30	仮想環境の構成要素	<ul style="list-style-type: none"> ・仮想化のコンポーネント ・仮想マシン ・仮想 CPU ・仮想メモ ・仮想 HBA, 仮想ディスク ・仮想 NIC ・仮想スイッチ、仮想ルータ ・演習
	0:50	0:20	0:30	仮想ネットワーク	<ul style="list-style-type: none"> ・仮想ネットワークの概要 ・ネットワークと VLAN ・仮想 NIC とネットワーク形態 ・仮想ネットワーク Host-only ・仮想ネットワーク NAT ・仮想ネットワーク Bridge ・演習

	1:20	0:20	1:00	仮想環境の運用	<ul style="list-style-type: none"> ・仮想環境の運用 ・様々な構築支援ツール ・Vagrant の概要 ・運用とバックアップ ・移行（マイグレーション） ・P2V マイグレーション ・V2V マイグレーション ・ライブマイグレーション ・Nested VM ・演習
	50	0:20	0:30	コンテナの概要	<ul style="list-style-type: none"> ・コンテナとは ・コンテナの動作 ・仮想化との違い ・Docker の概要 ・コンテナの概要 ・Docker Hub ・Docker と OS ・コンテナの用途 ・コンテナのサイズ ・Docker のプロビジョニング
	1:30	0:00	1:30	コンテナの実践	<ul style="list-style-type: none"> ・演習
合計時間	6:00	2:00	4:00		

8.仮想化講座詳細カリキュラム

時間	学習項目	学習項目の狙い	詳細内容
0:40	仮想化概要	<p>目的： 仮想化の概要について理解する。</p> <p>ネットワークの基礎知識を確認する。</p>	<p>【講義】</p> <p>①オリエンテーション</p> <p>1. 講師自己紹介</p> <p>2. コース全体の目的に関する説明 仮想化技術の概要について説明を行う。仮想化技術を習得する上で必要となる基礎知識を概要で「仮想化の概念、仮想化の種類、クラウド、ネットワーク（LAN、WAN、IP アドレス）について習得した後、仮想環境構築支援ツールの利用方法や、コンテナ技術について学び、非仮想環境で同様の作業をした際の差異を把握し、仮想化運用した際の利点を学ぶ。</p> <p>3. e-learning の仮想化概要について簡単に振り返り確認を行う。</p> <p>3. 注意点</p> <ul style="list-style-type: none"> ・すべての技術背景をこの研修で取り扱うのは不可能 ・研修ではコア技術を集中的に実施 ・試験が e-learning で 5 問 研修 10 問 <p>4. 配布資料の確認</p> <p>【演習】 なし</p>
0:20	仮想環境の構成要素	<p>目的： 仮想化の各コンポーネントについて説明をおこなう。</p> <p>ゴール： 実際に仮想マシンの構築を行い、仮想環境に関する理解を深める。</p>	<p>【講義】</p> <p>①仮想化の各構成要素に関する知識を習得する。</p> <ul style="list-style-type: none"> ・仮想マシン ・仮想 CPU ・仮想メモリ ・仮想ホストバスアダプタ（HBA） ・仮想ディスク ・仮想ネットワークスイッチ（NIC） ・仮想スイッチ

時間	学習項目	学習項目の狙い	詳細内容
			②物理コンポーネントと仮想コンポーネントのメリット・デメリットについて個人でまとめる。
0:30			【演習】 実際に仮想環境を導入し、各種設定を行う。 ① VirtualBox による VM 作成 ・VirtualBox のインストール ・拡張パックのインストール
0:20	仮想化ネットワーク	目的： 仮想化ネットワークの設計について説明を行う。 ゴール： ハイパーバイザによって、様々なトポロジ構成が可能であることを理解し、通信可能なネットワークについて把握する。	【講義】 ① 仮想化特有のネットワーク構造について理解する。 ・ハイパーバイザによって、仮想的なネットワークが作成可能 ・ハイパーバイザによってネットワークを独立させたり、様々なトポロジ構成が可能なることを知る ② VLAN について理解する。 ・データリンク層ではネットワーク層を本来分割することが不可能だが、VLAN 対応のスイッチを利用することでネットワークの分割が可能となる。 ・同じスイッチ配下であっても複数のネットワークを束ねることができることを学ぶ。 ③ 仮想 NIC の動作モードについて把握する。 ・Host-only ・NAT ・Bridge
0:30			【演習】 ① Hyper-V マネージャを操作し、仮想ネットワークの設定変更を理解する。 ・Hyper-V マネージャでは NAT に関する設定が GUI から操作不可能なため、Host-only と Bridge について、それぞれの違いを学習する。 ・Host-only では仮想マシンの中だけでネットワークが閉じられており、外部との通信が不可能である。完全な閉域ネットワークを利用

時間	学習項目	学習項目の狙い	詳細内容
			<p>する場合に有効であることを把握する。</p> <ul style="list-style-type: none"> ・Bridge はハイパーバイザが仮想マシンと物理マシンとの通信を直接行うことが可能である。仮想マシン→ハイパーバイザ、あるいはその逆の疎通確認や、他の受講者との通信の疎通確認を行う。
0:20	仮想環境の運用	<p>目的： 自動構築ツール Vagrant について学び、Vagrant Cloud から構築する方法を実習で行う。</p> <p>ゴール： 仮想環境自動構築ツールの利用方法について学び、設定ファイル等の役割を理解した上で設定の調整が可能になる。</p>	<p>【講義】</p> <p>①仮想環境の運用について理解する。</p> <ul style="list-style-type: none"> ・実際の仮想環境構築のフローを改めておさえる。 ・手動での煩雑な設定作業を避けるために仮想環境の自動構築ツールがあることを知る。 ・構築支援ツールのひとつである Vagrant について学ぶ。 <p>②Vagrant の設定ファイル (Vagrantfile) の設定方法について学ぶ。</p> <p>③Vagrant Cloud から仮想イメージ (Box) を取得できることを学ぶ。</p> <p>④仮想マシンの移行作業 (マイグレーション) について学ぶ。</p> <ul style="list-style-type: none"> ・P2V マイグレーション ・V2V マイグレーション ・ライブマイグレーション <p>⑤Nested VM (仮想マシン上での仮想マシン) について学ぶ。</p>
1:00			<p>【演習】</p> <p>①仮想環境構築支援ツールである Vagrant を利用して、仮想マシン構築の自動設定を行う。</p> <ul style="list-style-type: none"> ・Windows Server 上で設定を行うにあたり、コマンドプロンプト (管理者権限) で操作を行う。 ・Vagrant の設定ファイルである Vagrantfile に関する設定項目および起動方法について学び、演習を行う。
0:20	コンテナの概要	<p>目的： コンテナシステムの概要を説明する。</p>	<p>【講義】</p> <p>①コンテナシステムの概要を知る。</p> <ul style="list-style-type: none"> ・コンテナ (アプリケーションが分離された空間) は、プロセスのよう

時間	学習項目	学習項目の狙い	詳細内容
		<p>ゴール： Docker コマンドの利用 や Docker Hub へのア ップロードを通じ、コンテ ナシステムでのやりとりの 簡便さ把握し、簡単な やりとりが可能になる。</p>	<p>に、分離が可能であることを学ぶ。</p> <ul style="list-style-type: none"> ・ホスト名、ファイルシステムなどがコンテナ独自に設定可能であること <p>②コンテナエンジン Docker の概要について学ぶ。</p> <p>③先述の仮想マシンとの違いについて学ぶ。</p> <ul style="list-style-type: none"> ・仮想マシンはゲスト OS の容量が必要、HW のオーバーヘッド大 ・コンテナは共通のホスト OS のみで済む ・仮想マシン内アプリケーションはすべて同一ネットワーク ・コンテナはそれぞれ隔離され、ネットワークも独立可能 <p>④Docker Hub の概要について学ぶ。</p> <ul style="list-style-type: none"> ・コンテナイメージが Docker Hub に登録されており、自由にダウンロード可能であること ・自作したコンテナイメージを Docker Hub に登録可能なこと <p>⑤コンテナの具体的な用途について把握する。</p> <ul style="list-style-type: none"> ・起動速度についてや分離デーモンプロセスの複数起動など、仮想マシンのみでの実施に比べて、大掛かりなことが容易に実施可能で、負荷が軽い。 <p>注意点</p> <ul style="list-style-type: none"> ・コンテナシステムは仮想化と混同しやすいため、コンテナと仮想化の類似点と相違点について受講者にわかりやすく伝える必要がある。
0:30			<p>[演習]</p> <p>①コンテナエンジンである Docker を利用してコンテナの操作演習を行う。</p> <ul style="list-style-type: none"> ・Docker のインストールを行う。仮想マシン上に作成した Ubuntu Desktop 上にインストールする。 ・hello-world や bash といった簡単なコンテナを呼び出し、Docker のコマンドに慣れる。 ・Docker コマンドで Ubuntu イメージを起動する。 ・コンテナの状況確認や削除を実施する。

時間	学習項目	学習項目の狙い	詳細内容
			<ul style="list-style-type: none"> ・シェルスクリプトとの組み合わせで複数個のプロセスを同時に呼び出せる命令について実践をおこなう。 ②Docker Hub に接続し、コンテナイメージのやりとりを行う。 ・予め準備した Docker Hub のアカウントを利用して、Docker Hub に自身が作成したコンテナイメージをアップロードする。 ・他の受講者がアップロードしたコンテナイメージを呼び出す実践を行う。
1:30	仮想化技術およびコンテナの実践、総合演習	<p>目的： 仮想化技術およびコンテナシステムを実際にインストール、動作確認をおこなう。</p> <p>ゴール： 仮想化技術およびコンテナシステムの利用場面について考察し、実システムを想定した簡単な業務システムの構築をおこなうことができる。</p>	<p>【演習】 これまで学んだ仮想化技術およびコンテナシステムの概念および実践方法について理解し、自分の業務に関係するサービスやシステムについて、改善可能なシステム導入を実践する。</p> <p>注意点</p> <ul style="list-style-type: none"> ・規模が大きなものを設計・実施すると学習時間が足りなくなるため、身近でかつ規模がそれほど大きくないもので実施できるようにアドバイスを行う。 ・特に、仮想環境とコンテナシステムのどちらを使用するか、あるいは両方使用するか、それぞれのメリット・デメリットを見極めた上で「現状の不便なところがこうすると改善される」といった身近なところに何か改善点がないかアドバイスを行う。 ・コンテナのテンプレートや仮想マシンテンプレートのライセンスに注意する。

仮想化

仮想化は、1章から3章までがE-Learning対象です。

講座では1章から3章までは復習として簡単に触れますが、メインは4章以降です。また、2章のIPアドレスについては、もう少し詳細に講座内で解説します。

目次

第1章 仮想化概要	
1-1. 仮想化とは	9
1-2. 仮想化の種類	11
1-3. 仮想化製品	12
1-4. サーバ仮想化	13
1-5. デスクトップ仮想化	14
1-6. 仮想化とクラウド	15

目次

第2章 仮想化概要	
2-1. ネットワークとは	16
2-2. LAN	17
2-3. WAN	18
2-4. IPアドレス	19
2-5. ネットワーク部とホスト部	22
2-6. IPアドレスのクラス	24

目次

第3章 仮想環境の構成要素

3-1. 仮想化のコンポーネント	27
3-2. 仮想マシン	28
3-3. 仮想CPU	29
3-4. 仮想メモリ	32
3-5. 仮想HBA, 仮想ディスク	33
3-6. 仮想NIC	34
3-7. 仮想スイッチ、仮想ルータ	35
3-8. 演習	36

目次

第4章 仮想ネットワーク	
4-1. 仮想ネットワークの概要	38
4-2. ネットワークとVLAN	39
4-3. 仮想NICとネットワーク形態	40
4-4. 仮想ネットワーク Host-only	41
4-5. 仮想ネットワーク NAT	42
4-6. 仮想ネットワーク Bridge	43

目次

第5章 仮想環境の運用	
5-1. 仮想環境の運用	45
5-2. 様々な構築支援ツール	46
5-3. Vagrantの概要	47
5-4. 運用とバックアップ	49
5-5. 移行（マイグレーション）	51
5-6. P2Vマイグレーション	52
5-7. V2Vマイグレーション	53
5-8. ライブマイグレーション	54
5-9. Nested VM	55
5-10. 演習	56

目次

第6章 コンテナの概要	
6-1. コンテナとは	58
6-2. コンテナの動作	59
6-3. 仮想化との違い	60
6-4. Dockerの概要	61
6-5. コンテナの概要	62
6-6. Docker Hub	63
6-7. DockerとOS	64
6-8. コンテナの用途	65
6-9. コンテナのサイズ	66
6-10. Dockerのプロビジョニング	67
第7章 コンテナの実践	
7-1. 演習	69

第1章 仮想化概要

仮想化の基本を学ぶ

1章, 2章はe-Learningとしても提供するが、講座では一通り流して説明する。

1-1. 仮想化とは

プロセッサやメモリ、ディスク、通信回線など、コンピュータシステムを構成する資源（リソース）を、物理的構成に拠らず柔軟に分割したり統合したりすること

1台のサーバコンピュータをあたかも複数台のコンピュータであるかのように論理的に分割し、それぞれに別のOSやアプリケーションソフトを動作させる「サーバ仮想化」や、複数のディスクをあたかも1台のディスクであるかのように扱い、大容量のデータを一括して保存したり耐障害性を高めたりする「ストレージ仮想化」などの技術がある

IT用語辞典より転載

・説明の流れ

プロセッサやメモリ、ディスク、通信回線など、コンピュータシステムを構成する資源（リソース）を、物理的構成に拠らず柔軟に分割したり統合したりすること

・ポイント

仮想化は従来の物理環境とはまったく異なるということ

1-2. 仮想化の種類

仮想化は次の種類に分けられる

- サーバ仮想化
- デスクトップ仮想化
- ネットワーク仮想化
- ストレージ仮想化

・ポイント

仮想化の種類によって大きく4タイプに分かれるということ

・補足説明

サーバ仮想化：1台のコンピュータをリソース分割し、複数のコンピュータとして振る舞うこと。様々なOSやアプリケーションを同時に実行できる。需要に応じてプロセッサやメモリ、ストレージなどのリソースを自在に変更できる。

デスクトップ仮想化：デスクトップ上に他のOS環境を作成し、同時に実行できる。主に実験環境や一時的な開発環境の構築に用いられる。

ネットワーク仮想化：ネットワークを物理的な配線に縛られず、ソフトウェアによって自由に変更できる。近年はネットワーク仮想化のOpenFlowプロトコルに対応した安価な製品が多く販売されている

。

ストレージ仮想化：サーバ仮想化と同じく、ストレージを論理分割してあたかも複数のディスクや異なった種類のストレージのように振る舞うことができる。また、他の仮想化と同じく、需要に応じて自由にリソースの変更も可能。

1-3. 仮想化製品

代表的な製品 (ハイパーバイザ)

- サーバ仮想化 (Type-I)
 - VMware Sphere
 - Microsoft Hyper-V
 - Xen
- デスクトップ仮想化 (Type-II)
 - VMware Workstation, Player, Fusion
 - Microsoft Hyper-V
 - Virtual Box

・ポイント

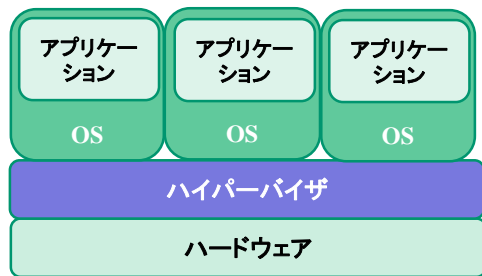
具体的な商品・サービスをここで紹介

・質問

これらの製品を使ったことがありますか？ また、名前を知っていますか？ と問いかけ。

1-4. サーバ仮想化

Type-I型、ネイティブ型、ベアメタル型
ハードウェア上で直接動作するハイパーバイザ
OSはすべてハイパーバイザ上で動作し、ほぼネイティブ環境に近い動作速度が得られる



・説明の流れ

図を示し、ハードウェア上で複数のシステムが動作することを中心に説明。

・補足説明

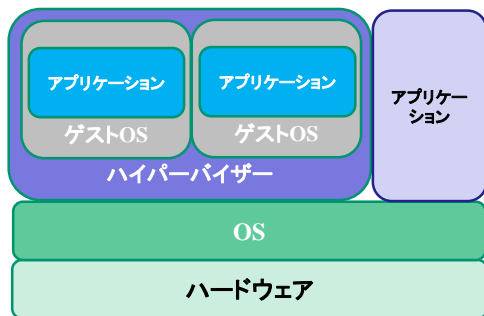
サーバ仮想化は、Type-I型、ネイティブ型、ベアメタル型、とも呼ばれる

1-5. デスクトップ仮想化

Type-II、ホストOS型

ハードウェア上にまずOSが動作し、その上で1アプリケーションとしてハイパーバイザーが動作

ハイパーバイザー上で仮想環境が動作





・補足説明

デスクトップ仮想化は、Type-II、ホストOS型、とも呼ばれる
Windowsを普通に使用しているときに平行して他OSを使用できる
など、具体例を示す。

1-6. 仮想化とクラウド

クラウドサービスでは仮想化環境を提供

クラウドサービスはユーザが自由にリソースを変更可能。仮想化環境がマッチしている

	suse-sles-12-sp3-v20171212-hvm-ssd-x86_64 - ami-8368cefb SUSE Linux Enterprise Server 12 SP3 (HVM, 64-bit, SSD-Backed) ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	RHEL-7.4_HVM_GA-20170808-x86_64-2-Hourly2-GP2 - ami-9fa343e7 Provided by Red Hat, Inc. ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20171121.1 - ami-0def3275 Canonical, Ubuntu, 16.04 LTS, amd64 xenial image build on 2017-11-21 ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	Windows_Server-2016-English-Full-Base-2017.11.29 - ami-f6d8008e Microsoft Windows Server 2016 with Desktop Experience Locale English AMI provided by Amazon ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい

・ポイント

クラウドサービスの基盤上で仮想化が使用されていることを説明。

ここで示している図は、AmazonのクラウドサービスAmazon EC2でテンプレートとして用意されている仮想マシンの一例。

第2章 ネットワークの基本

仮想化技術を習得する上で最低限
の知識

2-1. ネットワークとは

ネットワークとは、網という意味の英単語。複数の要素が互いに接続された網状の構造体のこと。ネットワークを構成する各要素のことを「ノード」(node)、ノード間の繋がりのことを「リンク」(link)あるいは「エッジ」(edge)と言う。

一般の外来語としては人間関係の広がりのことや、組織や集団の構造などを指すこともあるが、IT関連の分野で単にネットワークという場合は、複数のコンピュータや電子機器などを繋いで信号やデータ、情報をやりとりすることができるコンピュータネットワークあるいは通信ネットワークのことを意味することが多い。

IT用語辞典より転載

・ポイント

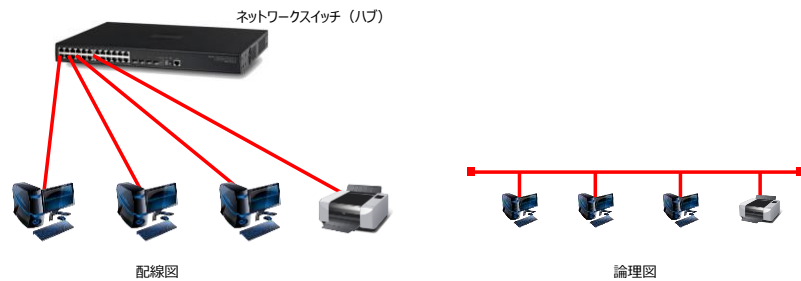
仮想化はネットワークと密接に関わってきます。

ここから先はネットワークについて少し説明しますが、ノード、リンク、エッジ、またホストなどの用語をしっかりとここで押さえてください。

2-2. LAN

LAN (Local Area Network)

- 1つのフロア, 組織, 部署といった, 比較的狭い範囲でのネットワーク



・説明の流れ

おそらくLANという言葉聞いたことのない方はいないと思いますが、社内LANとか言いますよね、社内LANありますよね、と問いかけて説明してください。

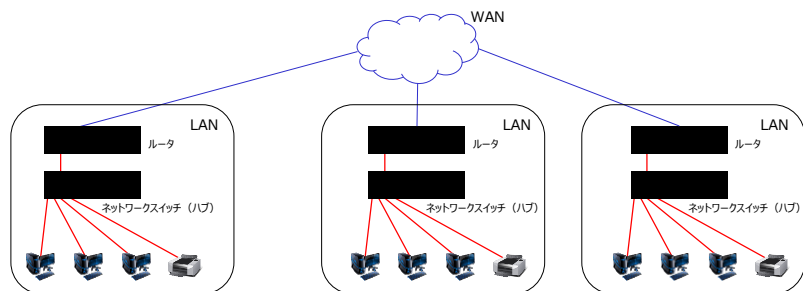
・ポイント

LANとWANはよく対比して説明されます。次のページのWANとともに受講者に説明してください。

2-3. WAN

WAN (Wide Area Network)

- 広域通信網（Wide Area Network）の略。
LANとLANを結ぶ公衆網のことを指すことが多い。
WANを世界規模で実現しているのがインターネット。



・ポイント

LANとWANはよく対比して説明されます。前のページのLANとともに受講者に説明してください。

2-4. IPアドレス

- ネットワーク上で機器の識別をするための番号
- 32bit でIPアドレスを管理 (IPv4, Internet Protocol Version 4)
 - bit … 情報量の最小単位, 1 or 0
- ネットワーク部とホスト部の存在

192.168.1.10

10進 → 2進に変換

$$192_{(10)} = \boxed{1100\ 0000}_{(2)}$$

$$168_{(10)} = \boxed{1010\ 1000}_{(2)}$$

$$1_{(10)} = \boxed{0000\ 0001}_{(2)}$$

$$10_{(10)} = \boxed{0000\ 1010}_{(2)}$$

・説明の流れ

IPアドレスは聞いたことありますか、と問いかけ、そこから識別のためのもの、ユニークなもの、ネットワーク部、ホスト部、と分かれることを説明してください。

IPアドレス

192.168.1.10

1100 0000 1010 1000 0000 0001 0000 1010

2進数の並びが $8 \times 4 = 32$ ｺ



32bit

※ 8bitのかたまり = 1オクテットと呼ぶことも octet

・ポイント

IPアドレスは2進数で扱くと計算が楽なので、2進数表記もときおり使用されます。

・補足説明

2進数になじみのない方がいる場合は、2進数の説明を少しした方がよいでしょう。

192.168を「いちくに いちろっぱ」とよく呼ぶので、そのことも合わせて補足説明した方が演習等でのアドレスの読み方も参考になります。

IPアドレス (続)

- 1オクテットの範囲

0000 0000 すべて 0 ~ 1111 1111 すべて 1

2進 → 10進に変換

0000 0000₍₂₎ = 0₍₁₀₎

1111 1111₍₂₎ = 255₍₁₀₎

- ※ あくまでも1オクテットで表現できる範囲を10進法で表しているだけ。
- ※ 3桁だからといって、999.999.999.999 のような値を表現できるわけではない。

・説明の流れ

このスライド含め、3枚はIPアドレスの基本的事項の確認です。
やや時間をかけて説明してください。

2-5. ネットワーク部とホスト部

192.168.1.10/24

この例だと、IPアドレスのビット列の、

先頭から24bit = ネットワーク部

残りの8bit = ホスト部

1100 0000 1010 1000 0000 0001 0000 1010

← 24bit : ネットワーク部

8bit : ホスト部 →

ネットワークプレフィックスと言うことも、
network prefix

ネットワーク内の端末を
識別

端末がどのネットワークに所属しているのかを判別。

・説明の流れ

色を付けてわかりやすく分離していますが、/を付けて、ネットワーク部のビット数を表記することをプレフィックス表記と呼びます。

ここでは頭から24bit分がネットワーク部なので、/24となります。

ひとつのネットワークに収容できるホスト数

ホスト部がすべて 0 ⇨ ネットワーク自身のことを表す

ホスト部がすべて 1 ⇨ そのネットワーク全体にデータを送る
ブロードキャストアドレスを表す

192.168.1.10 ならば…

ネットワーク自身 ⇨ **192.168.1.0**

1100 0000 1010 1000 0000 0001 0000 0000
すべて 0

ブロードキャストアドレス ⇨ **192.168.1.255**

1100 0000 1010 1000 0000 0001 1111 1111
すべて 1

・ポイント

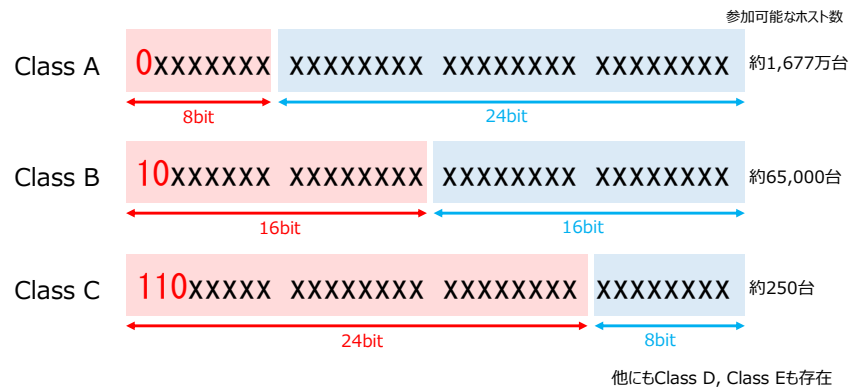
ネットワーク内では、すべてのIPアドレスがホストに割り当てられるわけではありません。

スライドにも書いてあるように、ネットワークの一番最初のアドレスはそのネットワーク自身を指すネットワークアドレス、一番最後は、ネットワーク全体にデータを送信するブロードキャストアドレスとして使用されるため、全アドレス数から-2の数がホストへの割り当て数になります。この例では、192.168.1.1から192.168.1.254までをホストに割り当てられます。

また、192.168.1.10の属するネットワークは、192.168.1.0になります。（ネットワークアドレスはこのようにして使用します）

2-6. IPアドレスのクラス

- 32bit中, 「何bitがネットワーク部で, 何bitがホスト部を表すか」
- ネットワークの規模を識別する



・説明の流れ

IPアドレスのクラスによって、ネットワーク部、ホスト部が異なることを説明し、その結果クラスによって何がかわるのかを説明します。

ホスト部を広くとれるということは、すなわちネットワーク内のホストをより多く収容できることを併せて説明してください。

IPアドレスのクラス

(例題) 次のIPアドレスのクラス (ネットワークの規模) は?

