LinuC レベル 1　Version10.0 学習教材

(新規追加出題範囲)

v1.0.3 版

【改訂履歴】

版	主な改訂内容（概要）
v1.0.0 （2020 年 4 月 1 日）	新規作成
v1.0.1 （2020 年 5 月 29 日）	1.01.2 仮想マシン・コンテナの概念と利用 ・ISO イメージのリンク先変更（最新版の CentOS 7.8 に変更） ・キャプチャ画面の修正 ・一部文言の修正
v1.0.2 （2021 年 4 月 1 日）	・タイトルの変更 ・本教材の位置付け説明のテキスト変更 ・問い合わせ先の変更
v1.0.3 （2023 年 12 月 22 日）	1.01.2 仮想マシン・コンテナの概念と利用 ・CentOS 7 の ISO イメージのリンク先を変更

本学習教材の位置づけ

本学習教材は、ボランティアの方のご協力により LinuC レベル 1 Version 10.0 の出題範囲の中で過去の LinuC のバージョン（レベル 1 ：Version4.0）と比べて新たに追加となった項目に絞って解説をするものです。
市販の LPI-Japan 認定教材（Version10.0 対応）と合わせてご利用いただくことを想定しており、学習の指針を示すものではありませんので、その点ご理解いただいた上でご活用いただければ幸いです。
よりよい内容にしていくため、皆様のコメント・フィードバックなど頂ければ幸いです。

今後必要とされる技術

IoT に代表されるような膨大なデータから価値を生むような現代において、クラウドを活用したシステム連携や主要技術のオープンソース化が進展しています。このような環境の中で活躍できるエンジニアになるためには、以下のような知識・技術を持つことが求められます。

1）クラウド活用技術
従来のオンプレミス環境のシステムにおいてもクラウド活用が進み、サーバー技術だけでなくクラウドの基本技術を身に付けている必要があります。

2）オープンソースへの理解
ソフトウェアを自社で新規に開発するというよりは、外部とコラボレーションしながらオープンソースで提供されている世の中の最新技術をいかに使いこなすかが重要になってきているため、オープンソースを利用する上での一般知識（リテラシー）を保有しておく必要があります。

3）システムアーキテクチャへの知見
システム環境の多様化により、コンピューター・アーキテクチャだけでなく、クラウドも含んだシステム全体のアーキテクチャへの知見が求められています。

一方で、クラウド技術が進展し使いやすくなってきたことに伴い、技術の本質を理解せず「操作しかできないエンジニア」が生まれて技術が空洞化するリスクもあり、問題が起きた際にも対処できるような優れたエンジニアになるためには、技術の本質を理解しておくことが重要になってきます。

LinuC レベル 1／レベル 2 Version 10.0 の価値

Version10.0 の出題範囲は、SIer、エンドユーザ企業、組込みシステムを扱う企業などに所属するトップエンジニア、Linux 技術の教育関係者と書籍の執筆者など約 50 名の SME（Subject Matter Expert）に参加いただいた議論を通じて決められ、現在の開発の現場で本当に必要とされる技術スキルや知識を問う内容になりました。
また、クラウド／仮想化を支える技術的な基盤であり、オープンソースであるが故にコンピューターの仕組みを学ぶ上での最適な教材でもある「Linux」を学ぶことでコンピューター・アーキテクチャを効率的に学べ、クラウド時代に必要な本質的な技術力を保有したエンジニアであることの証明となります。

LinuC レベル 1／レベル 2 Version 10.0 の出題範囲

従来の LinuC のバージョン（レベル 1 ：Version4.0、レベル 2 ：Version4.5）に対して、主に以下の項目が追加されています。

1） クラウドを支える技術領域へ拡大

レベル 1　1.01.2：仮想マシン・コンテナの概念と利用
　　1.10.4：クラウドセキュリティの基礎
レベル 2　2.04.6：システム構成ツール
　　2.05：　仮想化サーバー
　　2.06：　コンテナ

2） オープンソースのリテラシーの理解を追加

レベル 1　1.11：オープンソースの文化

3） システムアーキテクチャの要素を導入

レベル 2　2.13：システムアーキテクチャ

LinuC レベル 1 Version 10.0 の受験対象者

今までの LinuC は物理的なサーバーサイドの IT 技術を対象としていましたが、今回の Version10.0 では、それに加えてクラウド時代に現場で必要な技術要素を取り入れ、システム全体のアーキテクチャに対しても素養のあるエンジニアを育成・認定するものとなっています。Linux システムの構築・運用・保守に関わる IT エンジニアに留まらず、クラウドシステムや各種アプリケーション開発に携わる IT エンジニアにとっても有効な認定になっています。