172.16.0.1

10進 → 2進に変換

$$172_{(10)} = \boxed{1010\ 1100}_{(2)}$$

先頭ビットが 10... なので, 「**クラスB**」であることがわかる

・ポイント

10進・2進変換をここで実際に手計算し、先頭2ビットを見てクラスBであることを再確認してください。

第3章 仮想環境の構成要素

主要コンポーネントについて理解
する

3-1. 仮想化のコンポーネント

主なコンポーネント

- 仮想マシン
- 仮想CPU
- 仮想メモリ
- 仮想HBA
- 仮想ディスク
- 仮想NIC
- 仮想スイッチ

・ポイント

ここに挙げたコンポーネントはすべて物理的なリソースです。

仮想化とは、物理リソースを論理化するものだというをここで再確認してください。

3-2. 仮想マシン

物理的なPC,サーバをハイパーバイザによって仮想的な機械に置き換えたもの
Virtual Machine : VM
構成ファイル、仮想ストレージなど構成される

Hyper-Vの例

📁 ubuntu14d	2017/11/29 22:41	ファイル フォルダ	
📄 ubuntu14-desktop.vhdx	2017/12/19 14:29	ハードディスク イメ...	9,441,280 KB

↑
仮想マシン(ハードディスクイメージ)

・補足説明

この図で示しているように、仮想マシンは単一、あるいは複数のファイルになり、それがイメージになります。

この例はHyper-Vのvhdx形式ですが、ハイパーバイザごとにフォーマットがあることも触れてください。

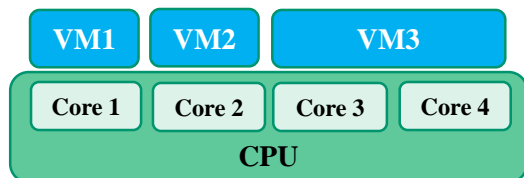
3-3. 仮想CPU

仮想マシンからは仮想CPUが物理CPUとして見える

仮想マシン1つにつき1つのCPU、もしくは1つのコアを割り振るのが理想的

重い仮想マシン（VM3）には多くのコアを割り振り、軽い仮想マシン（VM1,VM2）には少ないコアを割り振る

商用のハイパーバイザの中には、CPUのコア数、仮想CPUの総コア数でライセンスが決まるものもある



最近のCPUはデュアルコア、クアッドコア、さらにハイパースレッディングによって、1CPUでも論理コア数が4や8といった場合が多いので、物理CPU数とコア数は一致しないことに注意。

3-3. 仮想CPUと仮想化支援機能

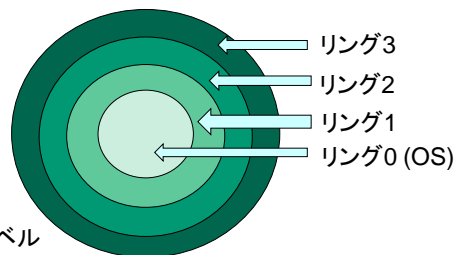
近年のCPUには、仮想CPUを物理CPUとほぼ同等に動作させる仮想化支援機能がある

Intel CPU: VT-x, AMD CPU:AMD-V

CPUの特権レベルが最も高いリング0にアクセスできるのはOSのみ

アプリケーションやハイパーバイザはリング3で動作

リング3からリング0へのアクセスは例外が発生する



CPUの特権レベル

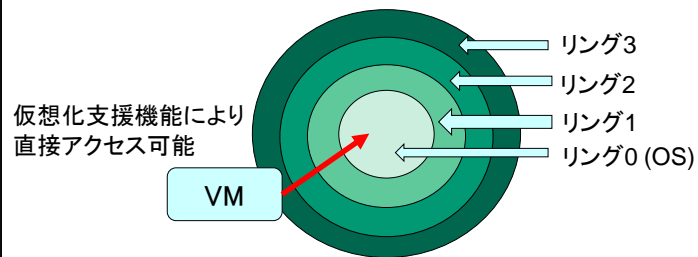
・質問

VT-xやAMD-Vという機能を聞いたことがありますか？と問いかけ。

3-3. 仮想CPUと仮想化支援機能

VT-xやAMD-Vに対応したハイパーバイザではVMからリング0へ例外なしにアクセス可能

I/Oもほぼ直接アクセスできるため、物理環境に近い速度で動作
デフォルトで仮想化支援機能が無効になっているPCがあるので注意



・ポイント

仮想CPUは物理CPUとは異なるため、ネイティブの速度は出ないが、支援機能によって近い速度が出ることがポイントです。

・補足説明

最近のハイパーバイザは、仮想化支援機能が有効になっていないとインストールできなかつたり起動できなかつたりします。このことも併せて説明してください。

有効になっていない場合は、BIOSやUEFIから有効にします。

3-4. 仮想メモリ

仮想マシンの動作には当然メモリが必要

ハイパーバイザによって最大メモリ量が異なる

仮想マシンへの仮想メモリとして割り当てる際に、完全固定として割り当てる方法（スタティック）と、最小容量と最大容量を定めておき、動的に変更できる方法がある

Type-I ハイパーバイザ：仮想マシンがほぼすべてのメモリを使用可能

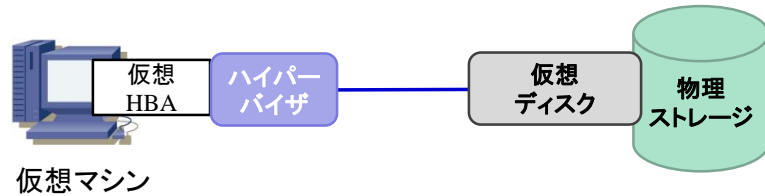
Type-II ハイパーバイザ：他アプリケーションが使用するメモリ容量を考慮して仮想メモリを設計する

・ポイント

仮想化のタイプによって、仮想メモリがどれくらい使用できるかが異なります。

3-5. 仮想HBA, 仮想ディスク

仮想マシンは仮想ディスクをストレージとして使用する
仮想マシンには、ホストバスアダプタ（HBA）としてIDEやSCSIのインターフェイスが接続され、各方式の内蔵ディスクとして見える
仮想ディスクは動的構成を取ることができる
設定上120Gの仮想ディスクを使用しても、実際に使用しているサイズのみ物理ストレージ上では消費する



・補足説明

HBAは仮想化関連でしかあまり出てこない用語です。

聞き慣れない人が多いと思いますが、スライドにあるようにPCのストレージインターフェイスのことだと説明すれば理解が早いでしょう。

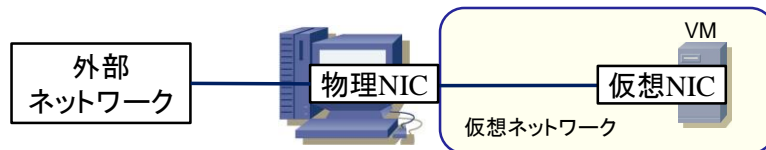
また、仮想ディスクは設定上20Gとしても、動的構成ならば実際に使用しているサイズしか使用しないこともポイントです。

3-6. 仮想NIC

ハイパーバイザによって仮想的なインターフェイスである仮想NICが作成され、仮想ネットワークも作成される

物理NICと仮想NICを接続しないと、ホストOSと仮想OSは通信できない

物理NICと仮想NICの間に仮想ブリッジや仮想ルータを挟むことで、仮想ネットワーク形態が変わる



・説明の流れ

今は無線が多いので、有線ネットワークの口を見ることがあまりないですが〜と物理NICのことをまず説明し、仮想ネットワークへの接続のために仮想NICが必要なことを説明する。

詳細は4章で、と前置きをして、仮想ブリッジや仮想ルータも使用することを触れる。

3-7. 仮想スイッチ、仮想ルータ

仮想スイッチ : Virtual Switch (vSwitch)



仮想ルータ : Virtual Router (vRouter)



仮想ネットワークにおいて、それぞれ物理機器と同じ役割を果たす
仮想ネットワークをNATにする場合はvRouterが
VLANを作成する場合はvSwitchが必要

・質問

スイッチとルータの違いはわかりますか？ これらの機器の設定を行ったことがありますか？ と問いかけ。

ネットワーク機器について詳しい方が多ければ、4章の説明はさらっと流してよいでしょう。

・補足説明

スイッチはレイヤ2でネットワークを分割、ルータはレイヤ3でネットワークを分割しますが、ネットワークになじみのない人にはわかりにくいと思います。

IPアドレス上でのネットワーク分割はルータで行う、という事例を出しながら、少し補足説明をしてください。

3-8. 演習

演習1 : VirtualBoxによるVM作成

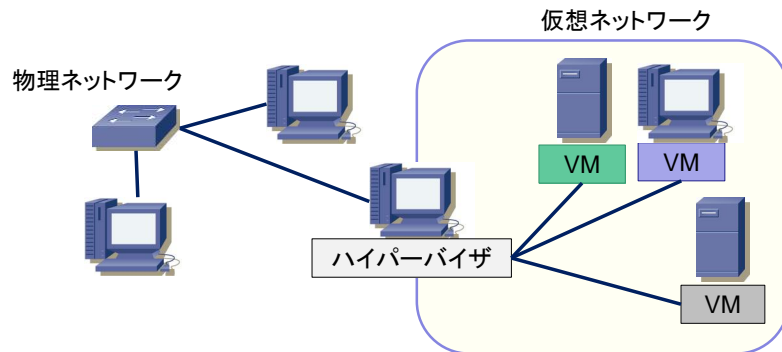
第4章 仮想化ネットワーク

仮想化特有のネットワーク構造に
ついて理解する

ここまで（1 - 3章）がE-Learningです。

4-1. 仮想ネットワークの概要

ハイパーバイザによって仮想的なネットワークの作成が可能
ハイパーバイザの設定によって、仮想ネットワークを独立させたり、物理ネットワークと接続したり、様々なトポロジ構成が可能

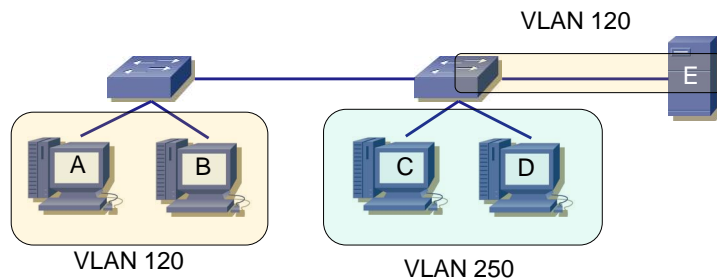


・ポイント

ここから、スライドでVM表記を多用します。仮想マシン=VMという認識を再確認してください。

4-2. ネットワークとVLAN

データリンク層（Ethernet）は本来ネットワークを分割できない
VLAN対応スイッチによって提供されるVLANによって分割可能
仮想ネットワーク内でも仮想スイッチによるVLAN分割が可能



・補足説明

必要ならばOSI 7層モデルを板書し、簡単に説明してください。

ユーザ側でVLANを意識することはあまりないのですが、自分で仮想ネットワークを構成するときを使用することがたまにあります。以下は、VLANについての詳細説明です。

VLANはVLAN対応スイッチによって論理分割されるネットワークです。スライドでも説明してあるように、本来スイッチはデータリンク層のため、レイヤ3での（IPアドレスでの）ネットワーク分割はできません。

ネットワーク分割はネットワーク層のルータによって行われますが、例外的にVLANによってネットワーク分割ができる。VLANは番号によって管理され、同じVLAN番号のホストは通信可能。異なったVLAN番号のホスト間は通信できません。

VLANを越えて通信するには、ルータによるルーティングが必要です。この例では、ホストA,B,E間はVLANが同じのため互いに通信できま

すが、ホストC,D間とはVLANが異なるため通信できません。

現在の主要なハイパーバイザは、VLAN番号を付与した通信（VLANタギング）が可能なので、VLANによるネットワーク分割をそのまま仮想ネットワークでも適用できます。

4-3. 仮想NICとネットワーク形態

ハイパーバイザによって仮想的なインターフェイスである仮想NICが作成され、仮想ネットワークも作成される

次の3形態

- Host-only
- NAT
- Bridge

・ポイント

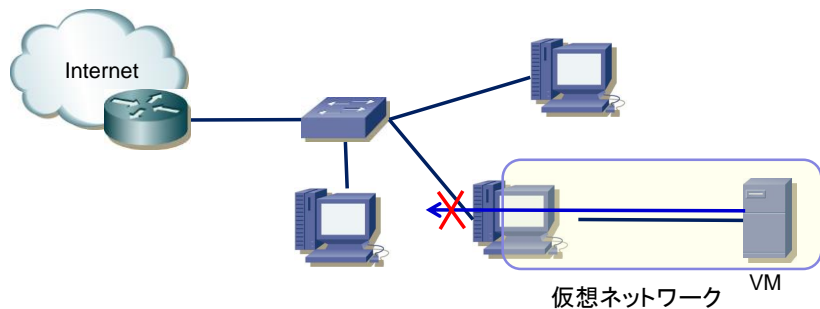
一般的な仮想ネットワークとしてこの3形態があることを押さえてください。

・補足説明

Type-I型ではブリッジが、Type-II型はNATかHost-onlyがデフォルトであることが多いです。また、ハイパーバイザによっては、この3形態すべてをサポートしていなかったり（Hyper-VはNATをオプション扱い）あるいはより細かいネットワーク形態を取っている場合もあり、必ずしもこの3形態とは限りません。

4-4. 仮想ネットワーク Host-only

ハイパーバイザが動作しているホストとのみ通信できるネットワーク
実ネットワーク内の物理マシンや外部ネットワークとは通信できない



・説明の流れ

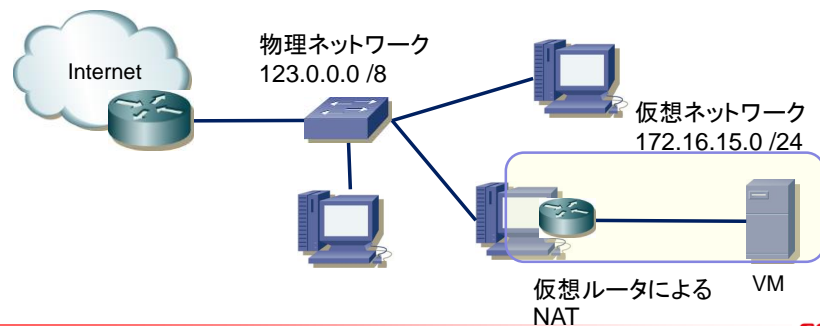
ここから、Host-only, NAT, Bridgeについて説明します。VMと物理ネットワークの接続性について着目して説明します。

・ポイント

VMはハイパーバイザが稼働しているホストとのみ通信可能です。もし複数VMが稼働していると、各VM間は通信できません。

4-5. 仮想ネットワーク NAT

仮想ネットワークにはプライベートアドレスが与えられる
ハイパーバイザが提供する仮想ルータによってアドレス変換が行われる
物理ネットワークに影響が少ない



・ポイント

VMにはプライベートアドレスが割り当てられるということ、物理ネットワークに影響が少ないことがポイントです。

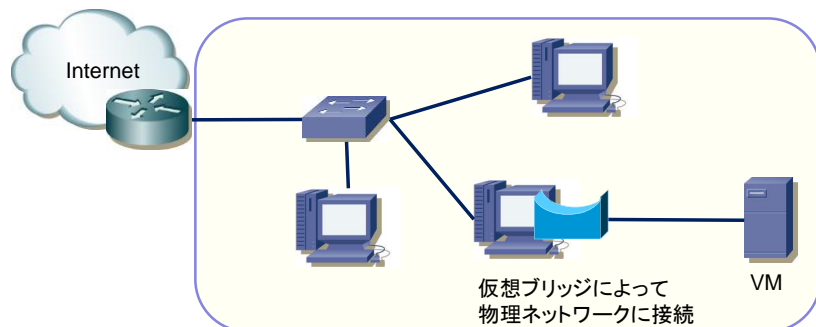
Type-II型のハイパーバイザはこの形態がデフォルトであることが多いです。

4-6. 仮想ネットワーク Bridge

ハイパーバイザーの提供する仮想ブリッジ（仮想スイッチ）を介して実ネットワークに直接接続

物理マシンと直接通信可能

物理ネットワークに影響を与えるおそれがある



・ポイント

Bridge構成にすると物理ネットワークとホストを經由して直接つながりません。

Type-I型のハイパーバイザではこの形態がデフォルトであることが多いです。

4-7. 演習

演習3：仮想ネットワーク設定

演習ではここまで紹介した仮想ネットワークを実際に体験します。

第5章 仮想環境の運用

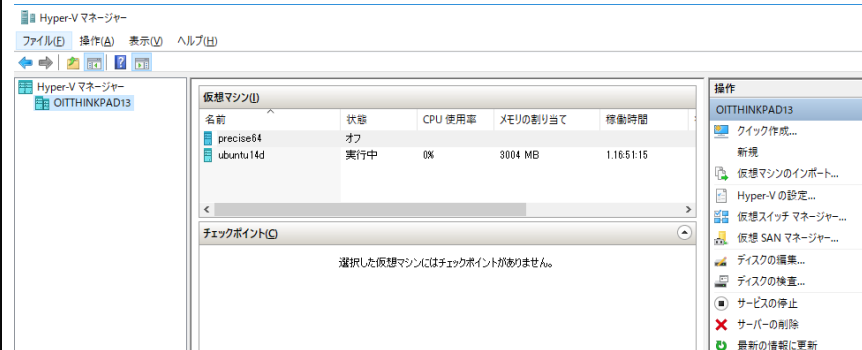
支援ツールや移行について学ぶ

5-1. 仮想環境の運用

ハイパーバイザの操作：GUI or コマンド

ハイパーバイザ独自のシェル環境など

Hyper-VはPowerShellによるコマンド、Hyper-Vマネージャ



・補足説明

Type-I型のハイパーバイザでは、多数のVMを管理することが多く、管理ツールが充実しています。

ここではHyper-Vの例を紹介していますが、VMware SphereではWebブラウザからハイパーバイザにアクセスし、様々な設定を行うことができます。

5-2. 様々な構築支援ツール

実験的に1台構築するだけでも、様々な設定を手動で行うのは煩雑
仮想環境を自動構築するための支援ツールを使用

- Vagrant : 仮想環境構築ツール
- Puppet : 構成管理ツール
- Chef : サーバ設定・更新自動化ツール
- libvirt : 仮想化環境共通API群



・説明の流れ

これらのツールを見聞きしたことはあるか、参加者に質問します。また、使ったことがあるという人がいれば、簡単に説明してもらおうのもよいでしょう。

・補足説明

ここに挙げたツールの他にもいろいろありますが、著名なものをスライドでは挙げています。

Vagrantをこの先少し紹介していますが、仮想環境を手早く構築できて大変便利なツールです。

5-3. Vagrantの概要

オープンソースの仮想環境構築ソフトウェア、MITライセンス
ファイル(Vagrantfile)に設定を記述し、仮想環境を自動的に構築
VirtualBox, VMware, Hyper-Vなどに対応
Amazon EC2といったクラウドにも対応
コンテナDockerにも対応している

Vagrantfileの例

```
config.vm.box = "generic/ubuntu1604"    #仮想イメージ:ubuntu16.04
config.vm.provider "hyperv" do |h|      #プロバイダ:Hyper-V
  h.cpus = "2"                           #仮想CPU:2
  h.maxmemory="4096"                     #最大メモリ:4096M
  h.enable_virtualization_extensions=true #仮想化支援機能:有効
end
```

・補足説明

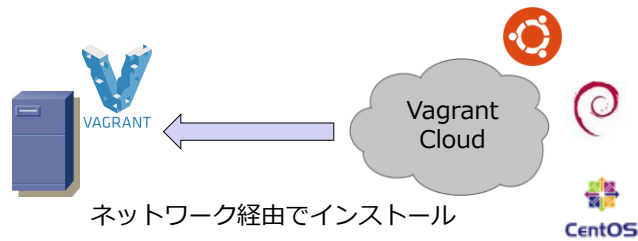
Vagrantは主要なハイパーバイザに対応しており、設定ファイルでどのハイパーバイザを使用するか指定します。

使用するハイパーバイザのことを、Vagrantではプロバイダと呼んでいて、この例ではコメントにもあるようにHyper-Vを指定しています。

Dockerはこの後6章で説明するので、コンテナとは？という疑問は一度置いておいてください。

5-3. Vagrantの概要(続き)

仮想イメージ : Box, 動作ハイパーバイザ : プロバイダ
コマンドでVagrant CloudからBoxを導入可能
<https://app.vagrantup.com/>
独自Boxの作成も可能



・補足説明

Vagrantでは導入できる仮想イメージのことをBoxと呼び、また動作対象になるハイパーバイザのことをプロバイダと呼びます。

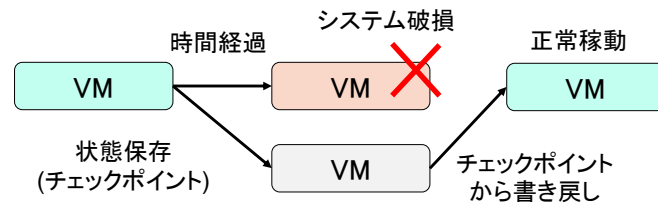
時間に余裕があれば、実際にスライドのURLにアクセスして、Box一覧を見せてください。UbuntuなどのオフィシャルBoxが多数あり、またユーザによって色々で作られたものがアップロードされています。

演習ではプロバイダをHyper-Vに設定してVagrantによる自動構築を行います。

5-4. 運用とバックアップ

仮想マシンの運用には、Vagrantなどの運用ツールの他、ハイパーバイザ用のシェル言語などもある。例) Hyper-VはPowerShell

仮想マシンなら、ファイルベースなのでバックアップも容易
Hyper-Vではある状態を保存することをチェックポイントと呼ぶ



・ポイント

VMはファイルベースなので、バックアップを容易にとれる、またハイパーバイザや管理ツールからある時点での状態（チェックポイント、スナップショット）を容易にバックアップ・復元できます。

5-5. 移行（マイグレーション）

現在の環境をそのまま仮想環境へ移す、あるいは新規に仮想マシンを作成する、など様々な移行方法がある

仮想マシンは物理マシンと完全には同じではない

ネットワークやハイパーバイザーによっては、仮想環境に対応していないOS、アプリケーション、ハードウェアがある

仮想環境、クラウドで使用するすべてのハードウェア、ソフトウェアについて対応状況をチェックする

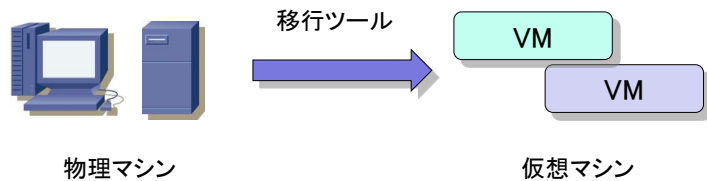
5-6. P2Vマイグレーション

Physical to Virtual : P2V

現在動作している物理マシンを仮想マシンへ移行するツールを使う

無停止で行うホットクローニング

物理マシンのシャットダウンを伴うコールドクローニング

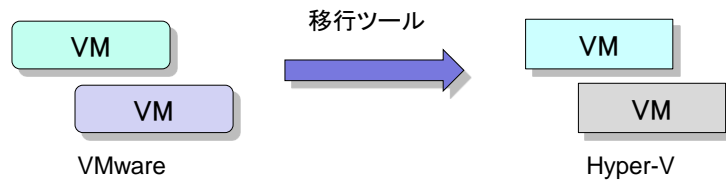


5-7. V2Vマイグレーション

Virtual to Virtual : V2V

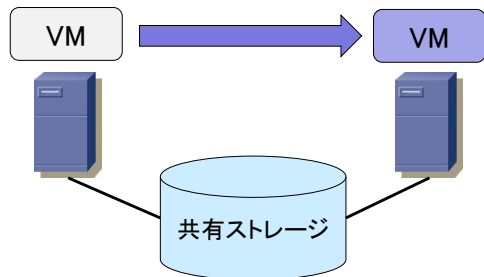
ハイパーバイザによって仮想マシンのフォーマットが異なるため、変換が必要

多くの場合互換性が有り、変換ツールも用意されている



5-8. ライブマイグレーション

仮想マシンをある物理サーバから他の物理サーバに移行する際に、ノンストップで行う技術がライブマイグレーション
VMwareがVMotionとして実装。現在のハイパーバイザはほぼ対応している
VMのストレージに共有ストレージを用いることで実装されている



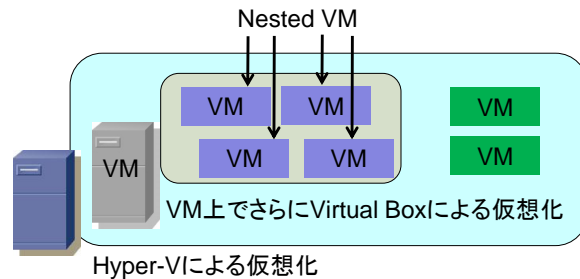
5-9. Nested VM

仮想マシン上でさらに仮想マシンを動かすこと、またはそれによって稼働しているVMのこと。入れ子VMなどとも呼ぶ

主要ハイパーバイザは対応済み。ただしオプション扱いも

Hyper-VではPowerShell上から以下のコマンドが必要

Set-VMProcessor -VMName <VM名> -ExposeVirtualizationExtensions \$true



・ポイント

VMの中にさらにVMを立てることが現在では容易にできます。

現在のハイパーバイザではほとんどがデフォルトでNested VMを有効にしていますが、スライドの説明にあるように、Hyper-Vではデフォルトは無効で、設定メニューには項目がないのでコマンドで有効にする必要があります。

第6章 コンテナの概要

コンテナシステムについて概要を
理解する

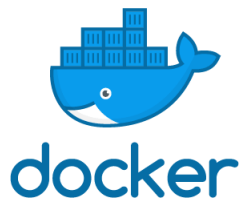
6-1. コンテナとは

Linuxカーネルの「コンテナ」機能を利用した分離環境

プロセスのように、コンテナ分離

ホスト名、ファイルシステム、ユーザ名、プロセスID、ネットワークなどを、
コンテナごとに独自設定可能

オープンソースのコンテナエンジンDockerが人気



・説明の流れ

コンテナは元々Linuxのコンテナ機能から来ていますが、なじみのない人にはわかりにくいでしょう。

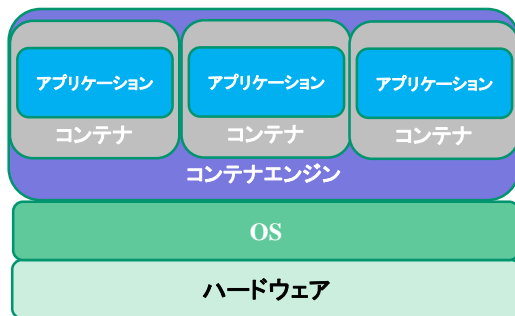
まずコンテナについて知っているかどうか質問し（おそらく大半の人が知らないと思いますが）説明を行います。

また、Dockerという名前について聞いたことがあるかも質問してみましょう。

Dockerは開発環境としてすでに定着しており、名前は聞いたことがある、という人もこれからの説明と演習で概要はつかめるようになるということを知ってもらえるでしょう。

6-2. コンテナの動作

アプリケーションはコンテナ単位で独立
1つのOS上に複数のアプリケーションコンテナが動作
ハードウェアリソースの消費が非常に少ない
現在多く使用されているコンテナエンジン： Docker

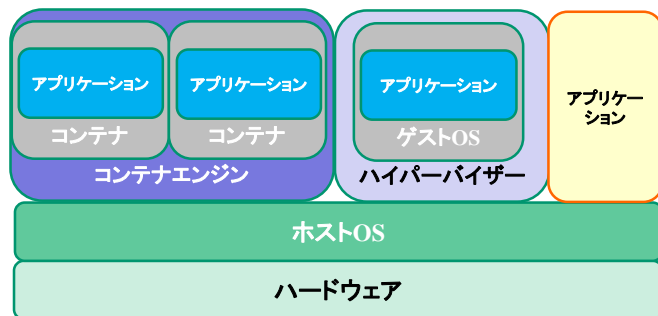


・ポイント

このスライドと次のスライドで、コンテナと仮想化の違いをおさえます。
ハードウェアリソースの消費が非常に少ないという点が重要です。

6-3. 仮想化との違い

VMはゲストOSの容量が必要、ハードウェアのオーバーヘッド大
コンテナはOSやハードウェアリソースはホストOSと共通
VM内アプリケーションはすべて同一ネットワーク
コンテナはそれぞれ隔離され、ネットワークも独立可能



・ポイント

前のスライドと併せて仮想化との違いおさえていきます。

コンテナは隔離・分離されているということ、また仮想化と異なり、OSやハードウェアはホストと共通のものを使用することが重要です。

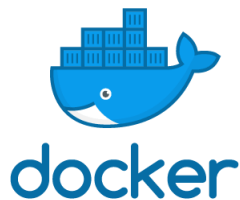
6-4. Dockerの概要

Docker社によるオープンソースのコンテナエンジン

無償版 : Docker Community Edition (Docker CE)

商用版 : Docker Enterprise Edition (Docker EE)

基本はLinuxだが、Windows, macOSにも対応



・補足説明

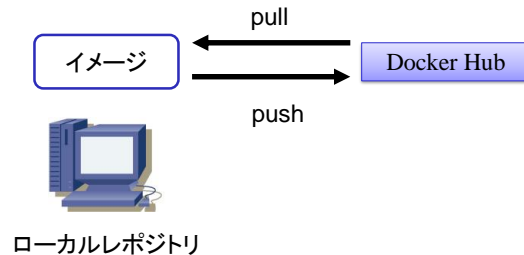
ここからDocker自体の説明に入ります。

Dockerは2017年にスライドにあるようにDocker CE/EEにエディションが分かれました。最近のことなので、少し前のWebサイトや書籍では単にDockerとしか書かれておらず、また詳細も異なります。

もし他に資料を当たる場合、2017年を一つの区切りとして考え、CE/EE以前か以降かを意識してください。

6-5. コンテナの概要

コンテナはDocker Hubに登録されている
コマンドでDocker Hubからダウンロードし実行 (pull)
実機内にはローカルレポジトリが構築される
自分が作成したコンテナイメージをDocker Hubにアップロード可能 (push)



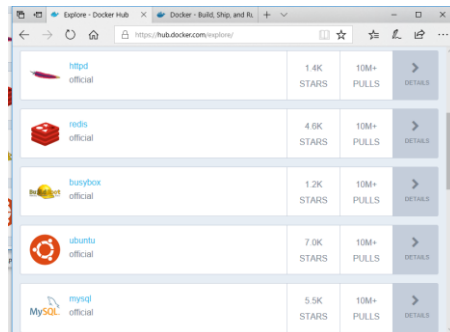
・補足説明

Dockerとコンテナの関係は、VagrantとBoxの関係と似ています。
Dockerはネットワーク経由でDocker Hubに登録されたコンテナをダウンロードして使用します。

もちろん、自分でコンテナを作成することもできますが、最初はず
Docker Hubに登録された様々なサンプルをダウンロードして
Dockerに慣れるところから始めます。

6-6. Docker Hub

ユーザが作成したコンテナイメージを自由に公開・共有できるサービス
ローカルにないコンテナイメージはここからダウンロードして実行される
利用は無料だが、アップロードしたコンテナイメージは原則公開
商用版(Docker Enterprise Edition)の利用なら、非公開のプライベートHub
として利用可能