本教材の範囲

本教材は、LinuC レベル 1　Version 10.0 で新規に追加した以下の出題範囲に関する知識を習得するための教材の位置づけです。

1.01.2：仮想マシン・コンテナの概念と利用
1.10.4：クラウドセキュリティの基礎
1.11：　オープンソースの文化

執筆者・校正者紹介

太田 俊哉（執筆、校正担当）
本資料の作成、校正を担当しました。Linux やオープンソースの技術者をめざす方にお役に立てれば幸いです。

竹本 季史(執筆担当)
仮想化とコンテナを担当しました。クラウド、オンプレミスを問わず、インフラ運用やアプリ開発の迅速化、効率化に幅広く使われている技術です。興味を持っていただく機会となれば幸いです。

著作権

使用に関する権利

●表示

本教材は、特定非営利活動法人エルピーアイジャパンに著作権が帰属するものであることを表示してください。

●非営利

本教材は、非営利目的で教材として自由に利用することができます。商業上の利得や金銭的報酬を主な目的とした営利目的での利用は、特定非営利活動法人エルピーアイジャパンによる許諾が必要です。ただし、本教材を利用した教育において、本教材自体の対価を請求しない場合は、営利目的の教育であっても基本的に利用できます。その場合も含め、LPI-Japan 事務局までお気軽にお問い合わせください。

※営利目的の利用とは以下のとおり規定しております。営利企業または非営利団体において、商業上の利得や金銭的報酬を目的に当教材の印刷実費以上の対価を受講者に請求して当教材の複製を用いた研修や講義を行うこと。

●改変禁止

本教材は、改変せず使用してください。ただし、引用等、著作権法上で認められている利用を妨げるものではありません。本教材に対する改変は、特定非営利活動法人エルピーアイジャパンまたは特定非営利活動法人エルピーアイジャパンが認める団体により行われています。

本教材の使用に関するお問合せ先：
特定非営利活動法人エルピーアイジャパン（LPI-Japan）事務局
〒100-0011 東京都千代田区内幸町 2-1-1 飯野ビルディング 9 階
E-Mail：info@lpi.or.jp

目次

出題範囲 新旧対照表（バージョン 10.0 vs バージョン 4.0）

101 試験出題範囲 新旧対照表

バージョン 10.0			バージョン 4.0		
出題範囲		v.4.0 の範囲	出題範囲		v.10.0 の範囲
1.01：Linux のインストールと仮想マシン・コンテナの利用			101：システムアーキテクチャ		
1.01.1	Linux のインストール、起動、接続、切断と停止	101.2, 101.3, 110.3	101.2	システムのブート	1.01.1 1.01.3
1.01.2	仮想マシン・コンテナの概念と利用	【新設】			
1.01.3	ブートプロセスと systemd	101.2, 101.3	101.3	ランレベル/ブートターゲットの変更とシステムのシャットダウンまたはリブート	1.01.1 1.01.3
1.01.4	プロセスの生成、監視、終了	103.5	103.5	プロセスの生成、監視、終了	1.01.4
			106：ユーザインターフェイスとデスクトップ		
			106.1	X11 のインストールと設定	1.01.5
1.01.5	デスクトップ環境の利用	106	106.2	ディスプレイマネージャの設定	1.01.5
			106.3	アクセシビリティ	【削除】
1.02：ファイル・ディレクトリの操作と管理			104：デバイス、Linux ファイルシステム、ファイルシステム階層標準（一部）		
			104.2	ファイルシステムの整合性の保守	2.02.2
			104.4	ディスククォータの管理	【削除】
1.02.1	ファイルの所有者とパーミッション	104.5	104.5	ファイルのパーミッションと所有者の管理	1.02.1
1.02.2	基本的なファイル管理の実行	103.3	103.3	基本的なファイル管理の実行	1.02.2
1.02.3	ハードリンクとシンボリックリンク	104.6	104.6	ハードリンクとシンボリックリンクの作成・変更	1.02.3
1.02.4	ファイルの配置と検索	104.7	104.7	システムファイルの確認と適切な位置へのファイルの配置	1.02.4