・補足説明

Docker Hub上のコンテナは無料で自由に使用できます。

有料でプライベートHubにすると、非公開にできます。自社でのみ使用するコンテナを共有したりするにはDocker EEで非公開Hubを作成します。

6-7. DockerとOS

DockerはLinuxのコンテナ技術を使用している

そのため、Dockerの対応OSはLinux

WindowsやmacOSでも動作するが、仮想環境と組み合わせている

Windows: Hyper-V上にLinuxを作成し、その上でコンテナを構築

Hyper-Vが必要なため、Windows 10 Pro以上で動作

macOS: 10.10.3(Yosemite)以降に搭載された仮想環境フレームワーク

Hypervisor.frameworkを利用して、Linuxを作成。その上でコンテナを構築

・補足説明

コンテナは実質Linux専用と考えてもよいでしょう。Win/Macの場合もこのように使用できますが、Windowsの場合は仮想化環境上ですし、MacもLinuxを一度作成して使用します。そのため、ハードウェアリソースの消費が非常に少ない、OSやホストを共有、というコンテナのメリットが活かせません。

6-8. コンテナの用途

コンテナは仮想化と異なりコンパクト、起動も速い
アプリケーション単位で隔離されるため、仮想化で実現するには大がかりな
ことが簡単に行える

同一ツールの複数バージョンを同居：Pythonのバージョン違いを同居
分離されたデーモンプロセスの複数起動：httpdの複数起動
開発環境や検証環境をコンテナで構築し、配布することで環境の同一を保つ
……他にも様々な用途がある

・ポイント

コンテナは単純に軽いというだけでなく、環境をコンテナ上に隔離できるため、様々なメリットがあります。

6-9. コンテナのサイズ

Ubuntuのコンテナは111MByte

“Hello from Docker!” と表示するだけのhello-worldは、わずか1.85kByte
ベースとなるOSとの差異がアプリケーションコンテナとして格納されている
ので、一般にコンテナのファイルサイズは小さい

```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest   00fd29ccc6f1   11 days ago   111MB
hello-world    latest   f2a91732366c   5 weeks ago   1.85kB
```

・説明の流れ

ここで、コンテナサイズに注目する。

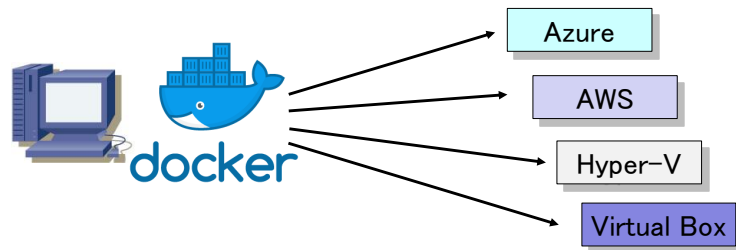
“Hello from Docker!” と表示するだけのコンテナhello-worldは
動作確認に使用されるサンプルコンテナだが、ベースはubuntuで、
ubuntu上に構築すると、ほとんどの部分がベースと共通なため、差
違の表示部分のみがコンテナとして消費する。そのため、驚異的なサ
イズに収まる。

6-10. Dockerのプロビジョニング

プロビジョニング：必要に応じてコンピュータ・リソースを提供・準備すること、または自動構築すること

Docker Machineによって仮想環境のHyper-VやVirtual Box, クラウド環境のAmazon EC2やMicrosoft Azureなどに展開可能

ローカルだけではなく、クラウド環境でもサポートされている



・補足説明

プロビジョニングは様々なところで自動構築の用語として使用されているが、dockerでも同様。

クラウド上への展開もサポートされており、AzureやAWSなど主要なクラウドでも独自のコンテナを展開して使用できる。

第7章 コンテナの実践

Dockerを使用してコンテナを学
ぶ

7-1. 演習

演習4 : Dockerのインストール&Hello Docker

演習5 : コンテナのカスタマイズ、独自コンテナの実施

演習6 : Docker Hubへ独自コンテナの公開

仮想化

目次

第1章 仮想化概要

1-1. 仮想化とは9
1-2. 仮想化の種類11
1-3. 仮想化製品12
1-4. サーバ仮想化13
1-5. デスクトップ仮想化14
1-6. 仮想化とクラウド15

目次

第2章 仮想化概要

2-1. ネットワークとは	16
2-2. LAN	17
2-3. WAN	18
2-4. IPアドレス	19
2-5. ネットワーク部とホスト部	22
2-6. IPアドレスのクラス	24

目次

第3章 仮想環境の構成要素

3-1. 仮想化のコンポーネント	27
3-2. 仮想マシン	28
3-3. 仮想CPU	29
3-4. 仮想メモリ	32
3-5. 仮想HBA, 仮想ディスク	33
3-6. 仮想NIC	34
3-7. 仮想スイッチ、仮想ルータ	35
3-8. 演習	36

目次

第4章 仮想ネットワーク

4-1. 仮想ネットワークの概要	38
4-2. ネットワークとVLAN	39
4-3. 仮想NICとネットワーク形態	40
4-4. 仮想ネットワーク Host-only	41
4-5. 仮想ネットワーク NAT	42
4-6. 仮想ネットワーク Bridge	43

目次

第5章 仮想環境の運用

5-1. 仮想環境の運用	45
5-2. 様々な構築支援ツール	46
5-3. Vagrantの概要	47
5-4. 運用とバックアップ	49
5-5. 移行 (マイグレーション)	51
5-6. P2Vマイグレーション	52
5-7. V2Vマイグレーション	53
5-8. ライブマイグレーション	54
5-9. Nested VM	55
5-10. 演習	56

目次

第6章 コンテナの概要	
6-1. コンテナとは	58
6-2. コンテナの動作	59
6-3. 仮想化との違い	60
6-4. Dockerの概要	61
6-5. コンテナの概要	62
6-6. Docker Hub	63
6-7. DockerとOS	64
6-8. コンテナの用途	65
6-9. コンテナのサイズ	66
6-10. Dockerのプロビジョニング	67
第7章 コンテナの実践	
7-1. 演習	69

第1章 仮想化概要

仮想化の基本を学ぶ

1-1. 仮想化とは

プロセッサやメモリ、ディスク、通信回線など、コンピュータシステムを構成する資源（リソース）を、物理的構成に拠らず柔軟に分割したり統合したりすること

1台のサーバコンピュータをあたかも複数台のコンピュータであるかのように論理的に分割し、それぞれに別のOSやアプリケーションソフトを動作させる「サーバ仮想化」や、複数のディスクをあたかも1台のディスクであるかのように扱い、大容量のデータを一括して保存したり耐障害性を高めたりする「ストレージ仮想化」などの技術がある

IT用語辞典より転載

1-2. 仮想化の種類

仮想化は次の種類に分けられる

- サーバ仮想化
- デスクトップ仮想化
- ネットワーク仮想化
- ストレージ仮想化

1-3. 仮想化製品

代表的な製品 (ハイパーバイザ)

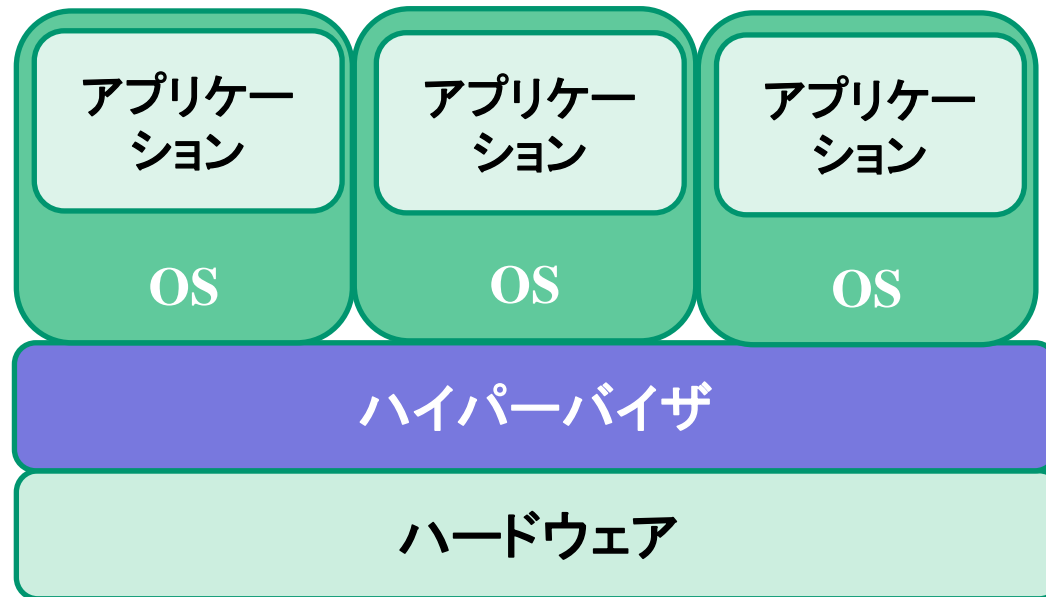
- サーバ仮想化 (Type-I)
 - VMware Sphere
 - Microsoft Hyper-V
 - Xen
- デスクトップ仮想化 (Type-II)
 - VMware Workstation, Player, Fusion
 - Microsoft Hyper-V
 - Virtual Box

1-4. サーバ仮想化

Type-I型、ネイティブ型、ベアメタル型

ハードウェア上で直接動作するハイパーバイザ

OSはすべてハイパーバイザ上で動作し、ほぼネイティブ環境に近い動作速度が得られる

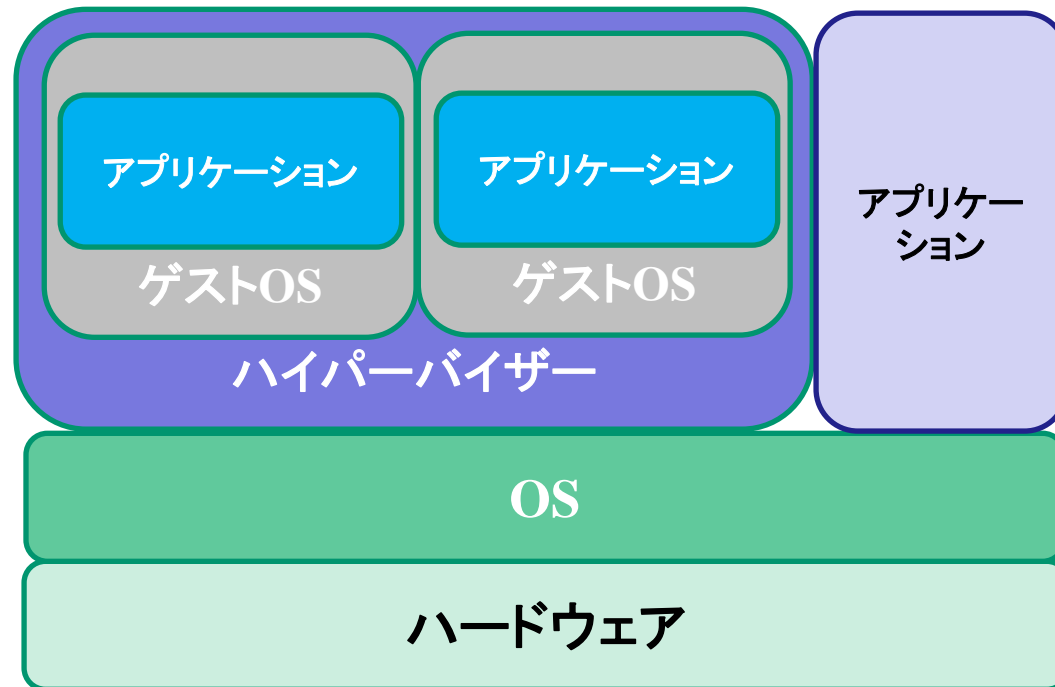


1-5. デスクトップ仮想化

Type-II、ホストOS型

ハードウェア上にまずOSが動作し、その上で1アプリケーションとしてハイパーバイザーが動作

ハイパーバイザー上で仮想環境が動作



1-6. 仮想化とクラウド

クラウドサービスでは仮想化環境を提供

クラウドサービスはユーザが自由にリソースを変更可能。仮想化環境がマッチしている

	suse-sles-12-sp3-v20171212-hvm-ssd-x86_64 - ami-8368cefb SUSE Linux Enterprise Server 12 SP3 (HVM, 64-bit, SSD-Backed) ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	RHEL-7.4_HVM_GA-20170808-x86_64-2-Hourly2-GP2 - ami-9fa343e7 Provided by Red Hat, Inc. ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20171121.1 - ami-0def3275 Canonical, Ubuntu, 16.04 LTS, amd64 xenial image build on 2017-11-21 ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	Windows_Server-2016-English-Full-Base-2017.11.29 - ami-f6d8008e Microsoft Windows Server 2016 with Desktop Experience Locale English AMI provided by Amazon ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい

第2章 ネットワークの基本

仮想化技術を習得する上で最低限
の知識

2-1. ネットワークとは

ネットワークとは、網という意味の英単語。複数の要素が互いに接続された網状の構造体のこと。ネットワークを構成する各要素のことを「ノード」(node)、ノード間の繋がりのことを「リンク」(link)あるいは「エッジ」(edge)と言う。

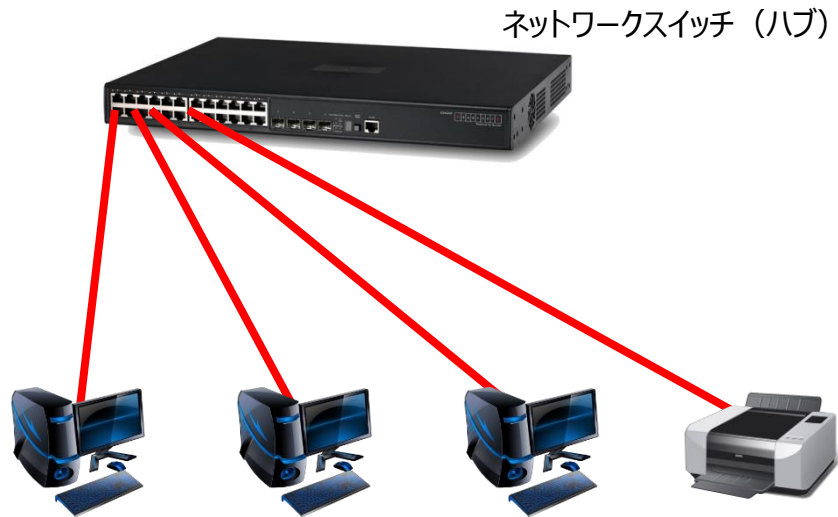
一般の外来語としては人間関係の広がりのことや、組織や集団の構造などを指すこともあるが、IT関連の分野で単にネットワークという場合は、複数のコンピュータや電子機器などを繋いで信号やデータ、情報をやりとりすることができるコンピュータネットワークあるいは通信ネットワークのことを意味することが多い。

IT用語辞典より転載

2-2. LAN

LAN (Local Area Network)

- 1つのフロア, 組織, 部署といった, 比較的狭い範囲でのネットワーク



配線図

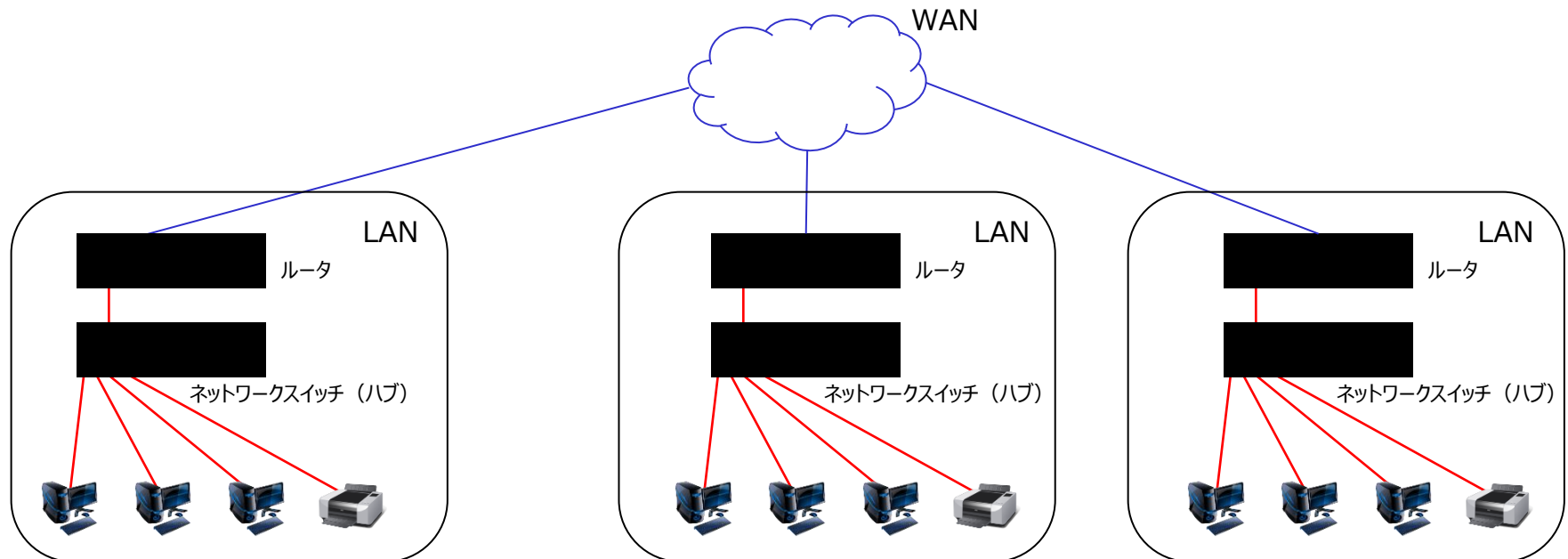


論理図

2-3. WAN

WAN (Wide Area Network)

- 広域通信網 (Wide Area Network) の略。
LANとLANを結ぶ公衆網のことを指す場合が多い。
WANを世界規模で実現しているのがインターネット。



2-4. IPアドレス

- ネットワーク上で機器の識別をするための番号
- 32bit でIPアドレスを管理 (IPv4, Internet Protocol Version 4)
 - bit … 情報量の最小単位, 1 or 0
- ネットワーク部とホスト部の存在

192.168.1.10

10進 → 2進に変換

$$192_{(10)} = \boxed{1100\ 0000}_{(2)}$$

$$168_{(10)} = \boxed{1010\ 1000}_{(2)}$$

$$1_{(10)} = \boxed{0000\ 0001}_{(2)}$$

$$10_{(10)} = \boxed{0000\ 1010}_{(2)}$$

IPアドレス

192.168.1.10

1100 0000

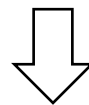
1010 1000

0000 0001

0000 1010



2進数の並びが $8 \times 4 = 32$ コ



32bit

※ 8bitのかたまり = 1オクテットと呼ぶことも
octet

2-5. ネットワーク部とホスト部

192.168.1.10/24

この例だと、IPアドレスのビット列の、

先頭から**24bit** = ネットワーク部

残りの**8bit** = ホスト部

1100 0000 1010 1000 0000 0001

0000 1010

← 24bit : ネットワーク部

8bit : ホスト部 →

ネットワークプレフィックスと言うことも。
network prefix

ネットワーク内の端末を
識別

端末がどのネットワークに所属しているのかを判別.

ひとつのネットワークに収容できるホスト数

ホスト部がすべて 0 ⇨ ネットワーク自身のことを表す

ホスト部がすべて 1 ⇨ そのネットワーク全体にデータを送る
ブロードキャストアドレスを表す

192.168.1.10 ならば…

ネットワーク自身 ⇨ **192.168.1.0**

1100 0000 1010 1000 0000 0001 0000 0000

すべて 0

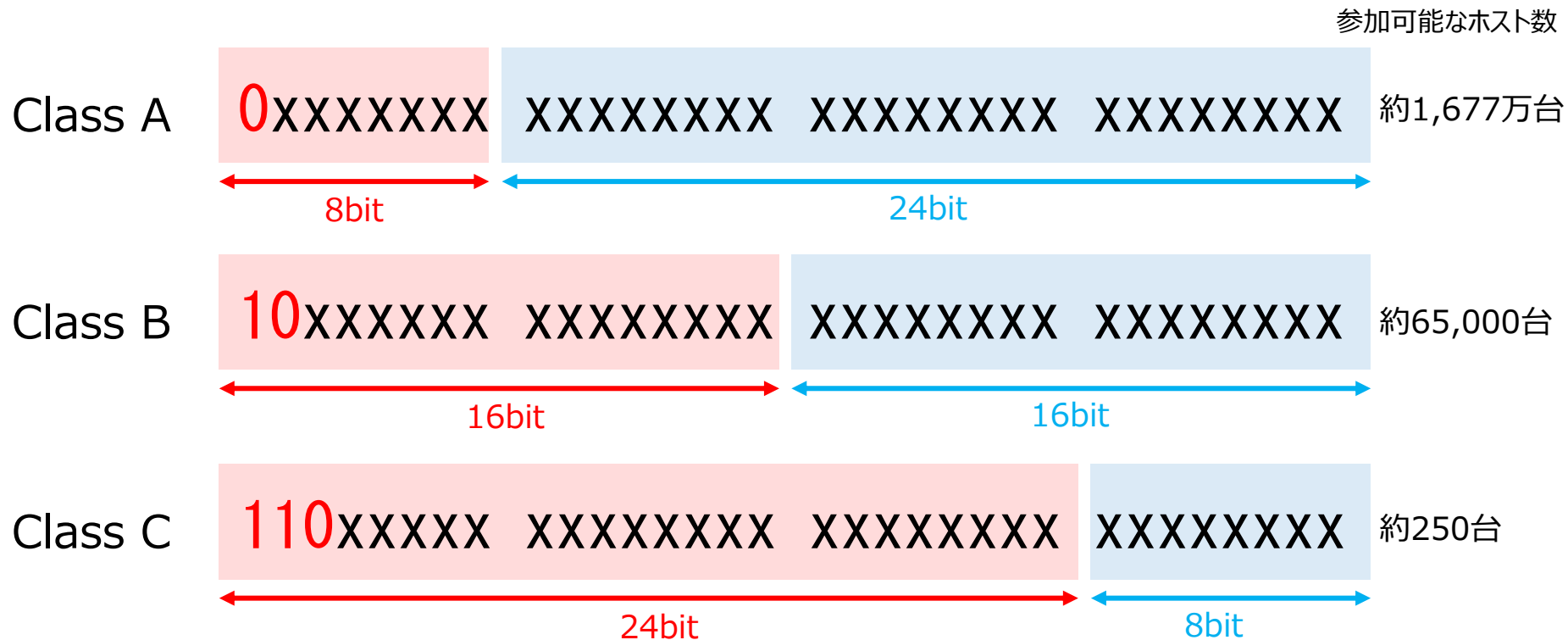
ブロードキャストアドレス ⇨ **192.168.1.255**

1100 0000 1010 1000 0000 0001 1111 1111

すべて 1

2-6. IPアドレスのクラス

- 32bit中, 「何bitがネットワーク部で, 何bitがホスト部を表すか」
- ネットワークの規模を識別する



他にもClass D, Class Eも存在

IPアドレスのクラス

(例題) 次のIPアドレスのクラス (ネットワークの規模) は？

172.16.0.1

10進 → 2進に変換

$$172_{(10)} = \boxed{1010 \ 1100}_{(2)}$$

先頭ビットが 10… なので, 「**クラスB**」であることがわかる

第3章 仮想環境の構成要素

主要コンポーネントについて理解
する

3-1. 仮想化のコンポーネント

主なコンポーネント

- 仮想マシン
- 仮想CPU
- 仮想メモリ
- 仮想HBA
- 仮想ディスク
- 仮想NIC
- 仮想スイッチ



3-2. 仮想マシン

物理的なPC,サーバをハイパーバイザによって仮想的な機械に置き換えたもの

Virtual Machine : VM

構成ファイル、仮想ストレージなど構成される

Hyper-Vの例

 ubuntu14d	2017/11/29 22:41	ファイルフォルダー	
 ubuntu14-desktop.vhdx	2017/12/19 14:29	ハードディスク イメー...	9,441,280 KB



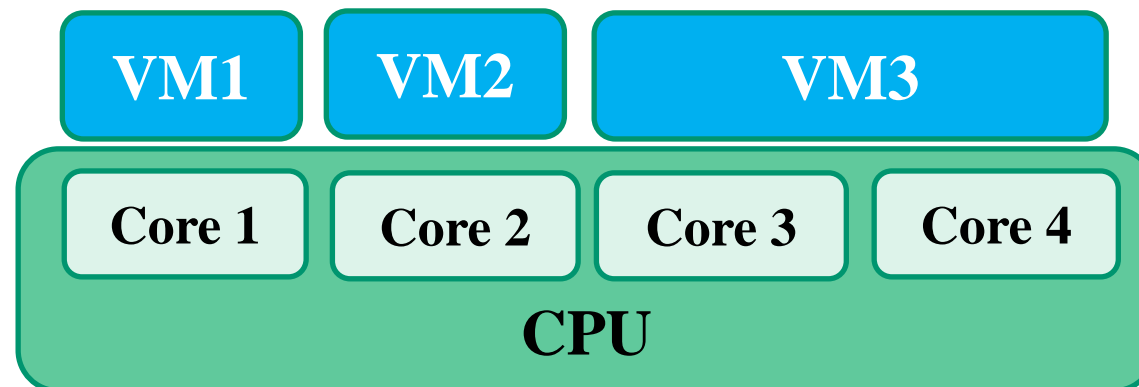
仮想マシン(ハードディスクイメージ)

3-3. 仮想CPU

仮想マシンからは仮想CPUが物理CPUとして見える

仮想マシン1つにつき1つのCPU、もしくは1つのコアを割り振るのが理想的
重い仮想マシン (VM3) には多くのコアを割り振り、軽い仮想マシン
(VM1,VM2) には少ないコアを割り振る

商用のハイパーバイザの中には、CPUのコア数、仮想CPUの総コア数でライセンスが決まるものもある



3-3. 仮想CPUと仮想化支援機能

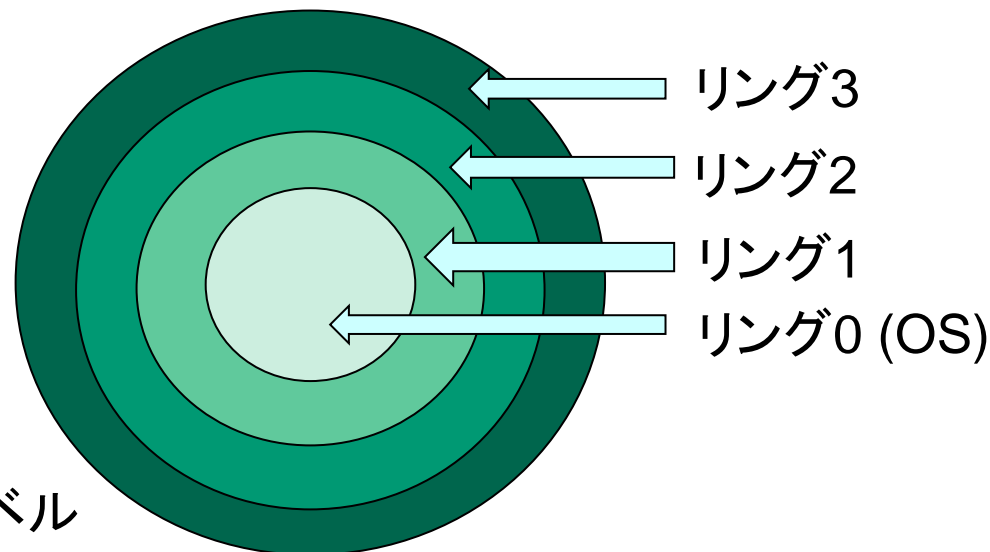
近年のCPUには、仮想CPUを物理CPUとほぼ同等に動作させる仮想化支援機能がある

Intel CPU: VT-x, AMD CPU:AMD-V

CPUの特権レベルが最も高いリング0にアクセスできるのはOSのみ

アプリケーションやハイパーバイザはリング3で動作

リング3からリング0へのアクセスは例外が発生する

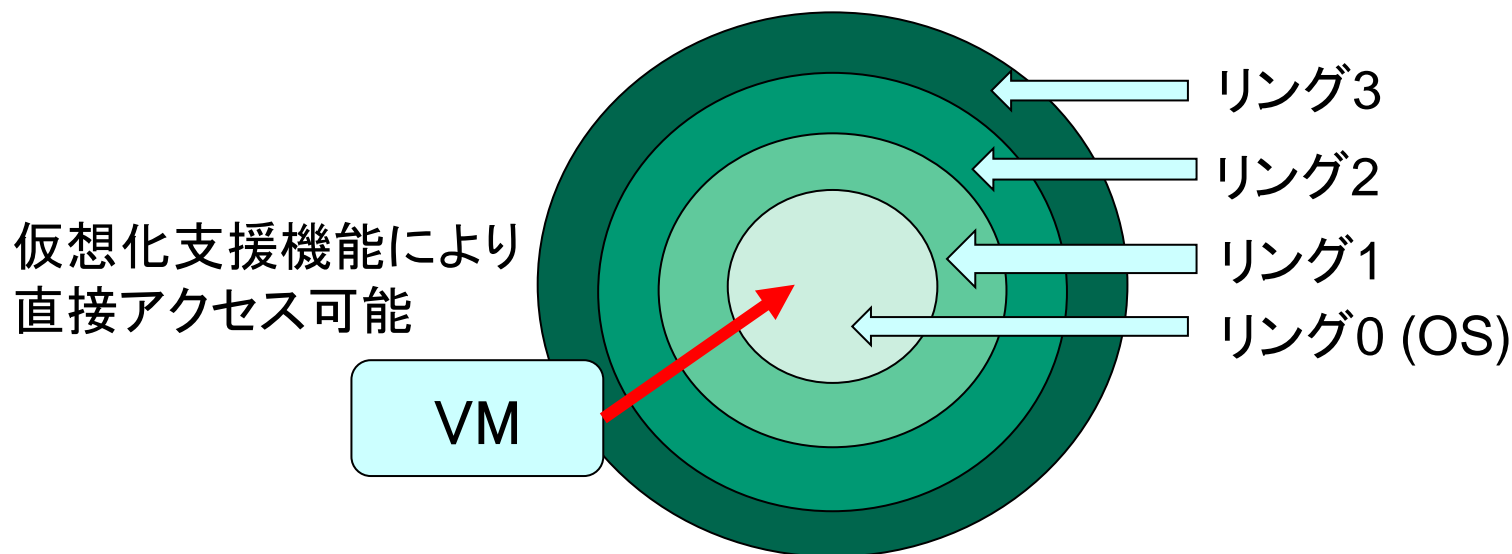


CPUの特権レベル

3-3. 仮想CPUと仮想化支援機能

VT-xやAMD-Vに対応したハイパーバイザではVMからリング0へ例外なしにアクセス可能

I/Oもほぼ直接アクセスできるため、物理環境に近い速度で動作
デフォルトで仮想化支援機能が無効になっているPCがあるので注意



3-4. 仮想メモリ

仮想マシンの動作には当然メモリが必要

ハイパーバイザによって最大メモリ量が異なる

仮想マシンへの仮想メモリとして割り当てる際に、完全固定として割り当てる方法（スタティック）と、最小容量と最大容量を定めておき、動的に変更できる方法がある

Type-I ハイパーバイザ：仮想マシンがほぼすべてのメモリを使用可能

Type-II ハイパーバイザ：他アプリケーションが使用するメモリ容量を考慮して仮想メモリを設計する

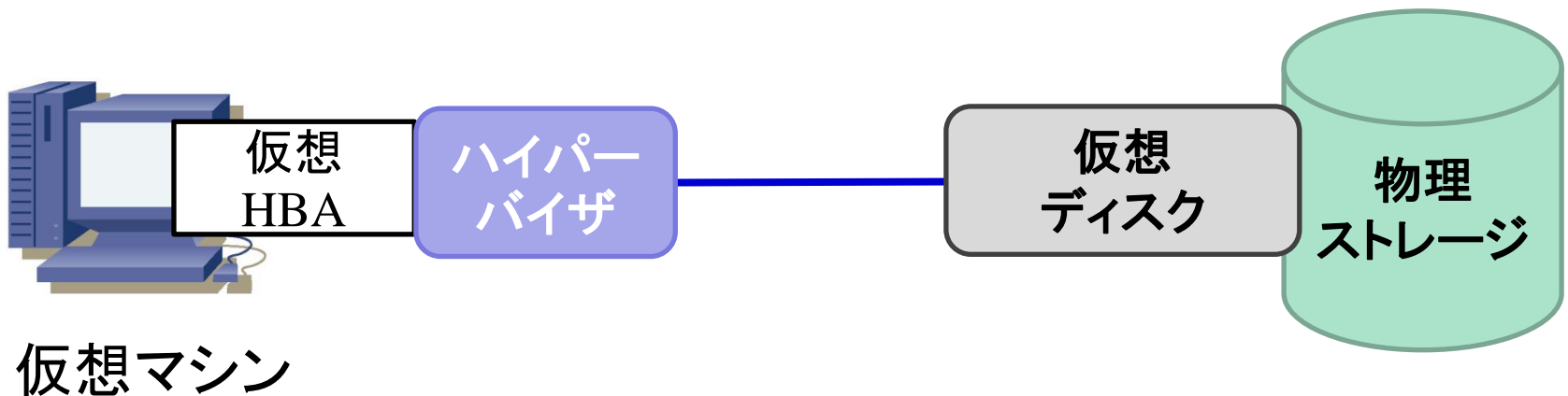
3-5. 仮想HBA, 仮想ディスク

仮想マシンは仮想ディスクをストレージとして使用する

仮想マシンには、ホストバスアダプタ（HBA）としてIDEやSCSIのインターフェイスが接続され、各方式の内蔵ディスクとして見える

仮想ディスクは動的構成を取ることができる

設定上120Gの仮想ディスクを使用しても、実際に使用しているサイズのみ物理ストレージ上では消費する

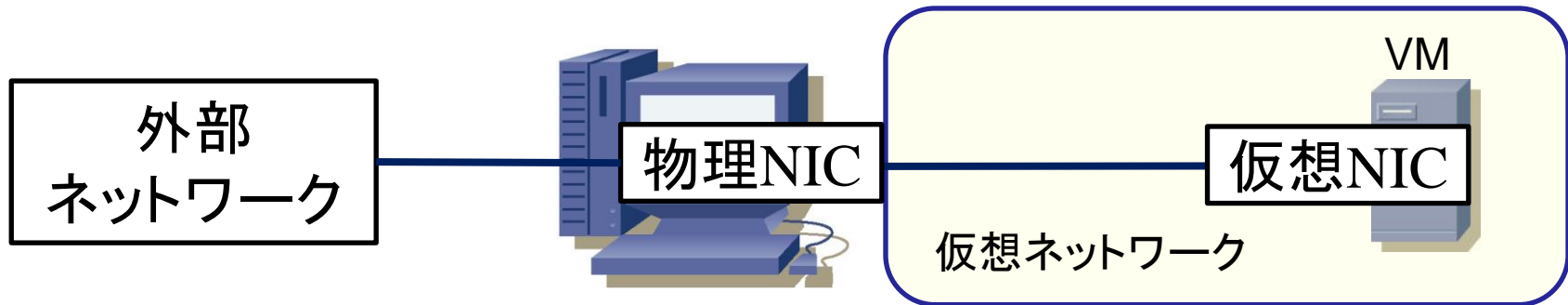


3-6. 仮想NIC

ハイパーバイザによって仮想的なインターフェイスである仮想NICが作成され、仮想ネットワークも作成される

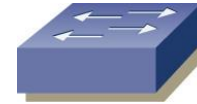
物理NICと仮想NICを接続しないと、ホストOSと仮想OSは通信できない

物理NICと仮想NICの間に仮想ブリッジや仮想ルータを挟むことで、仮想ネットワーク形態が変わる

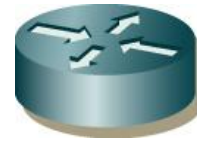


3-7. 仮想スイッチ、仮想ルータ

仮想スイッチ：Virtual Switch (vSwitch)



仮想ルータ：Virtual Router (vRouter)



仮想ネットワークにおいて、それぞれ物理機器と同じ役割を果たす
仮想ネットワークをNATにする場合はvRouterが
VLANを作成する場合はvSwitchが必要

3-8. 演習

演習1 : VirtualBoxによるVM作成

第4章 仮想化ネットワーク

仮想化特有のネットワーク構造について理解する

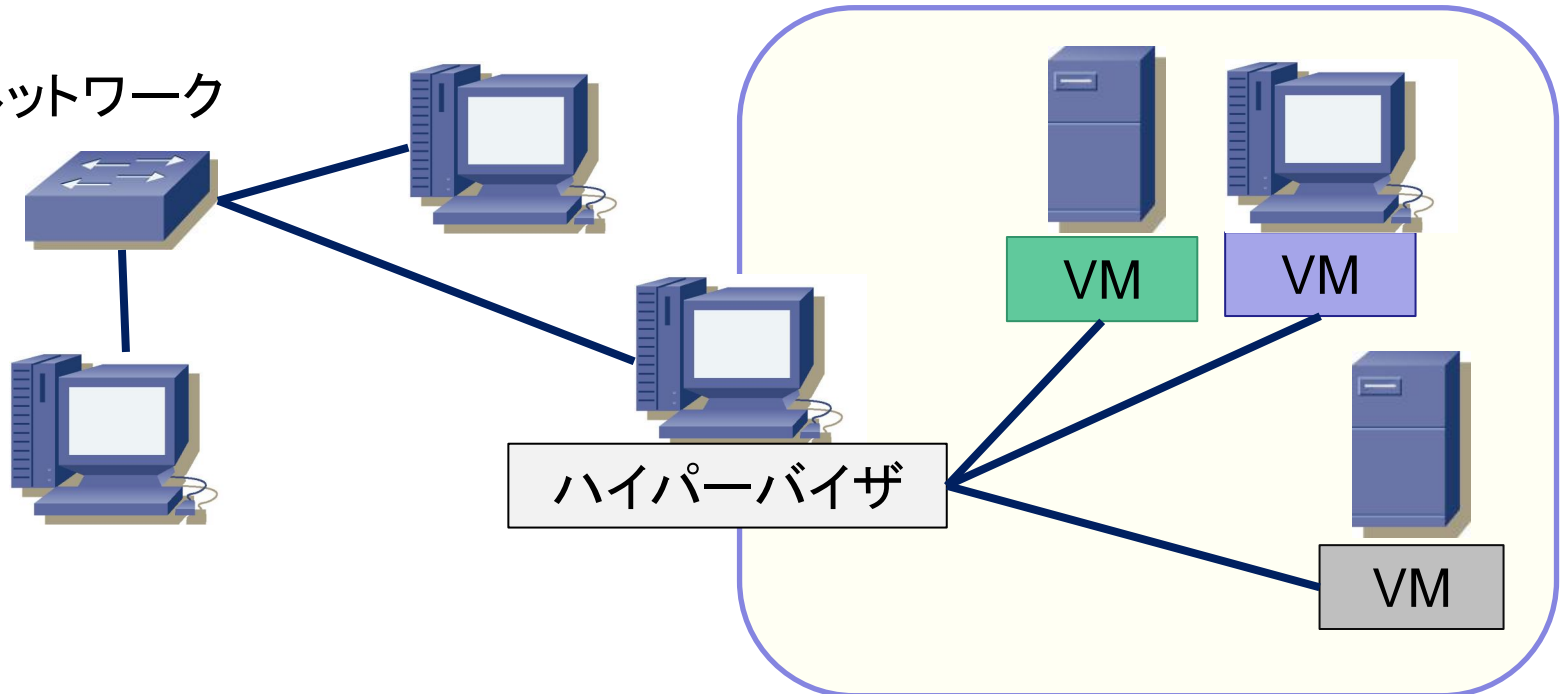
4-1. 仮想ネットワークの概要

ハイパーバイザによって仮想的なネットワークの作成が可能

ハイパーバイザの設定によって、仮想ネットワークを独立させたり、物理ネットワークと接続したり、様々なトポロジ構成が可能

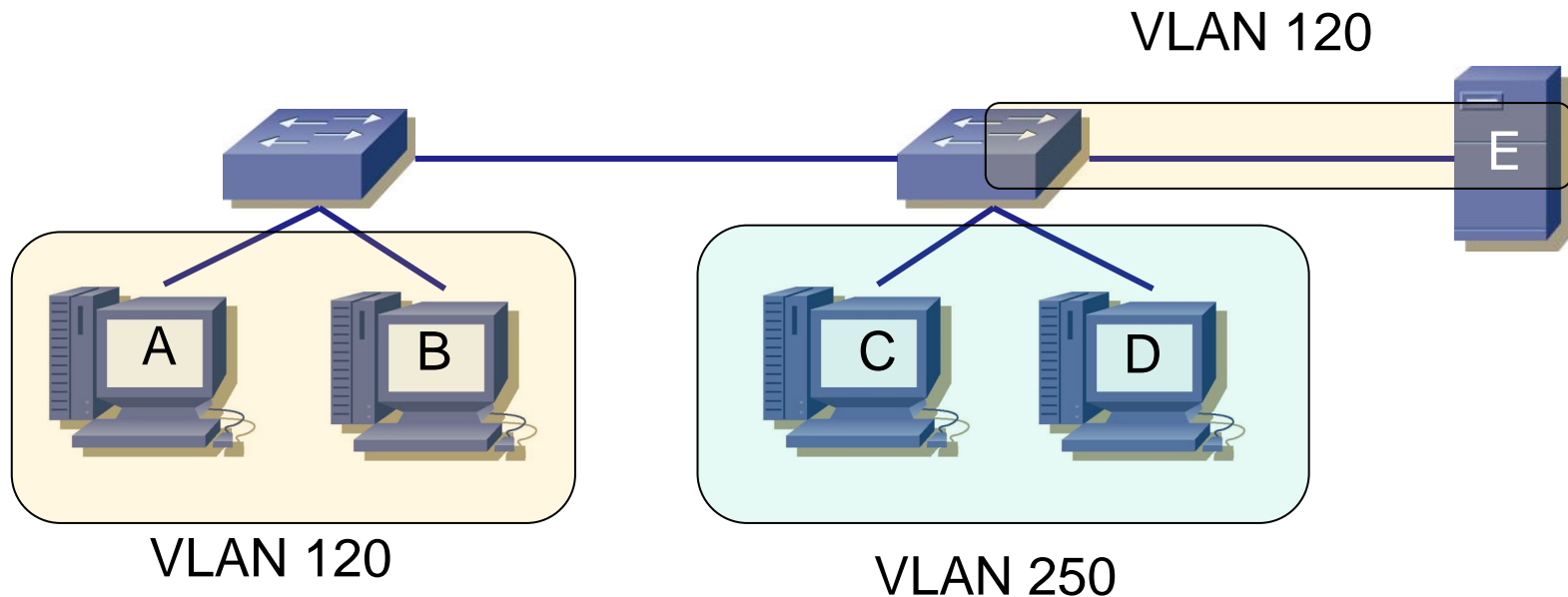
仮想ネットワーク

物理ネットワーク



4-2. ネットワークとVLAN

データリンク層（Ethernet）は本来ネットワークを分割できない
VLAN対応スイッチによって提供されるVLANによって分割可能
仮想ネットワーク内でも仮想スイッチによるVLAN分割が可能



4-3. 仮想NICとネットワーク形態

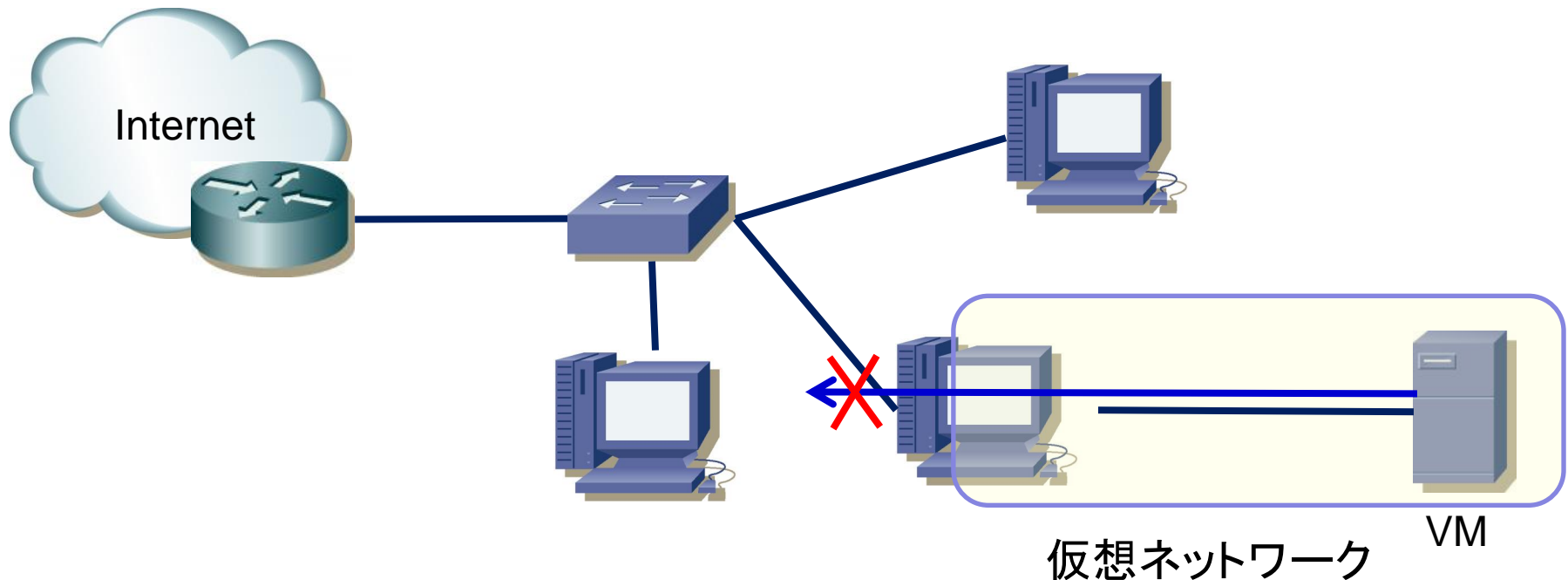
ハイパーバイザによって仮想的なインターフェイスである仮想NICが作成され、仮想ネットワークも作成される

次の3形態

- Host-only
- NAT
- Bridge

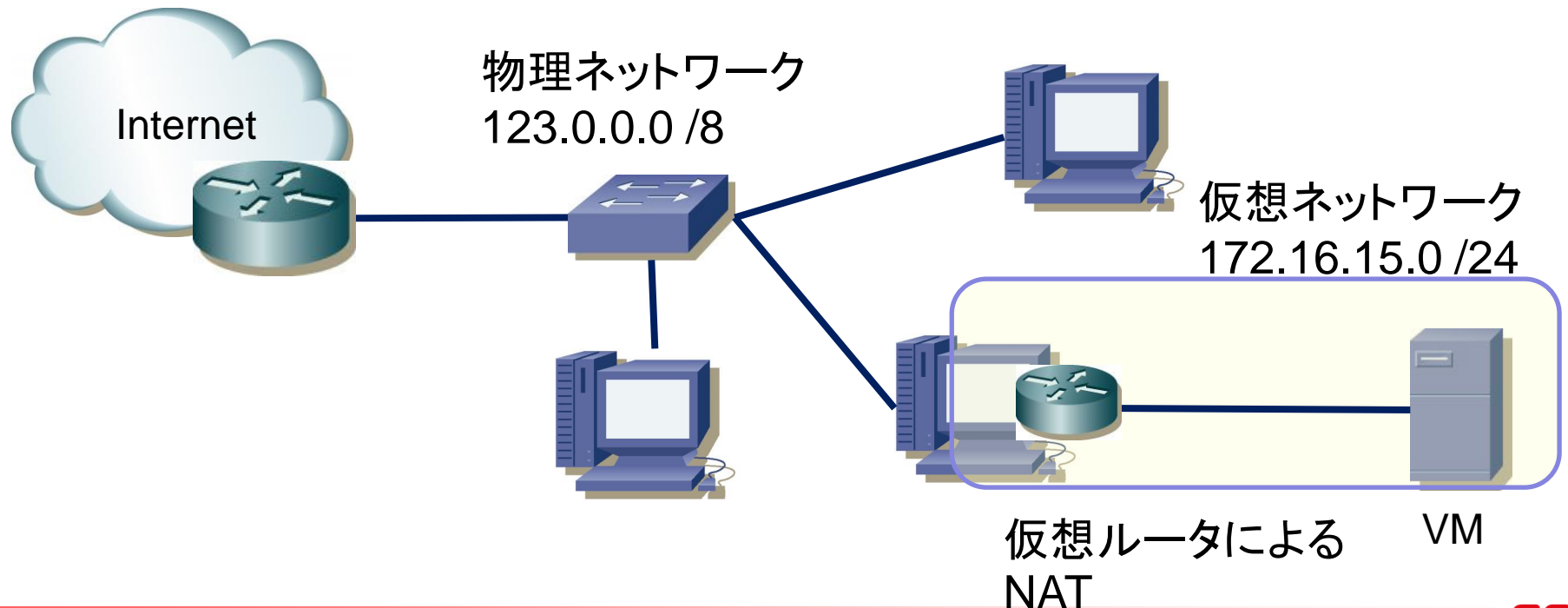
4-4. 仮想ネットワーク Host-only

ハイパーバイザが動作しているホストとのみ通信できるネットワーク
実ネットワーク内の物理マシンや外部ネットワークとは通信できない



4-5. 仮想ネットワーク NAT

仮想ネットワークにはプライベートアドレスが与えられる
ハイパーバイザが提供する仮想ルータによってアドレス変換が行われる
物理ネットワークに影響が少ない

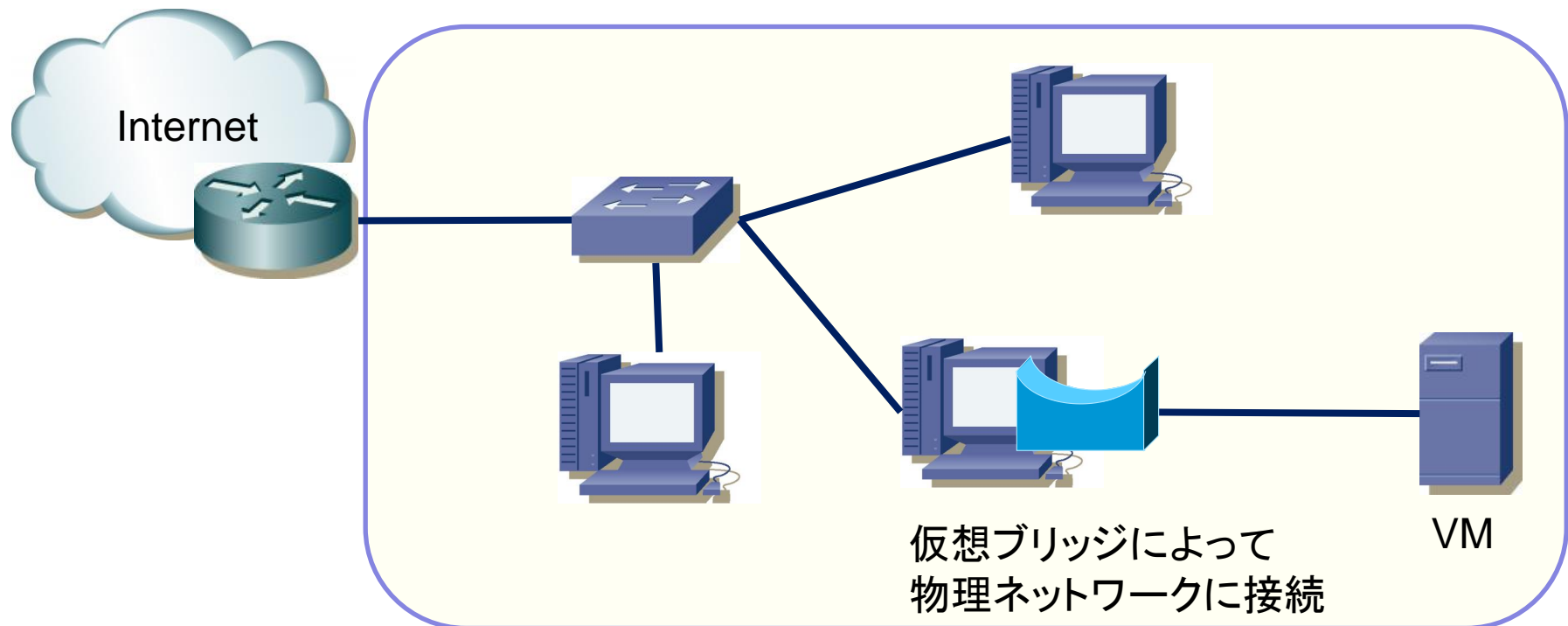


4-6. 仮想ネットワーク Bridge

ハイパーバイザーの提供する仮想ブリッジ（仮想スイッチ）を介して実ネットワークに直接接続

物理マシンと直接通信可能

物理ネットワークに影響を与えるおそれがある



4-7. 演習

演習3：仮想ネットワーク設定

第5章 仮想環境の運用

支援ツールや移行について学ぶ

5-1. 仮想環境の運用

ハイパーバイザの操作：GUI or コマンド

ハイパーバイザ独自のシェル環境など

Hyper-VはPowerShellによるコマンド、Hyper-Vマネージャ

The screenshot displays the Hyper-V Manager application window. The title bar reads "Hyper-V マネージャー". The menu bar includes "ファイル(F)", "操作(A)", "表示(V)", and "ヘルプ(H)". The toolbar contains navigation and help icons. The main area is divided into two panes. The left pane shows the "Hyper-V マネージャー" tree with "OITTHINKPAD13" selected. The right pane, titled "仮想マシン(I)", contains a table with the following data:

名前	状態	CPU 使用率	メモリの割り当て	稼働時間
precise64	オフ			
ubuntu14d	実行中	0%	3004 MB	1.16:51:15

Below the table is a section titled "チェックポイント(C)" with the message: "選択した仮想マシンにはチェックポイントがありません。". On the right side, a "操作" (Operations) pane is visible, listing actions for the selected VM "OITTHINKPAD13":

- クイック作成...
- 新規
- 仮想マシンのインポート...
- Hyper-V の設定...
- 仮想スイッチ マネージャー...
- 仮想 SAN マネージャー...
- ディスクの編集...
- ディスクの検査...
- サービスの停止
- サーバーの削除
- 最新の情報に更新

5-2. 様々な構築支援ツール

実験的に1台構築するだけでも、様々な設定を手動で行うのは煩雑
仮想環境を自動構築するための支援ツールを使用

- Vagrant : 仮想環境構築ツール
- Puppet : 構成管理ツール
- Chef : サーバ設定・更新自動化ツール
- libvirt : 仮想化環境共通API群



5-3. Vagrantの概要

オープンソースの仮想環境構築ソフトウェア、MITライセンス
ファイル(Vagrantfile)に設定を記述し、仮想環境を自動的に構築
VirtualBox, VMware, Hyper-Vなどに対応
Amazon EC2といったクラウドにも対応
コンテナDockerにも対応している

Vagrantfileの例

```
config.vm.box = "generic/ubuntu1604"           #仮想イメージ:ubuntu16.04
config.vm.provider "hyperv" do |h|             #プロバイダ:Hyper-V
  h.cpus = "2"                                  #仮想CPU:2
  h.maxmemory="4096"                            #最大メモリ:4096M
  h.enable_virtualization_extensions=true      #仮想化支援機能:有効
end
```

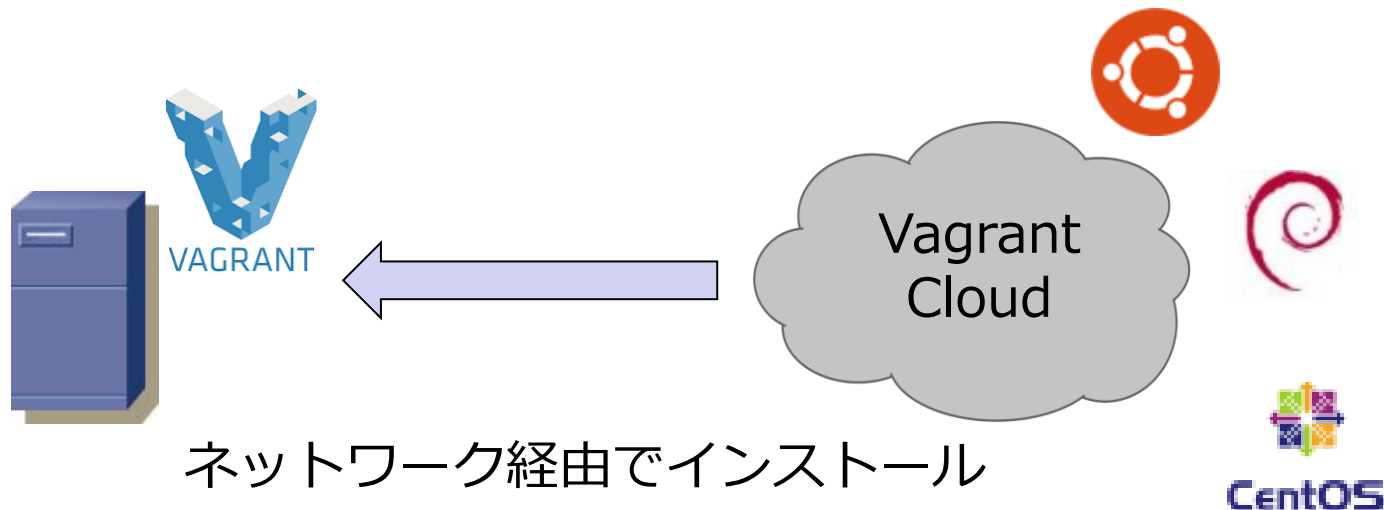
5-3. Vagrantの概要(続き)

仮想イメージ：Box, 動作ハイパーバイザ：プロバイダ

コマンドでVagrant CloudからBoxを導入可能

<https://app.vagrantup.com/>

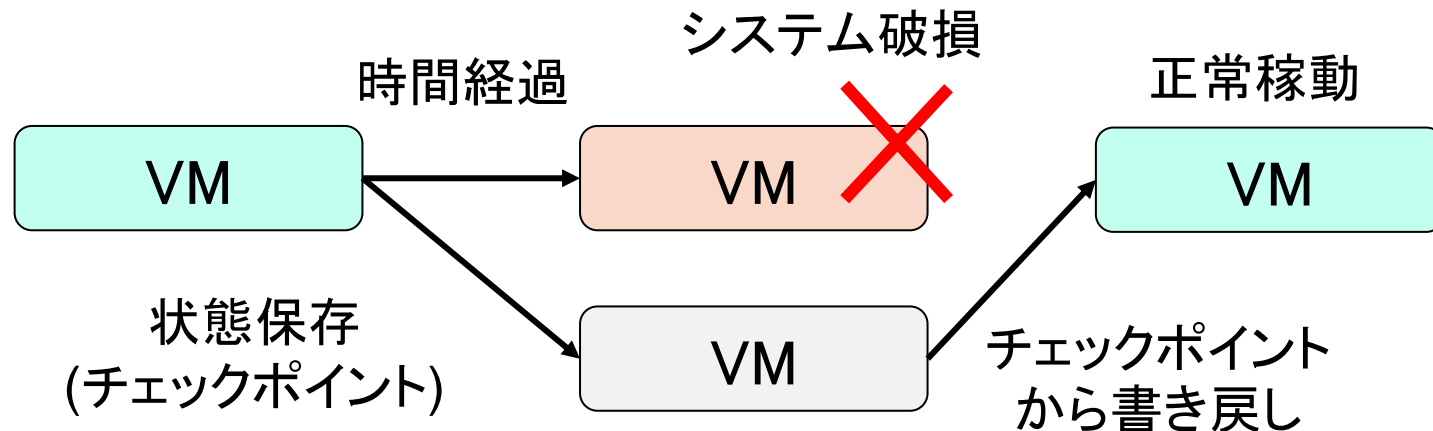
独自Boxの作成も可能



5-4. 運用とバックアップ

仮想マシンの運用には、Vagrantなどの運用ツールの他、ハイパーバイザ用のシェル言語などもある。例) Hyper-VはPowerShell

仮想マシンなら、ファイルベースなのでバックアップも容易
Hyper-Vではある状態を保存することをチェックポイントと呼ぶ



5-5. 移行（マイグレーション）

現在の環境をそのまま仮想環境へ移す、あるいは新規に仮想マシンを作成する、など様々な移行方法がある

仮想マシンは物理マシンと完全に同じではない

ネットワークやハイパーバイザーによっては、仮想環境に対応していないOS、アプリケーション、ハードウェアがある

仮想環境、クラウドで使用するすべてのハードウェア、ソフトウェアについて対応状況をチェックする

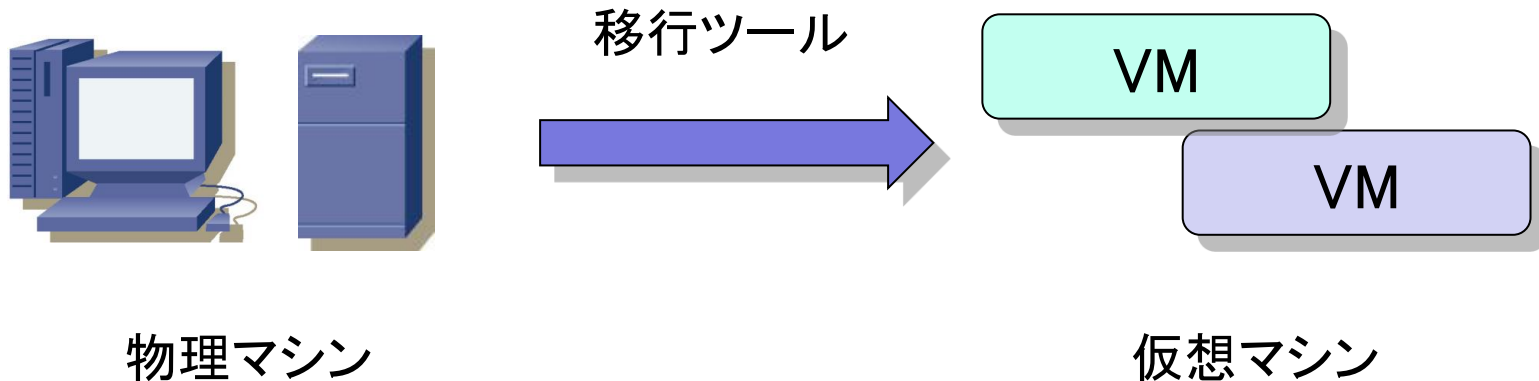
5-6. P2Vマイグレーション

Physical to Virtual : P2V

現在動作している物理マシンを仮想マシンへ移行するツールを使う

無停止で行うホットクローニング

物理マシンのシャットダウンを伴うコールドクローニング

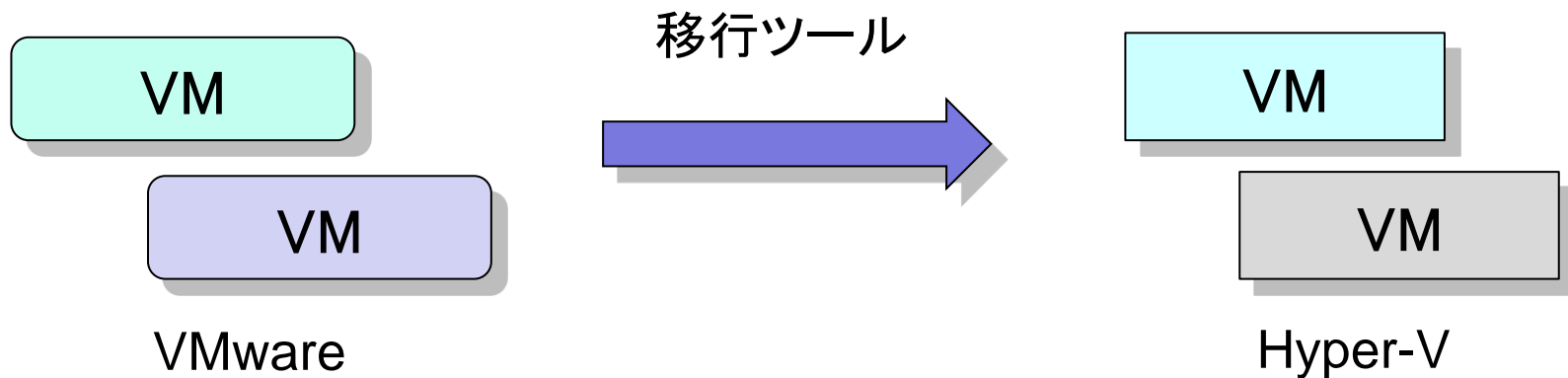


5-7. V2Vマイグレーション

Virtual to Virtual : V2V

ハイパーバイザによって仮想マシンのフォーマットが異なるため、変換が必要

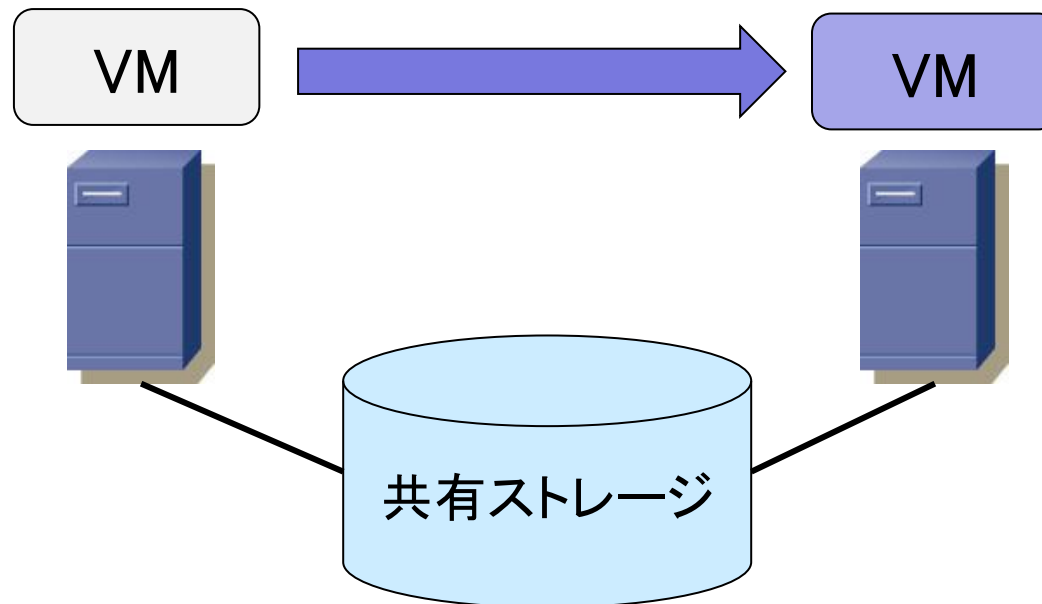
多くの場合互換性が有り、変換ツールも用意されている



5-8. ライブマイグレーション

仮想マシンをある物理サーバから他の物理サーバに移行する際に、ノンストップで行う技術がライブマイグレーション

VMwareがVMotionとして実装。現在のハイパーバイザはほぼ対応している
VMのストレージに共有ストレージを用いることで実装されている



第6章 コンテナの概要

コンテナシステムについて概要を
理解する

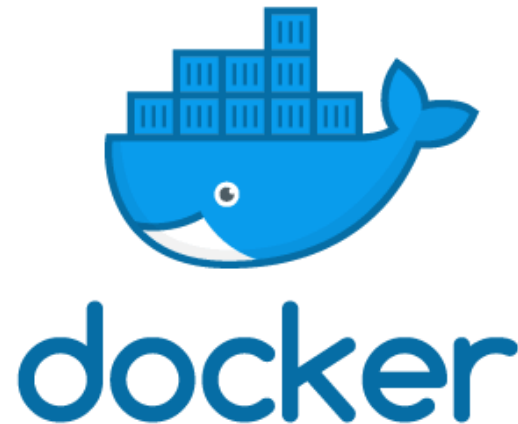
6-1. コンテナとは

Linuxカーネルの「コンテナ」機能を利用した分離環境

プロセスのように、コンテナ分離

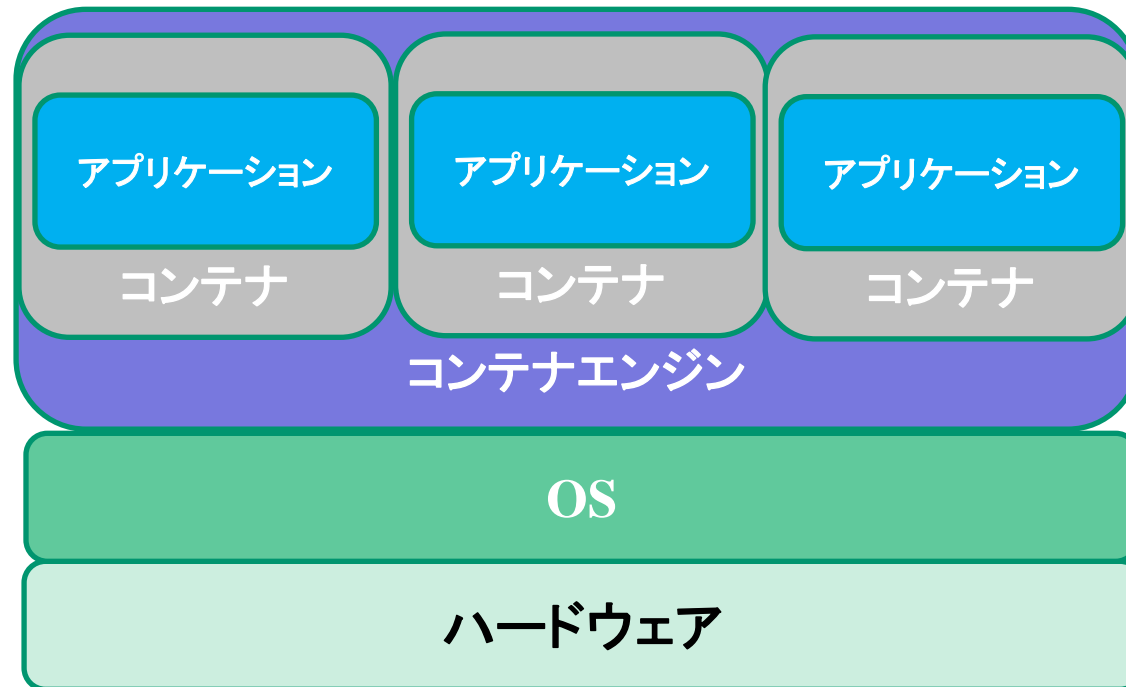
ホスト名、ファイルシステム、ユーザ名、プロセスID、ネットワークなどを、
コンテナごとに独自設定可能

オープンソースのコンテナエンジンDockerが人気



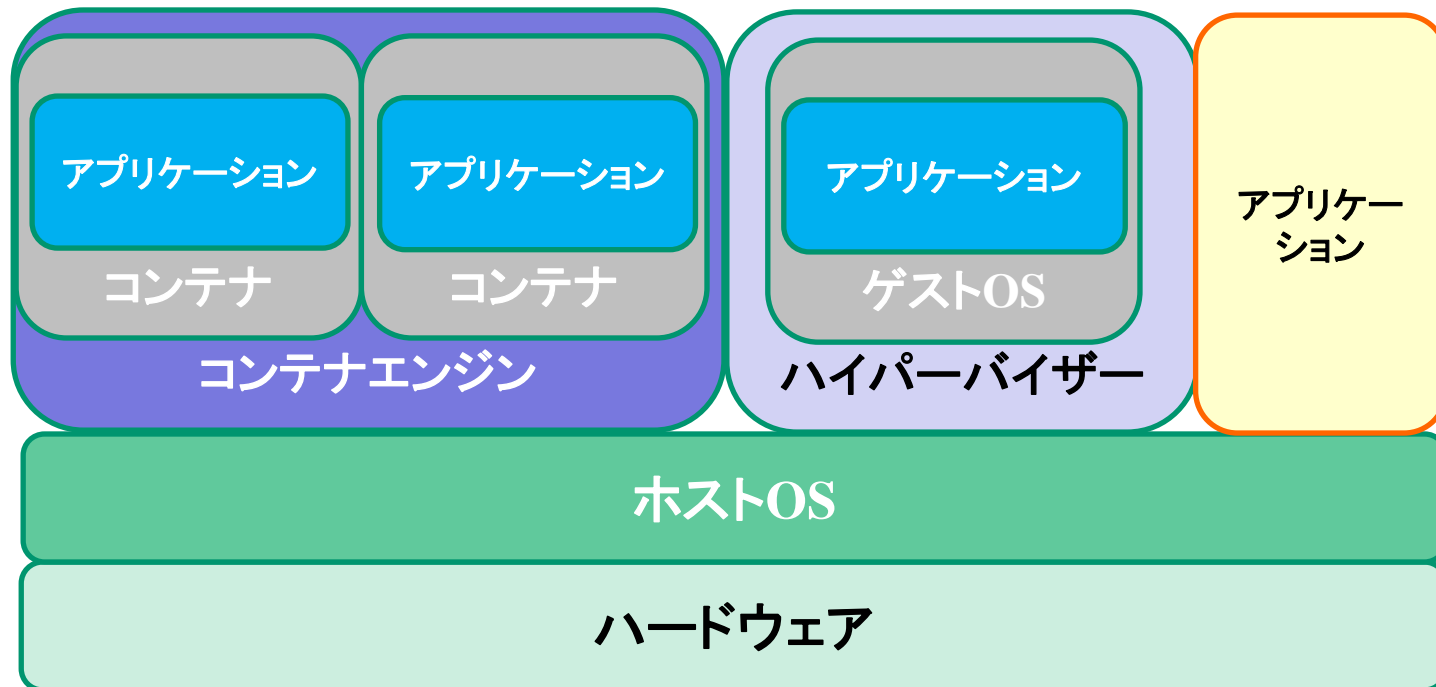
6-2. コンテナの動作

アプリケーションはコンテナ単位で独立
1つのOS上に複数のアプリケーションコンテナが動作
ハードウェアリソースの消費が非常に少ない
現在多く使用されているコンテナエンジン：Docker



6-3. 仮想化との違い

VMはゲストOSの容量が必要、ハードウェアのオーバーヘッド大
コンテナはOSやハードウェアリソースはホストOSと共通
VM内アプリケーションはすべて同一ネットワーク
コンテナはそれぞれ隔離され、ネットワークも独立可能



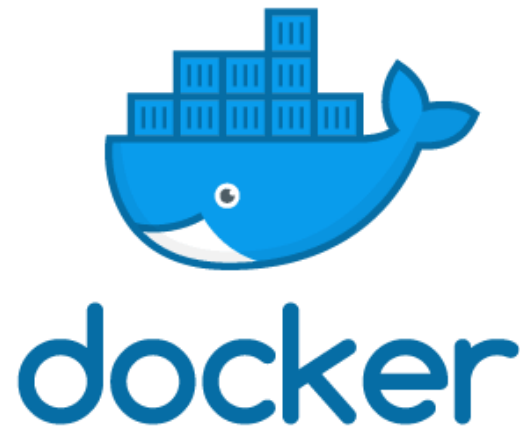
6-4. Dockerの概要

Docker社によるオープンソースのコンテナエンジン

無償版：Docker Community Edition (Docker CE)

商用版：Docker Enterprise Edition (Docker EE)

基本はLinuxだが、Windows, macOSにも対応



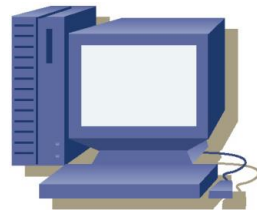
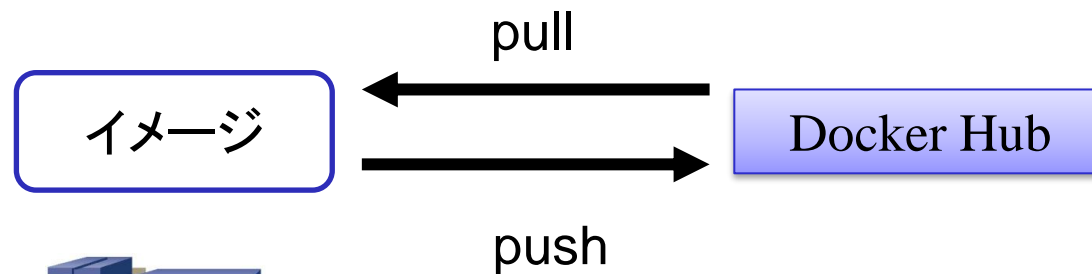
6-5. コンテナの概要

コンテナはDocker Hubに登録されている

コマンドでDocker Hubからダウンロードし実行 (pull)

実機内にはローカルレポジトリが構築される

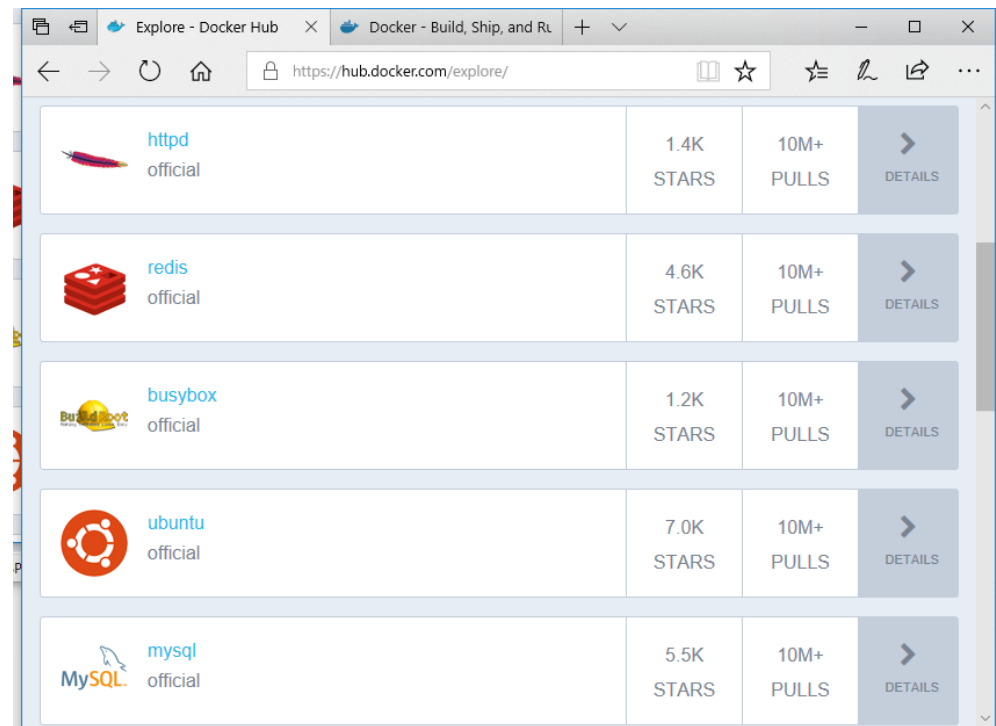
自分が作成したコンテナイメージをDocker Hubにアップロード可能 (push)



ローカルレポジトリ

6-6. Docker Hub

ユーザが作成したコンテナイメージを自由に公開・共有できるサービス
ローカルにないコンテナイメージはここからダウンロードして実行される
利用は無料だが、アップロードしたコンテナイメージは原則公開
商用版(Docker Enterprise Edition)の利用なら、非公開のプライベートHub
として利用可能



6-7. DockerとOS

DockerはLinuxのコンテナ技術を使用している
そのため、Dockerの対応OSはLinux
WindowsやmacOSでも動作するが、仮想環境と組み合わせている

Windows: Hyper-V上にLinuxを作成し、その上でコンテナを構築
Hyper-Vが必要なため、Windows 10 Pro以上で動作

macOS: 10.10.3(Yosemite)以降に搭載された仮想環境フレームワーク
Hypervisor.frameworkを利用して、Linuxを作成。その上でコンテナを構築

6-8. コンテナの用途

コンテナは仮想化と異なりコンパクト、起動も速い

アプリケーション単位で隔離されるため、仮想化で実現するには大がかりなことが簡単に行える

同一ツールの複数バージョンを同居：Pythonのバージョン違いを同居

分離されたデーモンプロセスの複数起動：httpdの複数起動

開発環境や検証環境をコンテナで構築し、配布することで環境の同一を保つ

……他にも様々な用途がある

6-9. コンテナのサイズ

Ubuntuのコンテナは111MByte

“Hello from Docker!” と表示するだけのhello-worldは、わずか1.85kByte
ベースとなるOSとの差異がアプリケーションコンテナとして格納されている
ので、一般にコンテナのファイルサイズは小さい

```
$ docker images
```

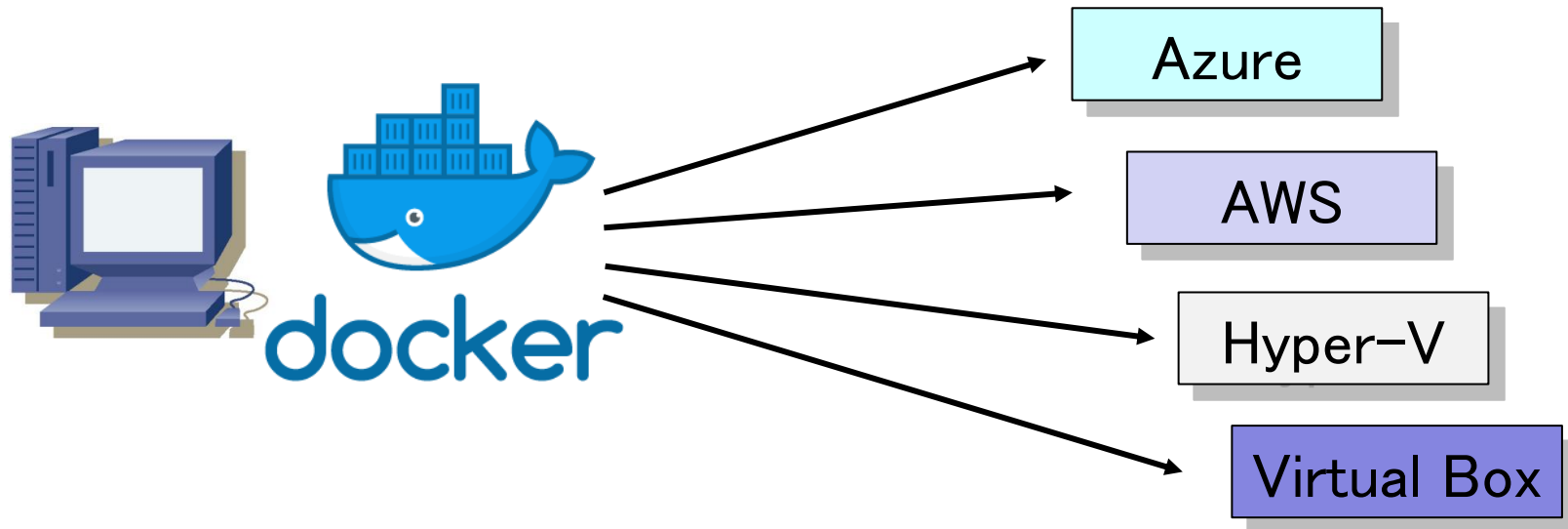
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	00fd29ccc6f1	11 days ago	111MB
hello-world	latest	f2a91732366c	5 weeks ago	1.85kB

6-10. Dockerのプロビジョニング

プロビジョニング：必要に応じてコンピュータ・リソースを提供・準備すること、または自動構築すること

Docker Machineによって仮想環境のHyper-VやVirtual Box, クラウド環境のAmazon EC2やMicrosoft Azureなどに展開可能

ローカルだけではなく、クラウド環境でもサポートされている



第7章 コンテナの実践

Dockerを使用してコンテナを学
ぶ

7-1. 演習

演習4 : Dockerのインストール&Hello Docker

演習5 : コンテナのカスタマイズ、独自コンテナの実施

演習6 : Docker Hubへ独自コンテナの公開

仮想化講座

■ e-learningの目的

- ・ 講義・演習を始める前に、仮想化という概念や基礎的な用語について学ぶことを目的としています。演習を円滑かつ効果的に進めていくためにも、本e-learningで基礎知識をしっかりと構築していきましょう。

■ 講義・演習を始める前の基礎知識

- ・ 仮想化概要
- ・ 仮想化の構成要素

★第1章：仮想化概要

- 1-1 仮想化とはなにか
- 1-2 仮想化の種類
- 1-3 仮想化製品
- 1-4 サーバ仮想化
- 1-5 デスクトップ仮想化
- 1-6 仮想化とクラウド

★第2章：仮想化概要

- 2-1. ネットワークとは
- 2-2. LAN
- 2-3. WAN
- 2-4. IPアドレス
- 2-5. ネットワーク部とホスト部
- 2-6. IPアドレスのクラス

仮想環境の構成要素

- 2-1 仮想化のコンポーネント
- 2-2 仮想マシン
- 2-3 仮想CPU
- 2-4 仮想メモリ
- 2-5 仮想HBA、仮想ディスク
- 2-6 仮想NIC
- 2-7 仮想スイッチ、仮想ルータ

第1章：仮想化概要

- プロセッサやメモリ、ディスク、通信回線など、コンピュータシステムを構成する資源（リソース）を物理的構成に拠らず柔軟に分割したり統合したりすること
- 1台のサーバコンピュータをあたかも複数台のコンピュータであるかのように論理的に分割し、それぞれに別のOSやアプリケーションソフトを動作させる「サーバ仮想化」や、複数のディスクをあたかも1台のディスクであるかのように扱い、大容量のデータを一括して保存したり耐障害性を高めたりする「ストレージ仮想化」などの技術がある

IT用語辞典より転載

仮想化とは、物理的なハードウェアのリソースを論理的（ソフトウェア的）に分割したり、統合したりする技術全般を指します。

ハードウェアの進歩により、サーバをはじめ様々なハードウェアの持つリソースが高性能、大容量、高速になってきているため、そのリソースをひとつのシステムでは使い切れず、リソースが効率的に使用されている状態とはいえなくなっています。そこで、その物理的なリソースの未使用部分（余剰リソース）を有効活用するために、論理的にリソースを分割し、あたかも複数台のハードウェアを利用しているように見せたり、逆に複数台の物理的リソースを統合して、1台の高性能、大容量、高速なハードウェアを利用しているように見せる技術を指します。

代表的な仮想化技術としてサーバ仮想化があります。サーバ仮想化とは、物理的には1台のサーバの未使用リソースを利用して、あたかも複数台のサーバがあるように見せかけて、異なる複数のOSを走らせたり、ひとつのサーバに処理が集中することを避けるために、処理を分散させ、効率よく処理を進めたりする技術です。そのほか、ストレージ仮想化、デスクトップ仮想化、ネットワーク仮想化などの仮想化技術があります。それぞれの仮想化技術の概要をみていきましょう。

■ 仮想化は次の種類に分けられています。

- サーバ仮想化
- デスクトップ仮想化
- ネットワーク仮想化
- ストレージ仮想化

仮想化の技術は大きく 4 つの種類に分けられます。

[サーバ仮想化]

サーバ仮想化は、前ページでも触れたように 1 台の物理サーバを論理的に分割し、複数台のサーバがあるように見せかけて利用する技術です。論理的に分割された複数台の仮想サーバは、あたかもそれぞれ独立したサーバのように利用することができ、物理サーバに導入されている OS とは別の OS を走らせたり、別のアプリケーションを利用することができます。物理的なサーバの高性能化によってできた余剰リソースを利用して、複数台のサーバを運用していると同様の運用ができるようになります。これによって、メインの物理サーバでは対応できないアプリケーションの運用や、テストなどにも利用することが可能となります。また、論理的に構成されたサーバなので、バックアップを利用して物理的故障に対

しての保全も簡単に行うことができます。

[デスクトップ仮想化]

デスクトップ仮想化は、サーバにおかれたPC環境をPCの利用者が使用する端末に反映させて利用させる技術です。利用者が使用するPCには、アプリケーションやデータが置かれずに、PCからサーバにアクセスしてPCを利用する仕組みとなります。ネットワークに接続する環境があれば、認証を受けた利用者は、いつでも、どこでも、どのPCからでも、アプリケーションやデータを利用することが可能となります。運用サイドの視点からは、アプリケーションやデータがサーバ側にあるために、利用者や利用状況の管理が簡単になるとともに、データも利用者のPCに残らないので、データ流出を未然に防ぐセキュアな環境といえます。利用者サイドの視点からは、いつでも、どこでも、同じ環境でPCを利用できるメリットがあります。

[ネットワーク仮想化]

一般的にネットワーク仮想化とは、SDN (Software-Defined Networking)のことを指している場合が多いですが、正確な定義づけがされているわけではありません。ネットワーク仮想化とは、物理的に構成されたネットワークを分割して利用したり、複数のネットワークを統合することによって、仮想的なネットワークを構成する技術を指します。サーバやストレージなどをネットワーク上で利用する際に、専用のネットワーク仮想化ソフトウェアを利用してネットワークを構成することによって、従来の有線ネットワークや無線ネットワークのように物理的な作業をすることなくソフトウェア的にネットワークを管理できるようにする技術です。

[ストレージ仮想化]

急激に増加してきている大量のデータを記録するために、物理的な記憶媒体を追加するのではなく、既存の複数のストレージをソフトウェア的に統合し、あたかもひとつの巨大なストレージのように見せかけ、データを保存する技術です。複数のシステムで別個に利用されているそれぞれのストレージでは、容量の足りないものと、容量の余ってしまうものが出てきます。容量の足りなくなったストレージに対して、新たに物理的なストレージを増設するのではなく、ソフトウェア的に足りないストレージと余っているストレージを統合して、巨大なストレージを仮想的につくることによって、効率的にストレージを利用することができます。

■ 代表的な製品（ハイパーバイザ）

• サーバ仮想化 (Type-I)

VMware Sphere

Microsoft Hyper-V

Xen

• デスクトップ仮想化 (Type-II)

VMware Workstation, Player, Fusion

Microsoft Hyper-V

Virtual Box

ハイパーバイザとは、物理的なサーバやPCなどのハードウェア上に仮想化技術を実行させるための制御プログラム全般のことを指します。それに対して、スーパーバイザという言葉があります。ここでいうスーパーバイザとは、ハードウェアを制御するオペレーティング・システム（OS）のことを指します。

現在、ハイパーバイザは様々なベンダーから提供されていますが、このハイパーバイザには大きく分けて、サーバ仮想化（Type-I）とデスクトップ仮想化（Type-II）の2つのタイプがあります。スライドには、それぞれの代表的な製品を挙げています。サーバ仮想化（Type-I）とは、仮想化が実行されるハードウェア上で直接稼働するタイプの仮想化技術のことを指します。それに対して、デスクトップ仮想化（Type-II）とは、ハードウェアを制御しているOS上で稼働するタイプの仮想化技術を指します。

それぞれの特徴を次のページからみていきましょう。

■ Type-I型、ネイティブ型、ベアメタル型

- ハードウェア上で直接動作するハイパーバイザ
- OSはすべてハイパーバイザ上で動作し、ほぼネイティブ環境に近い動作速度が得られる



Type-Iのハイパーバイザの特徴は、ハードウェアに直接インストールし、仮想環境を稼働させるところにあります。ホストOSを介せずに直接仮想環境からハードウェアを制御することが可能なため、ハードウェアの能力を有効に利用することができます。狭義では、ハイパーバイザとはこのType-I型のハイパーバイザのことを指します。

■ Type-II型、ホストOS型

- ハードウェア上にまずOSが動作し、その上で1アプリケーションとしてハイパーバイザが動作



Type-II型のハイパーバイザは、ハードウェアを制御しているOSの上のアプリケーションとしてハイパーバイザを稼働させるタイプを指します。ハードウェアを制御しているOSをホスト、ハイパーバイザ上で稼働している仮想環境上のOSをゲストと呼ぶこともあります。このタイプの特徴は、すでに稼働しているハードウェアとOSにインストールすることも可能なので、手軽に導入することができる反面、ゲストOSからのハードウェア制御には、ホストOSの経由が必要なため、余分な負荷がかかり、ハードウェアの持つ本来の性能を十分に利用できない場合があります。

■ クラウドサービスでは仮想化環境を提供

- ・ クラウドサービスはユーザが自由にリソースを変更可能。ニーズにマッチした仮想化環境が利用できる

	suse-sles-12-sp3-v20171212-hvm-ssd-x86_64 - ami-8368cefb SUSE Linux Enterprise Server 12 SP3 (HVM, 64-bit, SSD-Backed) ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	RHEL-7.4_HVM_GA-20170808-x86_64-2-Hourly2-GP2 - ami-9fa343e7 Provided by Red Hat, Inc. ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20171121.1 - ami-0def3275 Canonical, Ubuntu, 16.04 LTS, amd64 xenial image build on 2017-11-21 ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい
	Windows_Server-2016-English-Full-Base-2017.11.29 - ami-f6d8008e Microsoft Windows Server 2016 with Desktop Experience Locale English AMI provided by Amazon ルートデバイスタイプ: ebs 仮想化タイプ: hvm ENA 有効: はい

クラウドサービスとは、利用者が手元の端末からネットワーク経由でデータやアプリケーションを利用するサービスの総称で、クラウドサービスを利用するための環境が整った端末があれば、いつでも、どこでも、クラウド上の様々なサービスを利用できるメリットがあります。クラウド側のリソースを、利用者が自由に使えるのでニーズにマッチした仮想化環境を構築できます。現時点では主に3つのサービスに分類されています。

SaaS (Software as a Service) : ネットワーク経由でソフトウェアの機能を提供するサービス

PaaS (Platform as a Service) : ネットワーク経由で仮想化されたアプリケーション実行用のプラットフォームを提供するサービス

IaaS (Infrastructure as a Service) : ネットワー

クラウド経由で仮想化されたハードウェアやインフラストラクチャを提供するサービス

図は、Amazonのクラウドサービス「Amazon EC2」でテンプレートとして用意されている仮想マシンの一例です。

第2章 ネットワークの基本

- 仮想化技術を習得する上で最低限の知識

2-1. ネットワークとは

- ネットワークとは、網という意味の英単語。複数の要素が互いに接続された網状の構造体のこと。ネットワークを構成する各要素のことを「ノード」(node)、ノード間の繋がりのことを「リンク」(link)あるいは「エッジ」(edge)と言う。

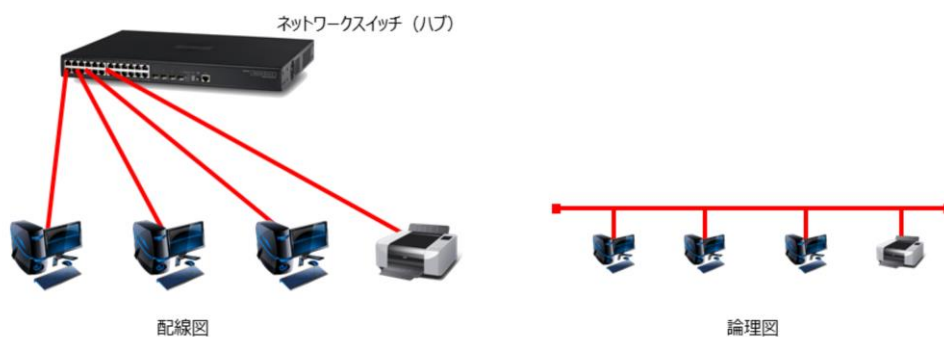
- 一般の外来語としては人間関係の広がりのことや、組織や集団の構造などを指すこともあるが、IT関連の分野で単にネットワークという場合は、複数のコンピュータや電子機器などを繋いで信号やデータ、情報をやりとりすることができるコンピュータネットワークあるいは通信ネットワークのことを意味することが多い。

■ IT用語辞典より転載

2-2. LAN

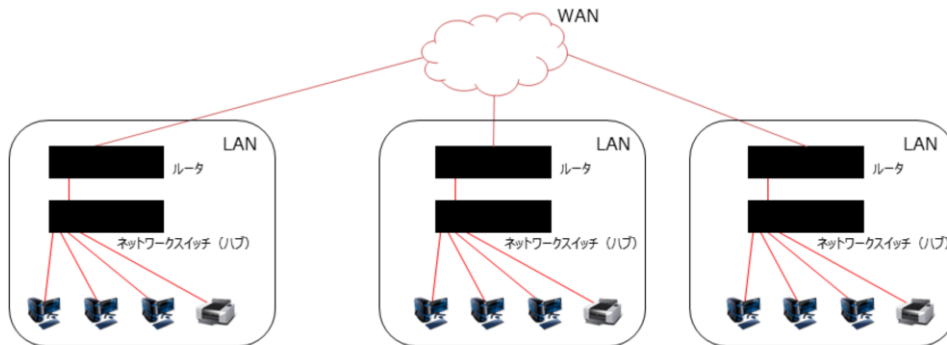
■ LAN (Local Area Network)

- 1つのフロア, 組織, 部署といった, 比較的狭い範囲でのネットワーク



■ WAN (Wide Area Network)

- 広域通信網 (Wide Area Network) の略。
LANとLANを結ぶ公衆網のことを指す場合が多い。
WANを世界規模で実現しているのがインターネット。



2-4. IPアドレス

- ネットワーク上で機器の識別をするための番号
- 32bit でIPアドレスを管理 (IPv4, Internet Protocol Version 4)
 - bit … 情報量の最小単位, 1 or 0
- ネットワーク部とホスト部の存在

192.168.1.10

10進 → 2進に変換

$$192_{(10)} = \boxed{1100\ 0000} \quad (2)$$

$$168_{(10)} = \boxed{1010\ 1000} \quad (2)$$

$$1_{(10)} = \boxed{0000\ 0001} \quad (2)$$

$$10_{(10)} = \boxed{0000\ 1010} \quad (2)$$

192.168.1.10

1100 0000 1010 1000 0000 0001 0000 1010

2進数の並びが $8 \times 4 = 32$ コ



32bit

※ 8bitのかたまり = 1オクテットと呼ぶことも

octet

192.168.1.10/24

この例だと、IPアドレスのビット列の、

先頭から24bit = ネットワーク部

残りの8bit = ホスト部

1100 0000 1010 1000 0000 0001 0000 1010

24bit : ネットワーク部

8bit : ホスト部

ネットワークプレフィックスと言うことも、
network prefix

ネットワーク内の端末を
識別

端末がどのネットワークに所属しているのかを判別.

ひとつのネットワークに収容できるホスト数

ホスト部がすべて 0 ⇨ ネットワーク自身のことを表す

ホスト部がすべて 1 ⇨ そのネットワーク全体にデータを送る
ブロードキャストアドレスを表す

192.168.1.10 ならば…

ネットワーク自身 ⇨ 192.168.1.0

1100 0000 1010 1000 0000 0001 0000 0000

すべて 0

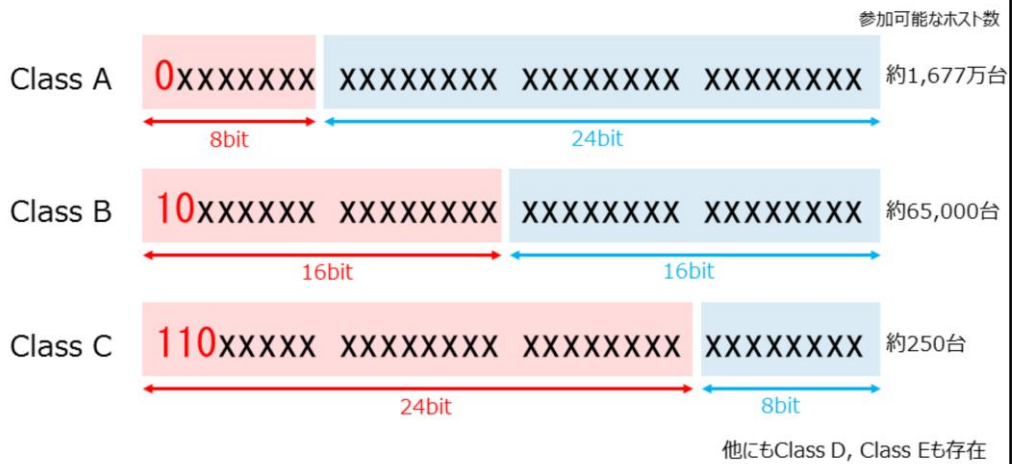
ブロードキャストアドレス ⇨ 192.168.1.255

1100 0000 1010 1000 0000 0001 1111 1111

すべて 1

2-6. IPアドレスのクラス

- 32bit中、「何bitがネットワーク部で、何bitがホスト部を表すか」
- ネットワークの規模を識別する



- (例題) 次のIPアドレスのクラス (ネットワークの規模) は？

172.16.0.1

10進 → 2進に変換

$$172_{(10)} = \boxed{1010\ 1100}_{(2)}$$

先頭ビットが 10… なので, 「クラスB」であることがわかる

第3章：仮想環境の構成要素

■ 主なコンポーネント

- 仮想マシン
- 仮想CPU
- 仮想メモリ
- 仮想HBA
- 仮想ディスク
- 仮想NIC
- 仮想スイッチ

仮想環境を実行するためには、その仮想環境を構成する仮想コンポーネントが必要です。ノートPCを動かすためには、ノートPCを構成しているCPUやメモリ、HDDやSSDのようなストレージ、各種物理スイッチなどのパーツが必要なように、仮想環境でもそれぞれの役割を担ったパーツが必要です。仮想環境を実行するために必要な主なコンポーネントは以下の通りです。

[仮想マシン]

サーバやデスクトップPC、ノートPC、タブレットなど、OSやアプリケーションソフトウェアを動かすために必要なハードウェアの環境を、仮想的にハイパーバイザ上で再現したマシンを指します。ソフトウェア的に再現されたPCと考えると分かりやすいかもしれません。

[仮想CPU]

実際に稼働しているハードウェアとしての物理CPUのリソースを分割し、ハイパーバイザ上で稼働している仮想マシンのCPUとして構成されたものを指します。

[仮想メモリ]

ここでいう仮想メモリとは、物理ハードウェア上で稼働しているOSが利用する物理メモリがリソース不足になった際に利用されるストレージのメモリ化とは異なり、ハイパーバイザが仮想マシンに割り当てて利用するメモリを指します。

[仮想HBA、仮想ディスク]

仮想マシンは、物理ハードウェア上に構成された仮想ディスクをストレージとして利用します。その仮想ディスクと仮想マシンを繋ぐ論理的な接続が仮想HBA（ホストバスアダプタ）です。

[仮想NIC、仮想スイッチ]

仮想マシンがネットワークを利用する場合は、物理ネットワーク機器と仮想マシンを間に仮想ネットワークを構成する必要があります。その仮想ネットワークを構成するには、論理的なNIC(ネットワークインターフェイスカード)が必要となり、論理的ルータ、論理的スイッチを利用します。

仮想マシン

- 物理的なPC、サーバをハイパーバイザによって仮想的な機械に置き換えたもの
- Virtual Machine : VM
- 構成ファイル、仮想ストレージなど構成される

Hyper-Vの例

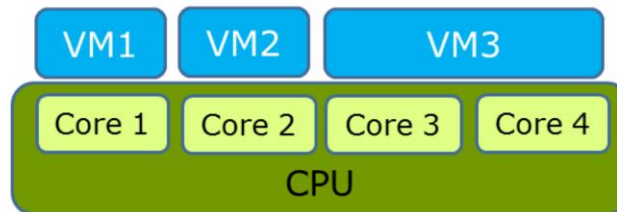
■ ubuntu 14d	2017/11/29 22:41	ファイル フォルダ	
■ ubuntu 14-desktop.vhdx	2017/12/19 14:29	ハードディスク イメ...	9,441,280 KB



仮想マシン(ハードディスクイメージ)

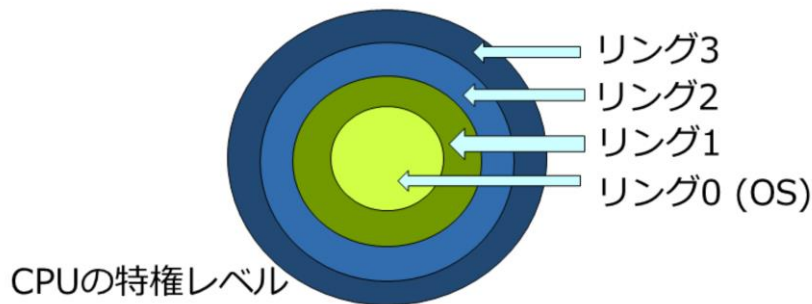
仮想マシンとは物理的なハードウェア上に論理的に構成された仮想的なコンピュータを指します。一般的にはVirtual Machine（仮想機械）の頭文字をとって「VM」と表します。仮想マシンを構成するコンポーネントをひとまとめにして、ハイパーバイザ上にフォルダやディスクイメージとして保存、活用されます。

- 仮想マシンからは仮想CPUが物理CPUとして見える
 - 仮想マシン1つにつき1つのCPU、もしくは1つのコアを割り振るのが理想的
 - 重い仮想マシン（VM3）には多くのコアを割り振り、軽い仮想マシン（VM1,VM2）には少ないコアを割り振る
 - 商用のハイパーバイザの中には、CPUのコア数、仮想CPUの総コア数でライセンスが決まるものもある



CPU（Central Processing Unit：中央演算装置）は、コンピュータの機能を実行するための主装置のひとつで、演算を実行するパーツです。CPUにはコアと呼ばれる計算を実行する心臓部があり、現在の一般的なCPUは、ひとつのCPUのなかに複数のコア（マルチコア）を備え、一度に情報を並列処理できるようになっています。仮想マシンが利用する仮想CPUは、仮想マシン1台あたりに1つのCPU、もしくは1つのコアが割り振られることが理想的です。また、動作の重たい仮想マシンや複雑な処理をする仮想マシンには複数のコアを割り振り、仮想マシン上のリソースを確保する設定も可能なハイパーバイザもあります。

- 近年のCPUには、仮想CPUを物理CPUとほぼ同等に動作させる仮想化支援機能がある
 - Intel CPU: VT-x, AMD CPU:AMD-V
 - CPUの特権レベルが最も高いリング0にアクセスできるのはOSのみ
 - アプリケーションやハイパーバイザはリング3で動作
 - リング3からリング0へのアクセスは例外が発生する



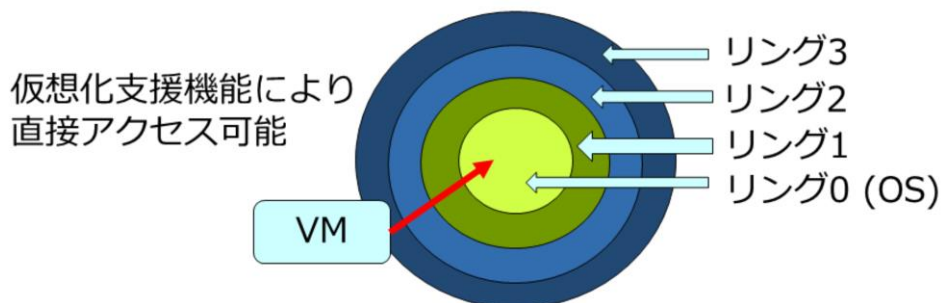
CPUは、セキュリティやシステム保護の観点から、動作モードが設定されています。この動作モードは概念的にリング状になっており、中心のリング0（特権レベル）から外側に向かってリング1、リング2、リング3のように構成されています。リング0は、別名カーネルモードとも呼ばれ、CPUの全機能が無制限に利用することができますが、通常リング0にアクセスができるのはハードウェアを制御しているOSのみです。リング1からはカーネルモードに対して、ユーザーモードと呼ばれ、アプリケーションは通常リング3で動作しています。この仕組みは、アプリケーションによって出された不当な命令によって、ハードウェアやシステムを破壊してしまうこと未然に防ぐ保護機能として働いています。

ハイパーバイザは、通常リング3で動作しています。つまり、リング3で仮想マシンによってOSが走ら

させているわけですが、そのOSが必要に応じて直接リング0へアクセスを試みると、命令矛盾が発生してしまいます。この命令矛盾を解決するために、仮想化支援機能としてCPUに新たな動作モードを搭載したCPUが開発されています。

■ VT-xやAMD-Vに対応したハイパーバイザではVMからリング0へ例外なしにアクセス可能

- I/Oもほぼ直接アクセスできるため、物理環境に近い速度で動作
- デフォルトで仮想化支援機能が無効になっているPCがあるので注意



仮想化支援機能を利用すると、通常リング3で動作しているハイパーバイザからのリング0へのアクセスが可能となり、物理CPUを監視しているOSを介さずに直接CPUを動作させることができるようになります。その結果、オーバーヘッドを回避することができるようになり、物理CPUが持つ本来の性能を仮想マシンでも利用することが可能になります。

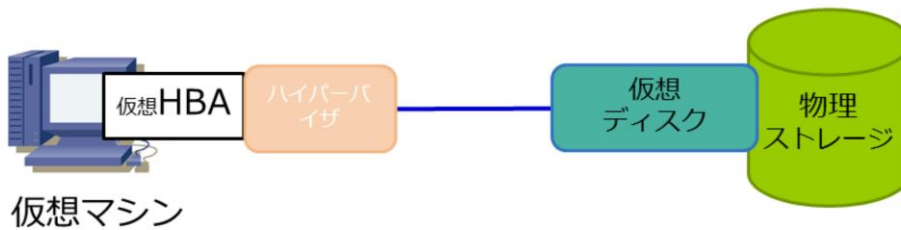
■ 仮想マシンの動作には当然メモリが必要

- ハイパーバイザによって最大メモリ量が異なる
- 仮想マシンへの仮想メモリとして割り当てる際に、完全固定として割り当てる方法（スタティック）と、最小容量と最大容量を定めておき、動的に変更できる方法がある
- Type-I ハイパーバイザ：仮想マシンがほぼすべてのメモリを使用可能
- Type-II ハイパーバイザ：他アプリケーションが使用するメモリ容量を考慮して仮想メモリを設計する

仮想マシンの動作に必要なメモリは、ハイパーバイザの種類によって利用できる領域が異なります。

Type-I ハイパーバイザの場合、ハイパーバイザから直接メモリへのアクセスが可能のため、ハードウェアの搭載されている物理メモリのほぼ全領域を利用することができます。しかし、Type-II ハイパーバイザの場合は、そのハイパーバイザを走らせているホストOSや、その他のアプリケーションが利用しているメモリ領域を考慮して、仮想マシンの仮想メモリ領域が決められます。どちらも場合も、物理メモリ内に固定された領域を仮想メモリとして仮想マシンに割り当てる方法（スタティック）と、必要に応じて、メモリ領域を変化させることが可能な設定があります。メモリ領域を動的に増減できるように設定した場合、物理ハードウェアのリブートが必要になることがあります。

- 仮想マシンは仮想ディスクをストレージとして使用する
- 仮想マシンには、ホストバスアダプタ (HBA) としてIDEやSCSIのインターフェイスが接続され、各方式の内蔵ディスクとして見える
- 仮想ディスクは動的構成を取ることができる
- 設定上120Gの仮想ディスクを使用しても、実際に使用しているサイズのみ物理ストレージ上では消費する



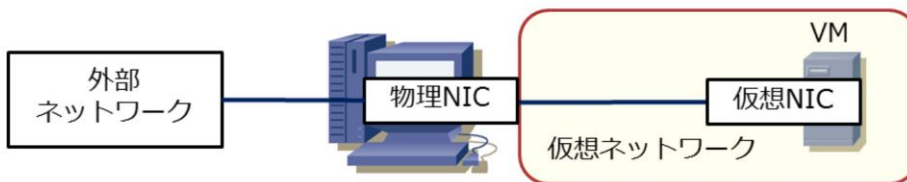
仮想マシンのストレージは、仮想HBAをインターフェースとした仮想ディスクを利用します。

仮想ディスクは物理ストレージと異なり、動的構成をとることができます。ここでいう動的構成とは、仮想ディスクのメリットののひとつであり、仮想マシンから見えている仮想ディスクのサイズと、実際に物理ディスクの上で消費されているストレージのサイズが異なるように構成することです。仮に120GBの仮想ディスクを繋いだ仮想マシンがあるとします。仮想マシンからみれば、そのストレージ領域として120GBが確保され、あたかも120GBの物理ディスクが繋がれているように構成されています。ストレージは、使用されている領域と未使用の領域がありますので、120GBの仮想ディスクも実際にデータが書き込まれ使用されている部分と、空いている部分があります。この実際には使用

していない部分は、仮想マシンからはストレージとして確保された領域として、物理ディスクからは空領域として構成できるということです。仮想ディスク上で使用されている部分のみが物理ディスク上でも使用されている領域として捉えられるので、リソースを効率よく利用することができます。

仮想NIC

- ハイパーバイザによって仮想的なインターフェイスである仮想NICが作成され、仮想ネットワークも作成される
- 物理NICと仮想NICを接続しないと、ホストOSと仮想OSは通信できない
- 物理NICと仮想NICの間に仮想ブリッジや仮想ルータを挟むことで、仮想ネットワーク形態が変わる



仮想NIC (Network Interface Card) は、仮想マシンを物理ネットワークやホストOSと通信を行うためのインターフェイスです。仮想マシンを構成する他の仮想コンポーネント同様に、論理的にネットワークインターフェイスを構成し、物理ハードウェア同様の役割をします。

- 仮想スイッチ：Virtual Switch (vSwitch)
 - 仮想ルータ：Virtual Router (vRouter)
- 仮想ネットワークにおいて、それぞれ物理機器と同じ役割を果たす
 - 仮想ネットワークをNATにする場合はvRouterが
 - VLANを作成する場合はvSwitchが必要

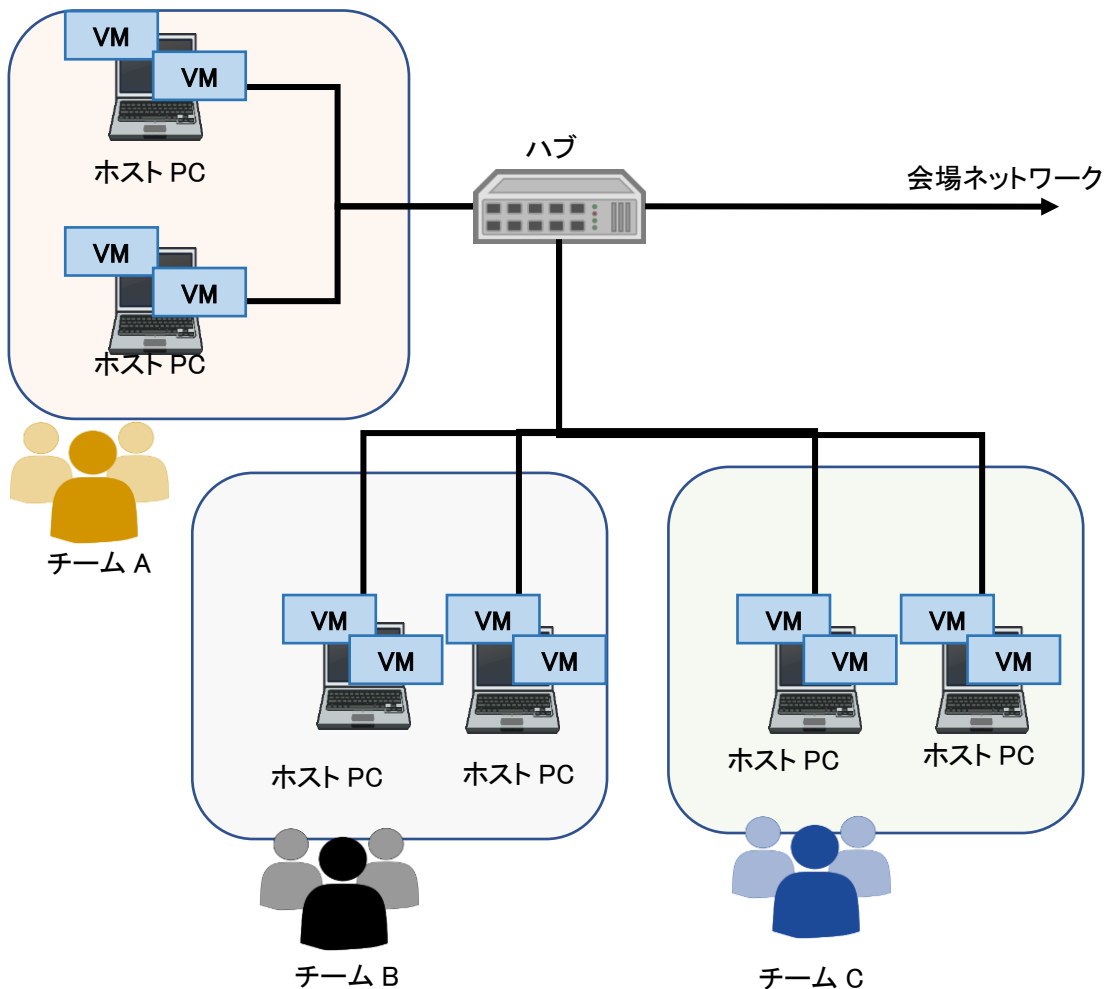


物理マシンをネットワークに接続する際に必要な仮想ネットワーク装置です。仮想マシンからは物理スイッチ、物理ルータとして見え、それぞれの役割も物理機器と同様です。インターネットプロトコルによって構成されているネットワークに接続する場合は、NAT（Network Address Translation：ネットワークアドレス変換）が必要なため、仮想ルータを利用します。また、VLAN（Virtual Local Area Network：仮想LAN）を構成する場合には仮想スイッチを利用してネットワークに接続します。

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ト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。

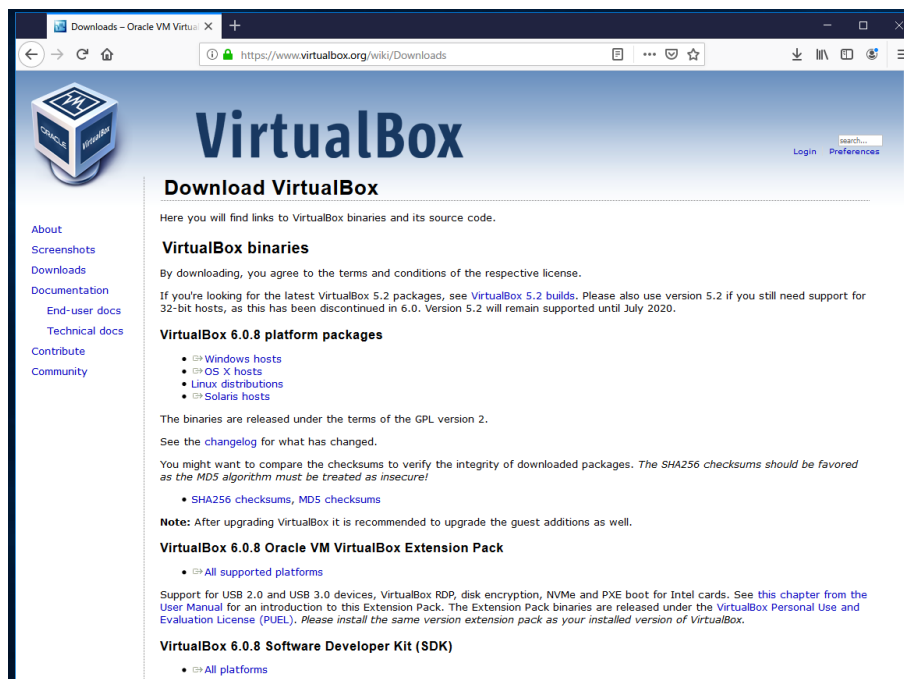
流れとして、ホスト OS の Windows に VirtualBox をインストールし、そこにゲスト OS をインストールしていきます。その後、別の端末からもネットワーク経由で操作できることを体験してもらいます。

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。

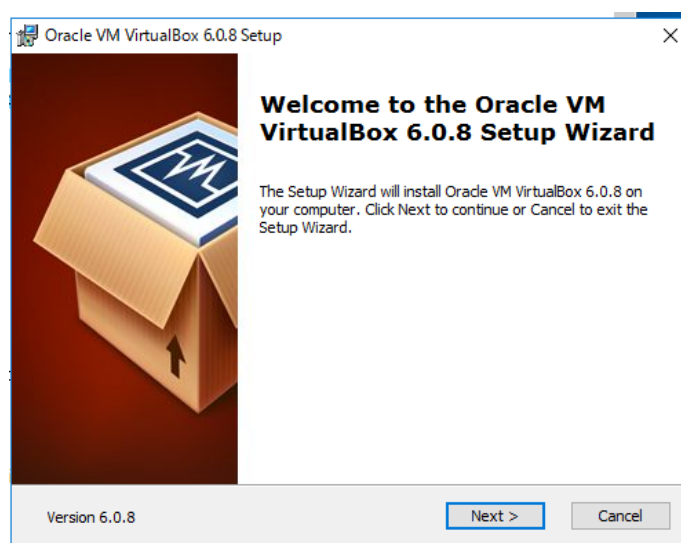
URL: <https://www.virtualbox.org/>

（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）

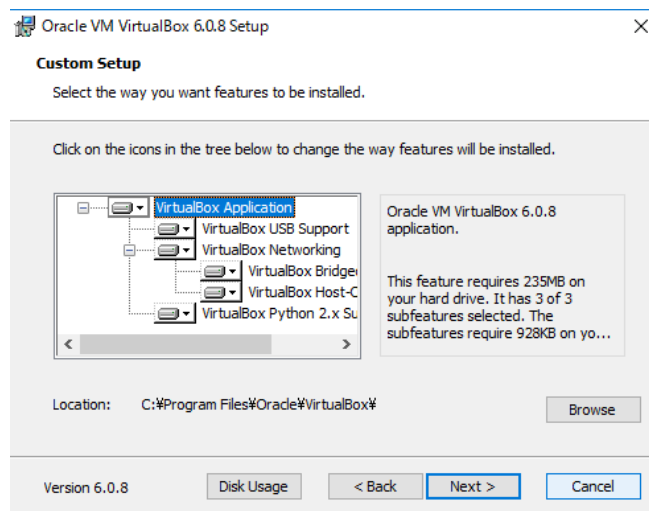


トップページにある、“VirtualBox 6.0.8 platform packages” の欄にある
“Windows hosts” から、Windows 用のインストーラをダウンロードします。
また、後ほど拡張パックも併せてインストールするため、“VirtualBox 6.0.8 Oracle VM
VirtualBox Extension Pack” の欄にある“All supported platforms” も併せてダウンロ
ードしておきます。
※ VirtualBox 6.0.8 は 2019/05/13 にリリースされたものです。

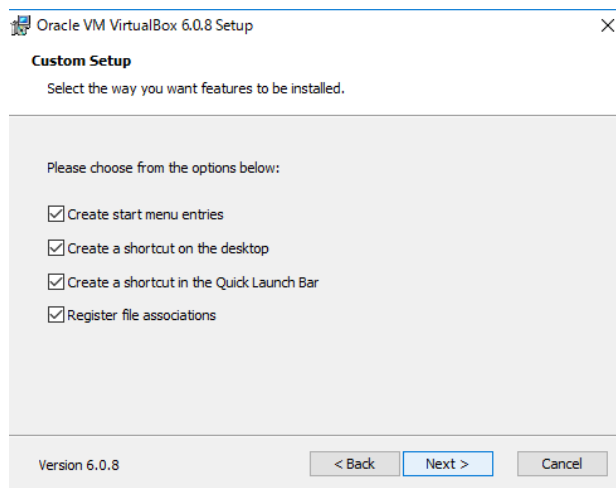
ダウンロードが完了したら、“VirtualBox-6.0.8-130520-Win.exe” を展開し、インスト
ーラを起動します。



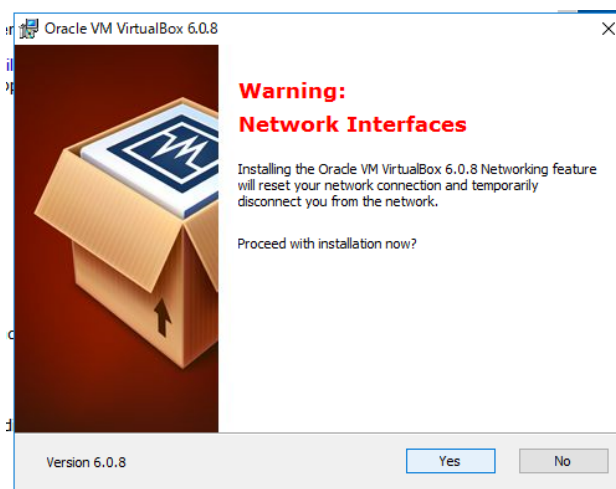
セットアップウィザードに従ってインストールを進めていきます。
特に設定を変更する必要はなく、[Next >] を押して順番に進めていきます。



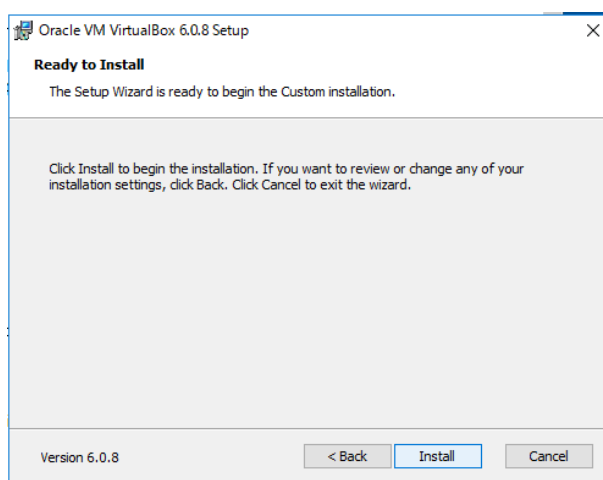
スタートメニューやショートカットをデスクトップに表示する場合は、“Custom Setup”の項目にチェックをいれて、[Next >]に進みます（原則、そのまま進めてもらってOKです）。



“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。



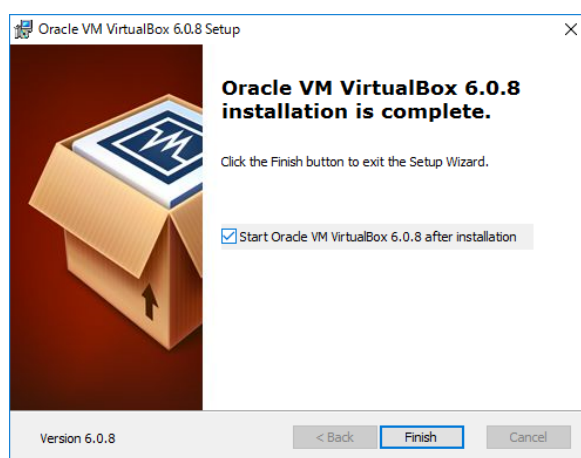
インストールの準備ができると、次の画面になります。[Install]を選んでインストールを開始します。



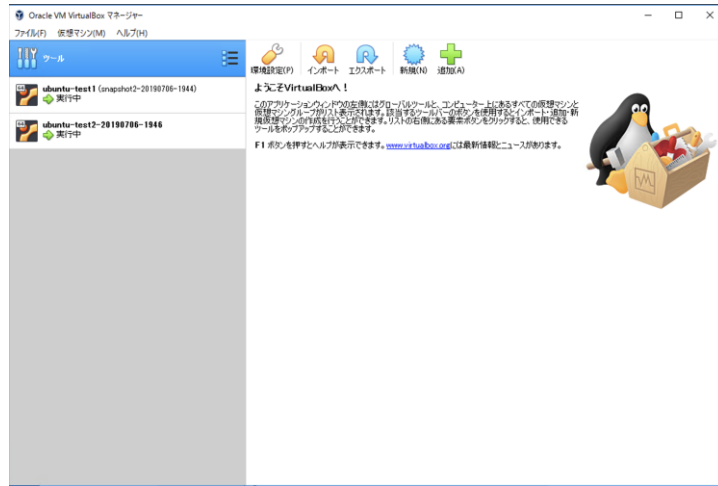
インストールの途中で USB コントローラのインストールが求められることがありますが、続けて[インストール]を選択します。



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)

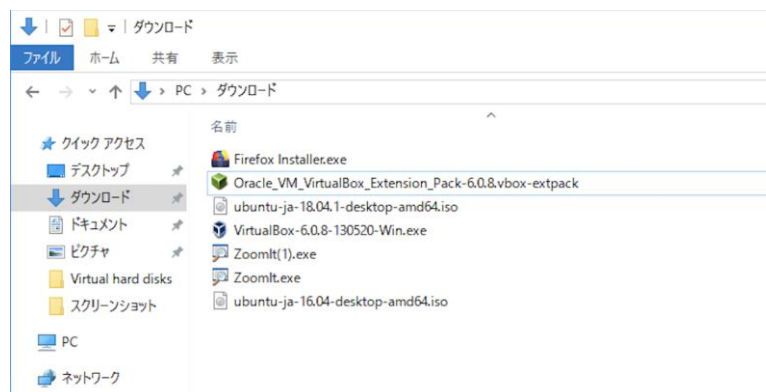


③拡張パックのインストール

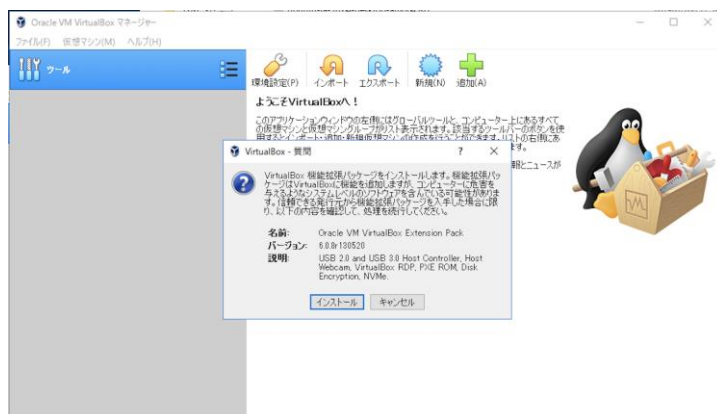
必要最低限の動作を行う場合、拡張パックの導入は必須ではないのですが、利用を便利にする機能が多く備わっていますので、導入しておきましょう。

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。

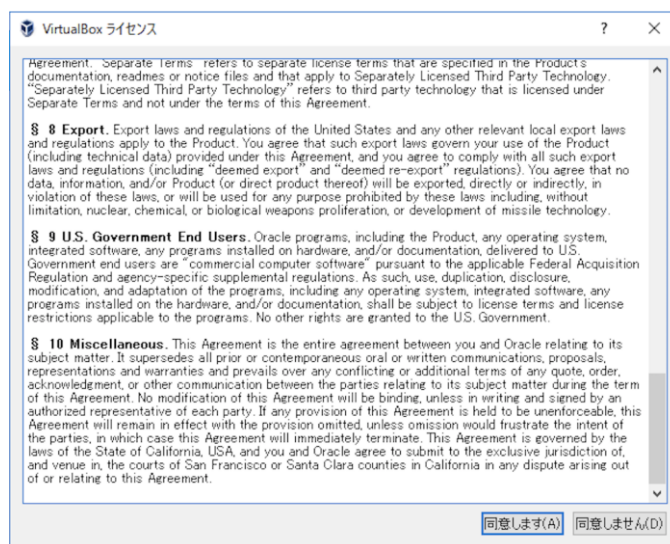
VirtualBox 本体のインストールが完了すると、②でダウンロードした“VirtualBox 6.0.8 Oracle VM VirtualBox Extension Pack”のインストールが可能になっています（ファイルの関連付けが完了しています）。当該フォルダを開き、ダブルクリックして展開します（例ではダウンロードフォルダに入っています）。



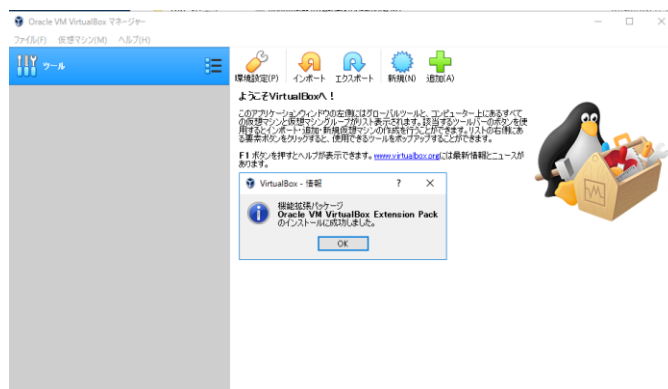
拡張パックのインストールが始まります。内容を確認し、[インストール]を選択します。



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。



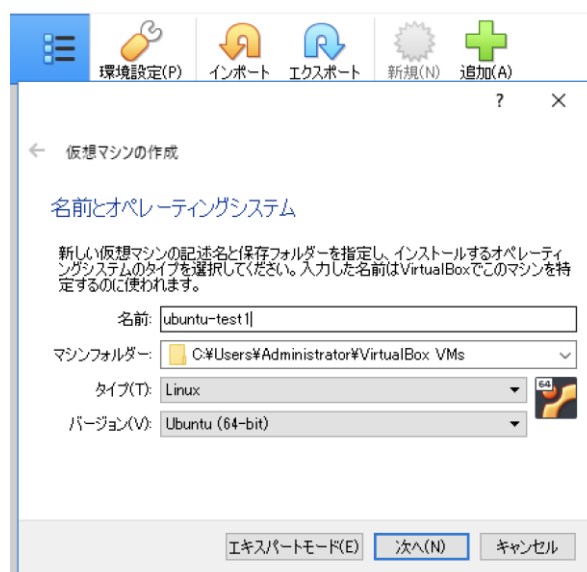
拡張パックのインストールが完了すると、「機能拡張パッケージ Oracle VM VirtualBox Extension Pack のインストールに成功しました。」と表示されます。確認の後、[OK]を選択します。



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。

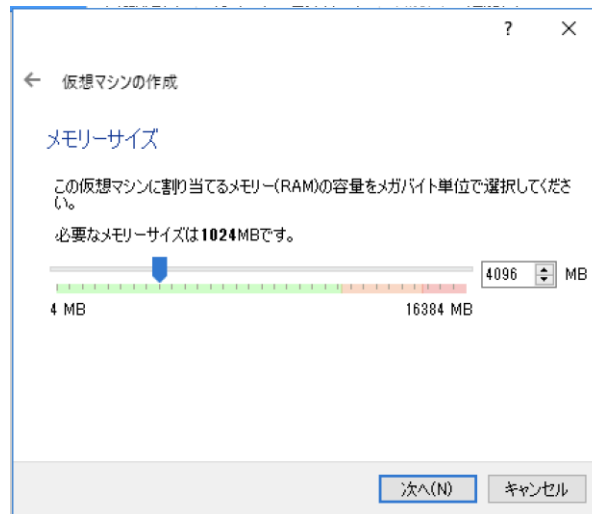


ここでは、以下のように設定します。

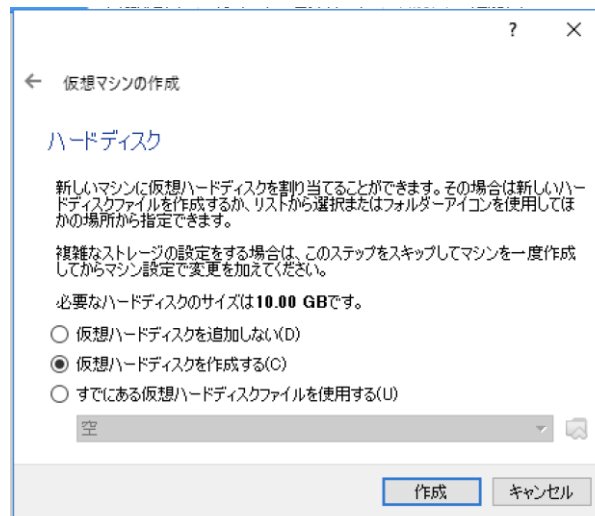
項目	内容
仮想マシンの名前	ubuntu-test1 など (任意)
マシンフォルダー	(任意)
タイプ	Linux
バージョン	Ubuntu (64-bit)

※ 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。



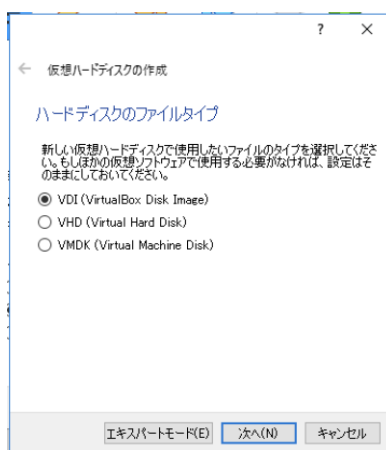
仮想マシンのハードディスクを 10GB に設定します。設定項目を「仮想ハードディスクを作成する」にします。



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。

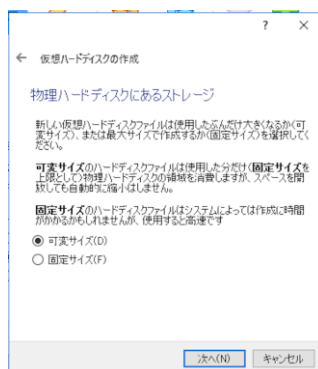
これらはいずれも互換性があります。今回は、“VDI”を選択します。



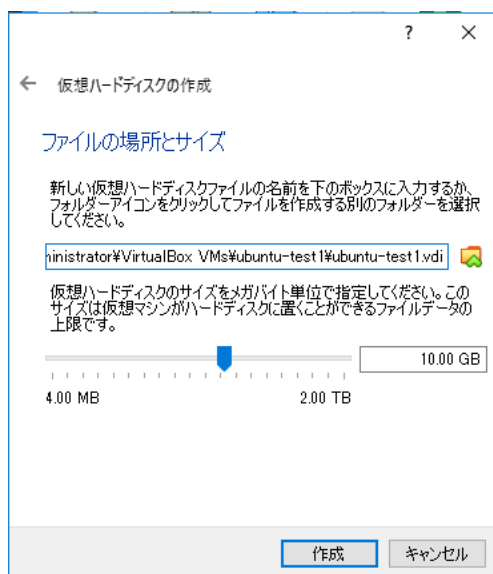
ストレージのタイプを選択します。

- ・ 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。



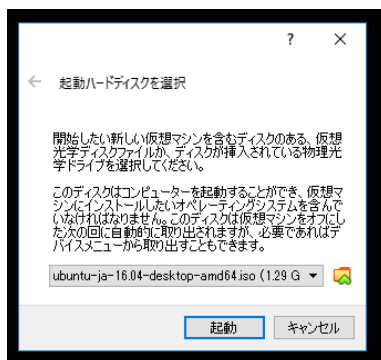
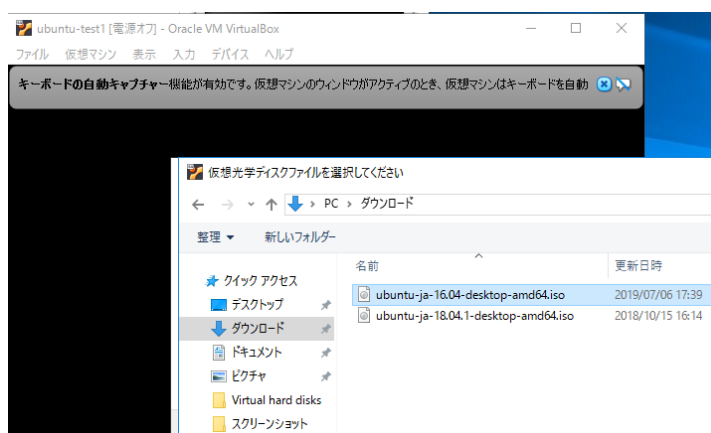
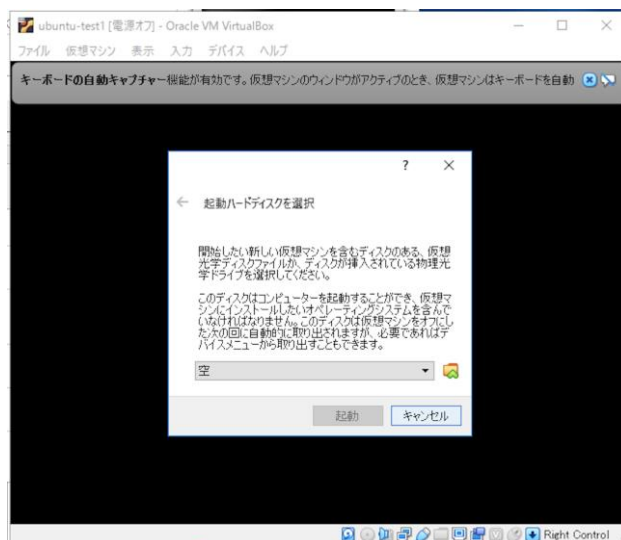
完了すると、仮想マシンの領域を作成することができます。いわゆる、OS を未インストール状態のマシンが出来上がっている状態です。初回は、何かしらのイメージから起動し、インストール作業を行います。予め [設定] からディスクイメージを認識させる方法もありますが、ディスクイメージを認識させていない場合は、起動ハードディスクをどのようにするか聞かれますので、そのまま [起動] を選択し、仮想マシンを起動させます。



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。

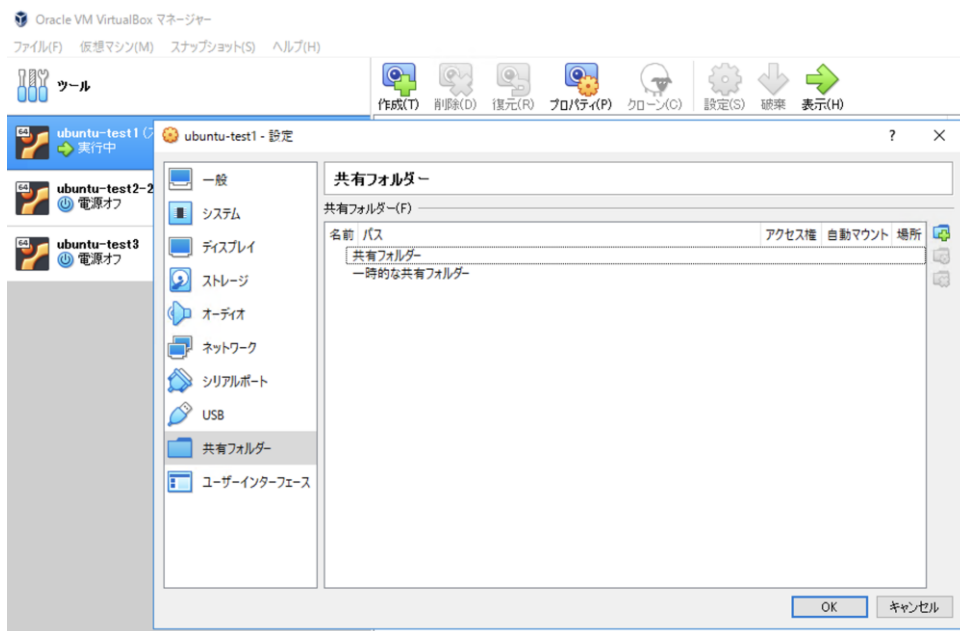
(VirtualBox Guest Additions をインストール)

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

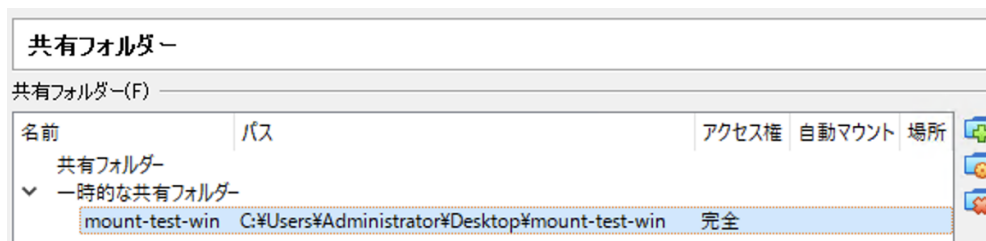
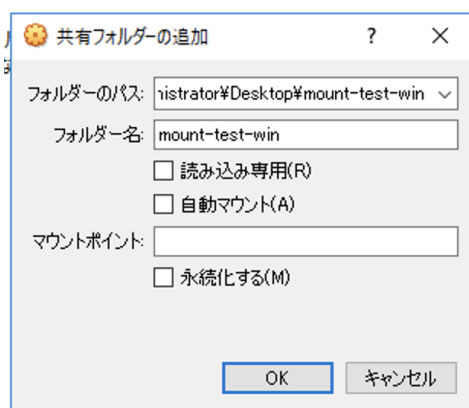
```
ubuntu@ubuntu1:~$ sudo apt install virtualbox-guest-dkms
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
以下の追加パッケージがインストールされます:
 dkms virtualbox-guest-utils
```

```
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。



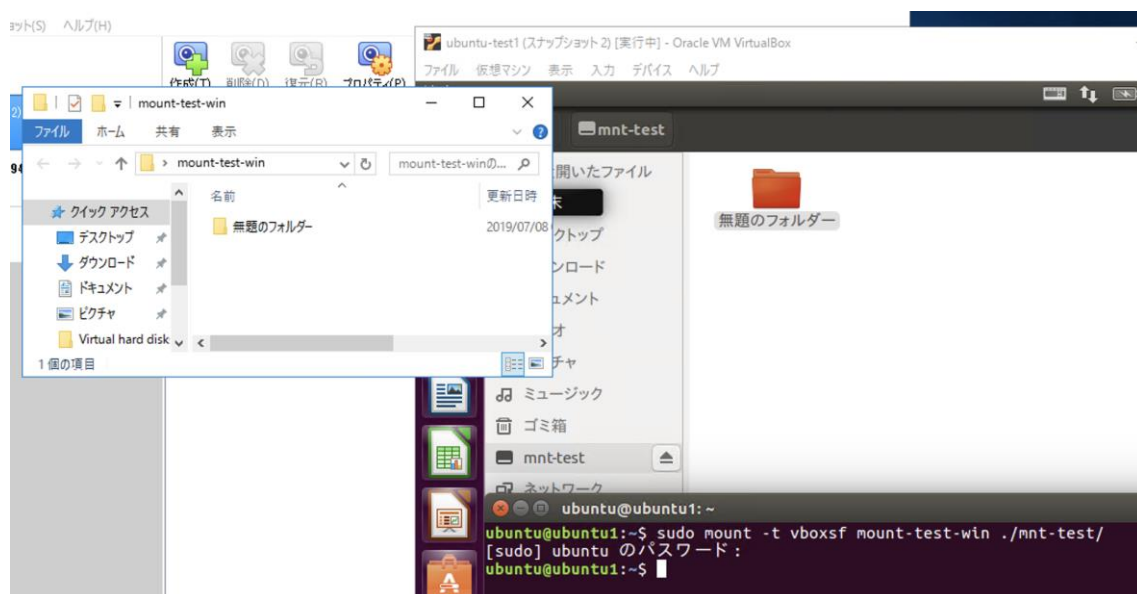
マウントポイントを更新するため、ホスト OS (Ubuntu) を再起動します。その後、ホスト OS 側の Terminal で次のコマンドを入力します。

(VirtualBox Guest Additions をインストール)

```
$ mkdir mnt-test
```

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

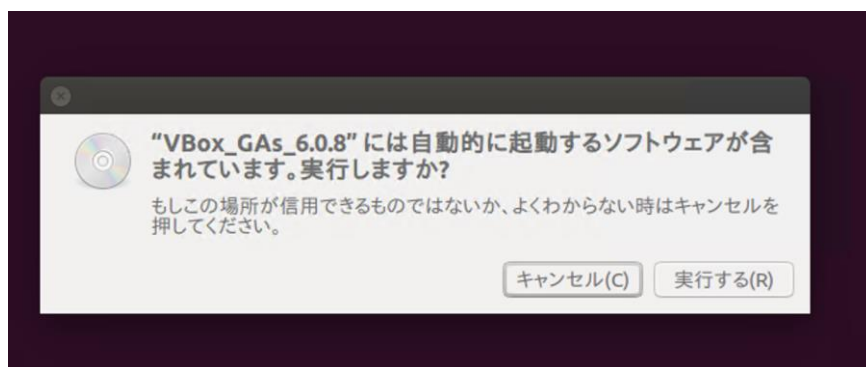
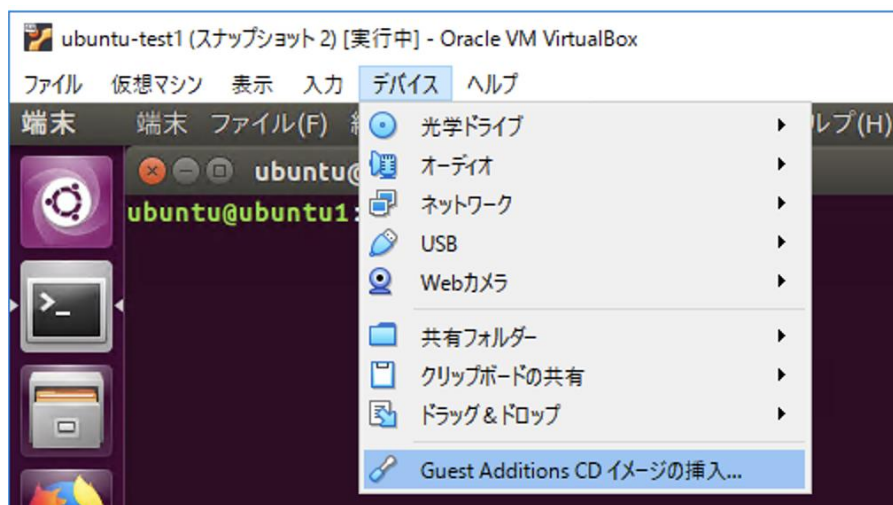
(sudo mount -t vboxsf [Windows 側の共有フォルダ名 (VirtualBox で設定したもの)] [Ubuntu 側のディレクトリ名])



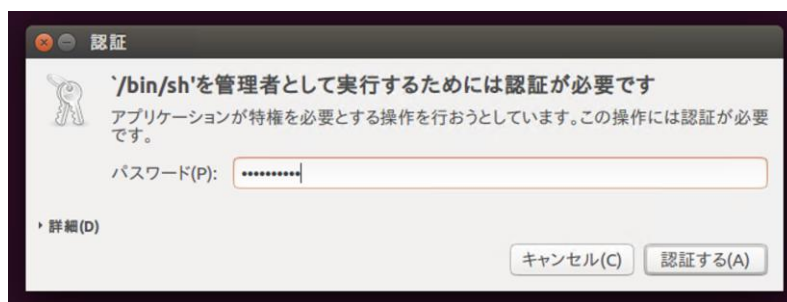
すると、ホスト OS 側の共有フォルダとゲスト OS 側のディレクトリを連携させることが可能となります。

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
```

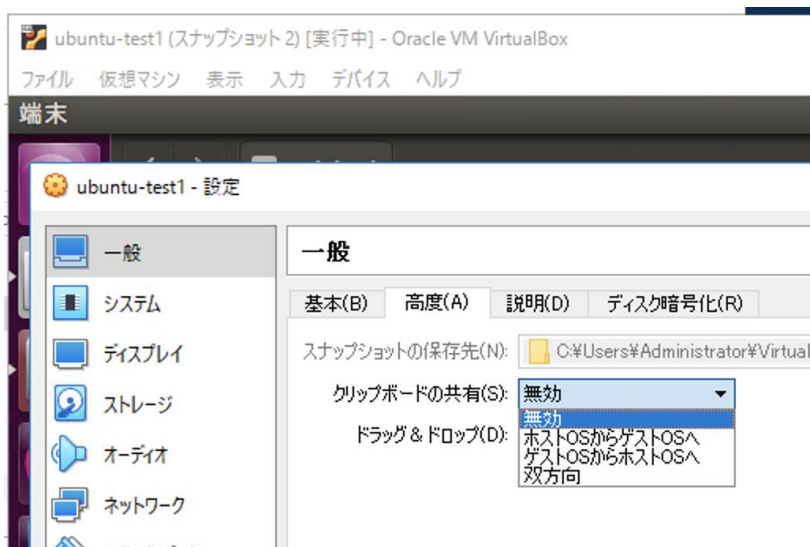
最終、 Press Return to close this window... と表示されれば成功です。

```
× ◯ ◻ 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。

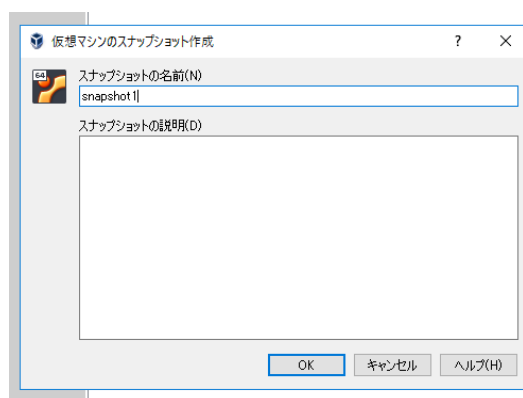
VirtualBox メニューバーにある[仮想マシン]→[ツール]→[スナップショット]と進みます。



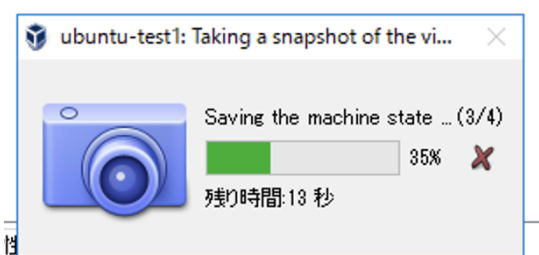
何も手を加えないうちは、最新の状態のみが存在します。[作成]を選択肢、スナップショットを作成します。



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。

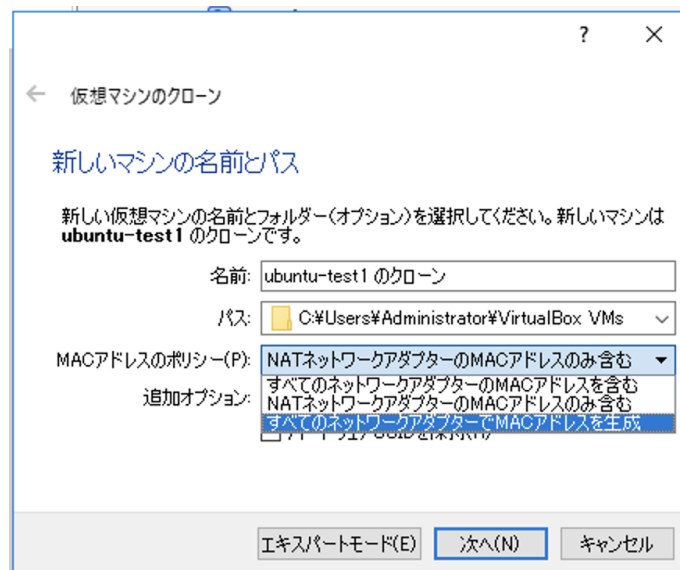
なお、ゲスト OS の電源が ON のままクローンを作成することもできるのですが、MAC アドレスの重複等が発生してしまうため、クローンを作成する場合には、ゲスト OS の電源を OFF にしてから作業を行います。

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。

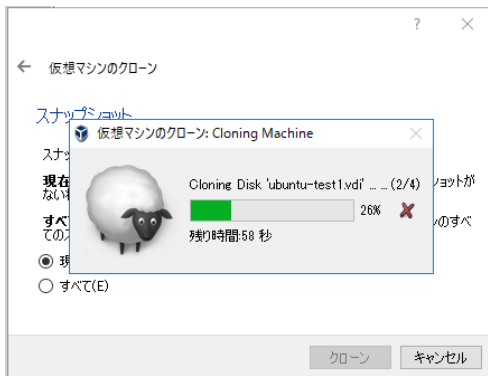
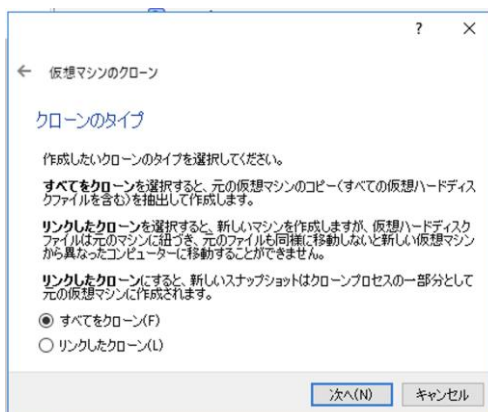
既存の仮想マシンと MAC アドレスの重複を避けるために、[MAC アドレスのポリシー]については[すべてのネットワークアダプターで MAC アドレスを生成]を選択してください。



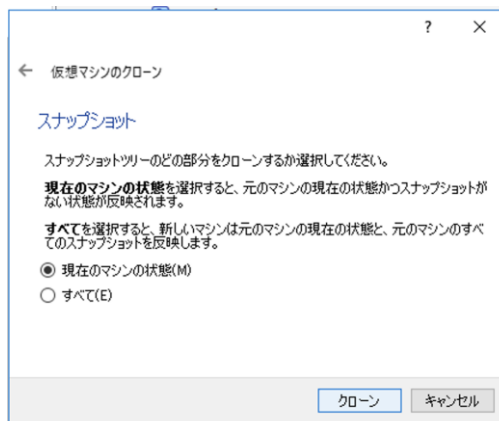
クローンのタイプを選択します。

- ・ 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。



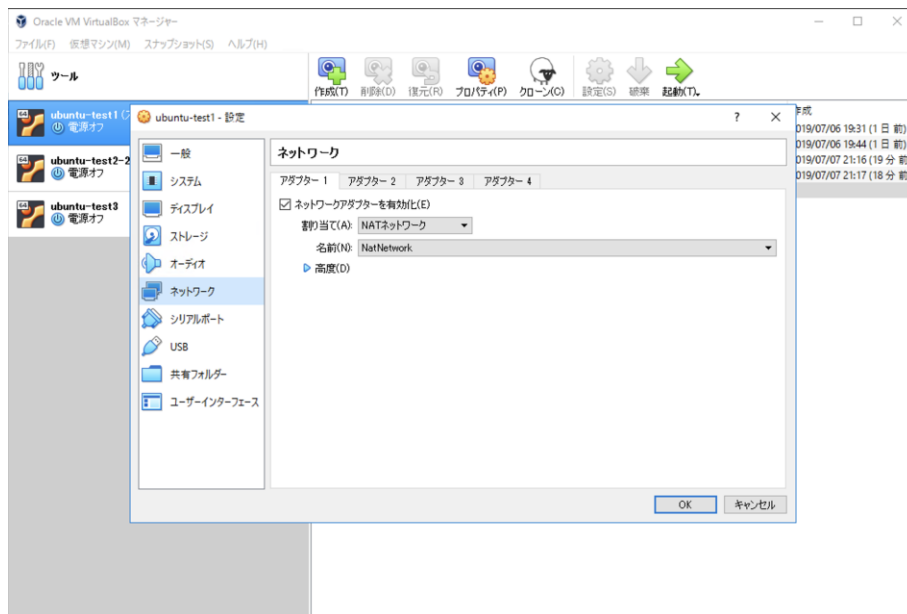
クローン生成が成功すると、左側のペインに、新しい項目が増えます。



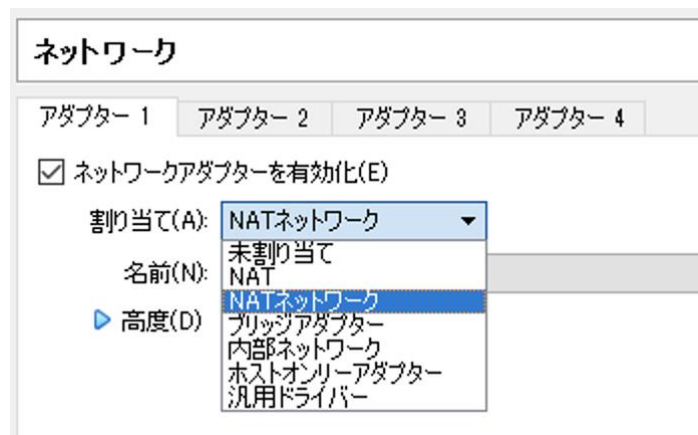
演習 3 仮想ネットワークの設定変更

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。

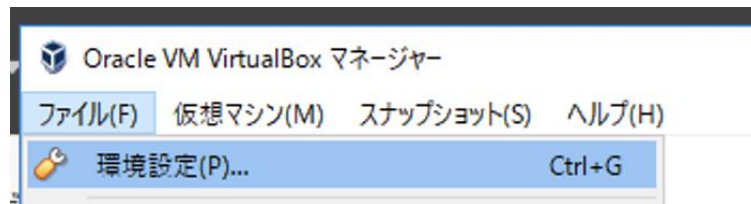


[アダプター1] (ホスト OS が利用しているネットワークアダプタ) に対して、どのようなネットワークにするかを決めます。

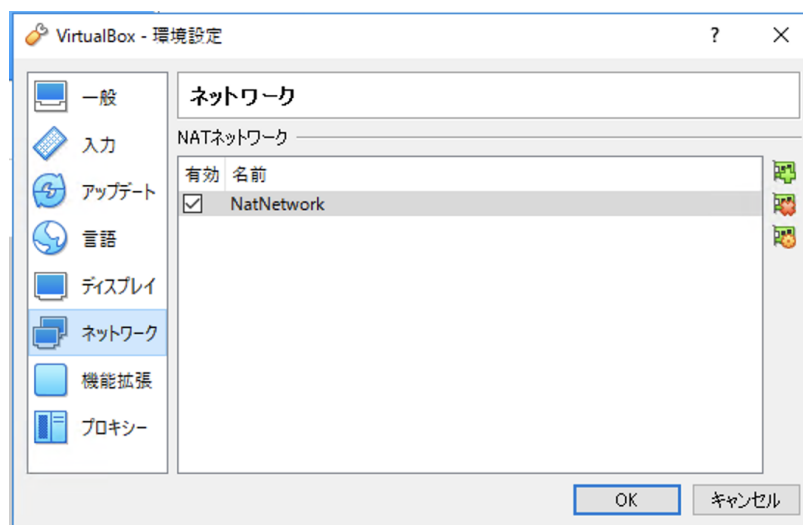



① 仮想スイッチ（NAT ネットワーク）の設定

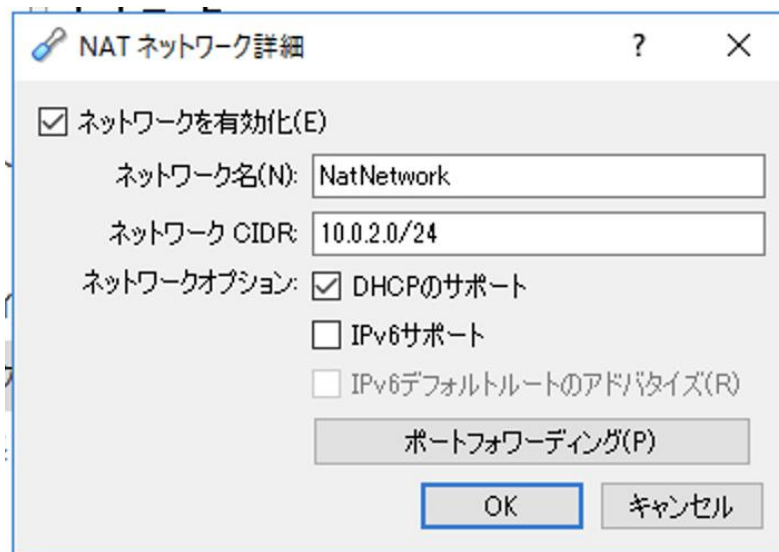
[ファイル]→[環境設定]と進み、[ネットワーク]の項目に進みます。



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。



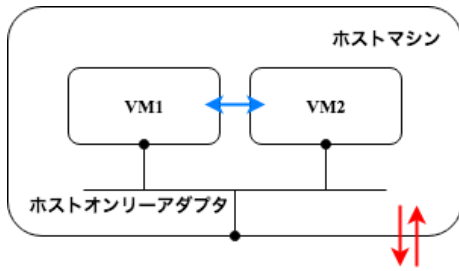
NAT ネットワーク配下の IP アドレスが割り振られていますが、DNS サーバ等は、上位のネットワークで設定されたものを継承しています。



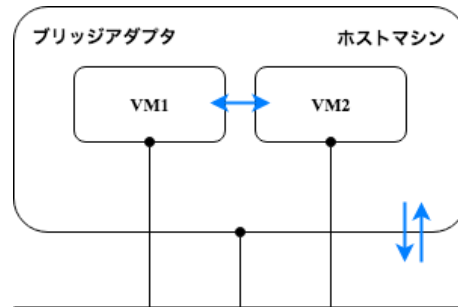
(参考) ネットワークアダプタの違い

- 通信可能
- 通信不可能

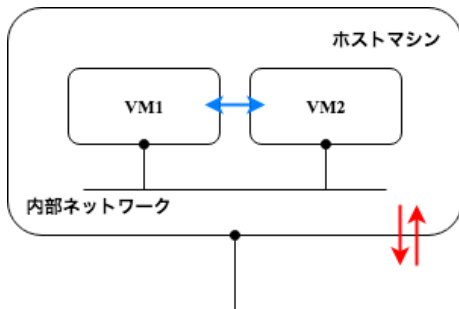
・ ホストオンリーアダプタ



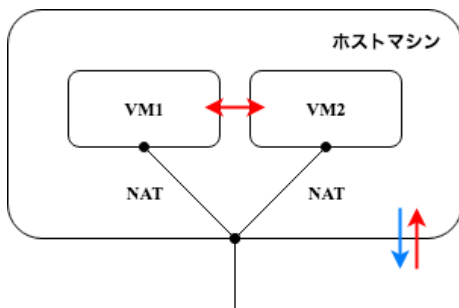
・ ブリッジアダプタ



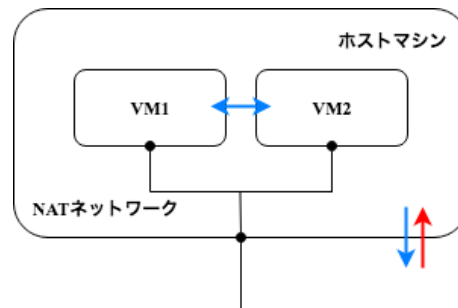
・ 内部ネットワーク



・ 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）

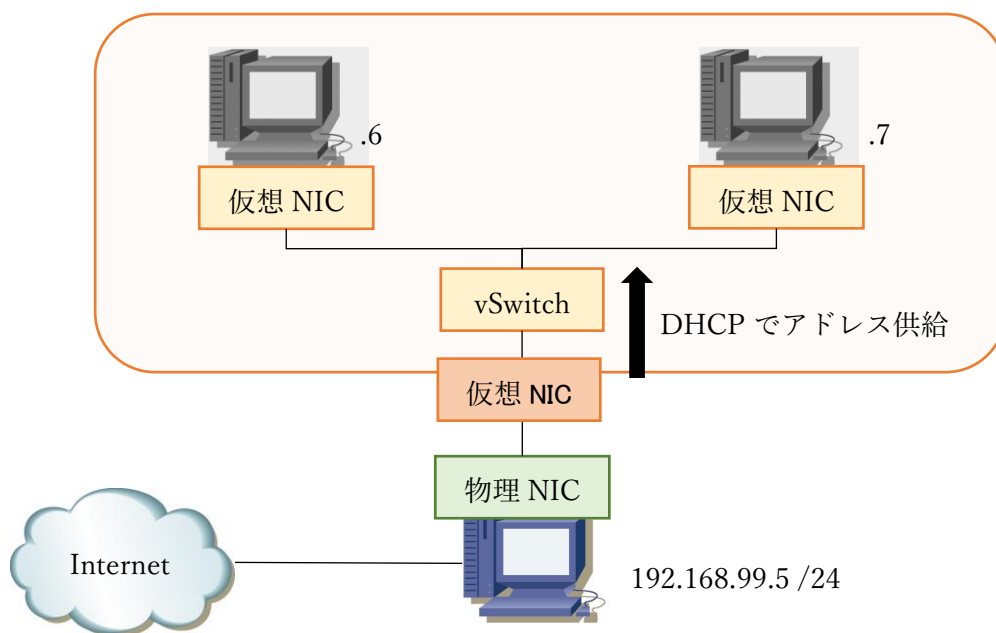
ここで、仮想マシンをもう 1 台作成し、サーバと 2 台の仮想マシンの間でネットワーク通信が可能かどうか確認します。

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



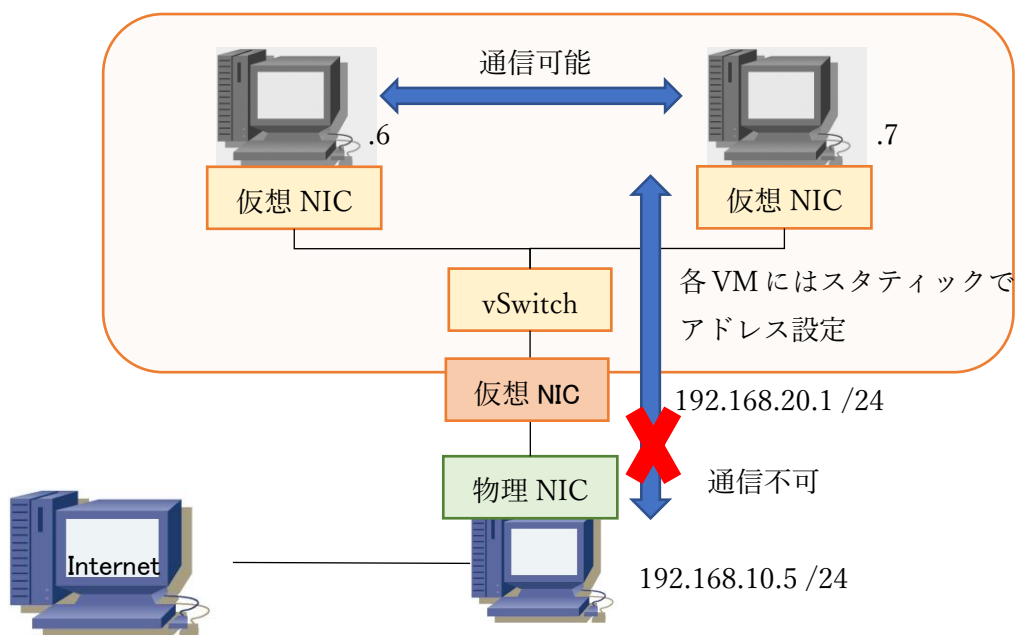
③ 仮想マシンを仮想 NIC に接続して確認（内部ネットワーク）

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)

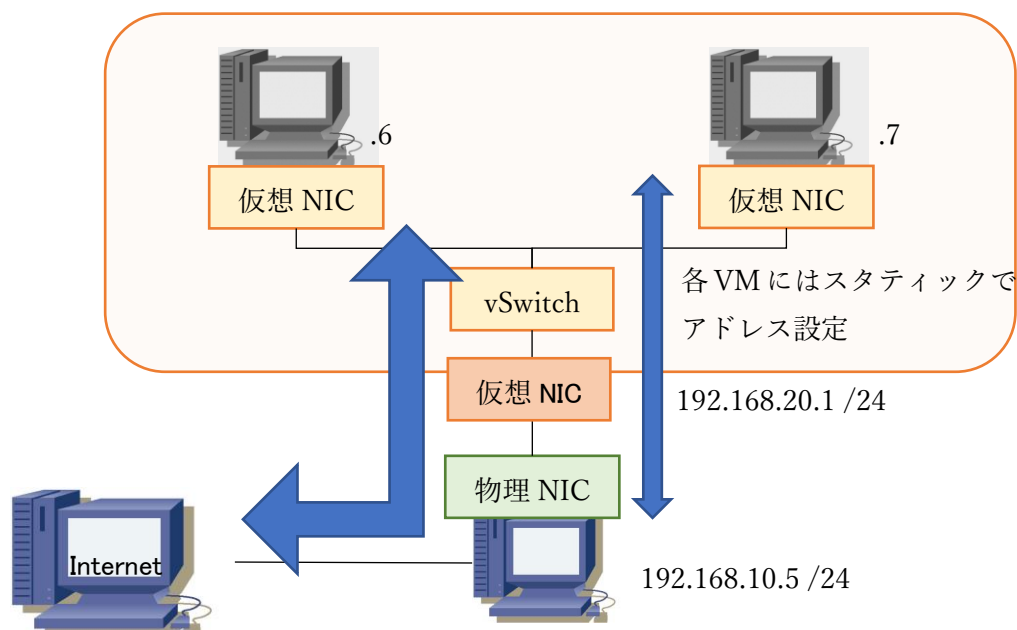
最後に NAT ネットワークです。NAT ネットワークに設定し、それぞれの設定について検証します。各仮想 NIC の IP アドレス設定は次のようになります。

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	0.0.0.0:80->80/tcp	dockerwordpress_wordpress_1
f2fc48620de2	mysql:5.7	"docker-entrypoint.s..."	About an hour ago	Up	0.0.0.0:3306->3306/tcp	dockerwordpress_db_1
4b102f6d049d	php:7.0-apache	"docker-php-entrypoi..."	2 hours ago	Exited (0)		php70-apache
93bcd3151af1	httpd	"httpd-foreground"	5 months ago	Exited (255)	0.0.0.0:80->80/tcp	focused_mcclintock
afd58e477525	httpd	"httpd-foreground"	5 months ago	Exited (0)		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な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
```

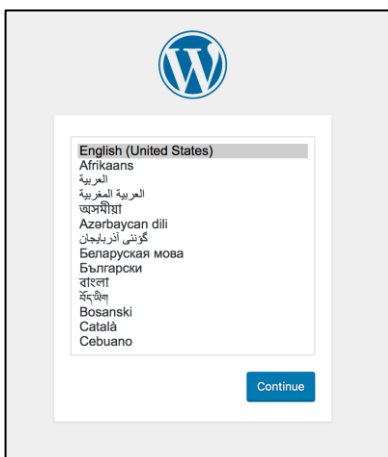
「ファイル例 : docker-compose.yml」に従ってファイルを作成します。適当なテキストエディタで編集してください。

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。