1.03：GNU と Unix のコマンド			103：GNU と Unix のコマンド		
1.03.1	コマンドラインの操作	103.1	103.1	コマンドラインの操作	1.03.1
1.03.2	フィルタを使ったテキストストリームの処理	103.2	103.2	フィルタを使ったテキストストリームの処理	1.03.2
			103.3	基本的なファイル管理の実行)	1.02.2
1.03.3	ストリーム、パイプ、リダイレクトの使用	1.03.4	103.4	ストリーム、パイプ、リダイレクトの使用	1.03.3
			103.5	プロセスの生成、監視、終了	1.01.4
			103.6	プロセスの実行優先度の変更	【削除】
1.03.4	正規表現を使用したテキストファイルの検索	103.7	103.7	正規表現を使用したテキストファイルの検索	1.03.4
1.03.5	エディタを使った基本的なファイル編集の実行	103.8	103.8	vi を使った基本的なファイル編集の実行	1.03.5
1.04：リポジトリとパッケージ管理			102：Linux のインストールとパッケージ管理		
			102.2	ブートマネージャのインストール	2.01.1
			102.3	共有ライブラリの管理	【削除】
1.04.1	apt コマンドによるパッケージ管理	102.4	102.4	Debian パッケージ管理の使用	1.04.1 1.04.2
1.04.2	Debian パッケージ管理	102.4			
1.04.3	yum コマンドによるパッケージ管理	102.5	102.5	RPM および YUM パッケージ管理の使用	1.04.3 1.04.4
1.04.4	RPM パッケージ管理	102.5			
1.05：ハードウェア、ディスク、パーティション、ファイルシステム			101, 102, 104 （混合）		
1.05.1	ハードウェアの基礎知識と設定	101.1	101.1	ハードウェア設定の決定と構成	1.05.1
			102.1	ハードディスクのレイアウト設計	1.05.2
1.05.2	ハードディスクのレイアウトとパーティション	102.1, 104.1	104.1	パーティションとファイルシステムの作成	1.05.2, 1.05.3
1.05.3	ファイルシステムの作成と管理、マウント	104.1, 104.3	104.3	ファイルシステムのマウントとアンマウントの制御	1.05.3

102 試験出題範囲 新旧対照表

バージョン 10.0			バージョン 4.0		
出題範囲		v.4.0 の範囲	出題範囲		v.10.0 の範囲
1.06：シェルとスクリプト			105：シェル、スクリプト、およびデータ管理		
1.06.1	シェル環境のカスタマイズ	105.1	105.1	シェル環境のカスタマイズと使用	1.06.1
1.06.2	シェルスクリプト	105.2	105.2	簡単なスクリプトのカスタマイズまたは作成	1.06.2
			105.3	SQL データ管理	【削除】
1.07：ネットワークの基礎			109：ネットワークの基礎		
1.07.1	インターネットプロトコルの基礎	109.1	109.1	インターネットプロトコルの基礎	1.07.1
1.07.2	基本的なネットワーク構成	109.2	109.2	基本的なネットワーク構成	1.07.2
1.07.3	基本的なネットワークの問題解決	109.3	109.3	基本的なネットワークの問題解決	1.07.3
1.07.4	クライアント側の DNS 設定	109.4	109.4	クライアント側の DNS 設定	1.07.4
1.08：システム管理			107：管理業務		
1.08.1	アカウントの管理	107.1	107.1	ユーザアカウント、グループアカウント、および関連するシステムファイルの管理	1.08.1
1.08.2	ジョブスケジューリング	107.2	107.2	ジョブスケジューリングによるシステム管理業務の自動化	1.08.2
1.08.3	ローカライゼーションと国際化	107.3	107.3	ローカライゼーションと国際化	1.08.3
1.09：重要なシステムサービス			108：重要なシステムサービス		
1.09.1	システム時刻の管理	108.1	108.1	システム時刻の保守	1.09.1
1.09.2	システムのログ	108.2	108.2	システムのログ	1.09.2
1.09.3	メール転送エージェント (MTA)の基本	108.3	108.3	メール転送エージェント (MTA)の基本	1.09.3
			108.4	プリンタと印刷の管理	【削除】
1.10：セキュリティ			110：セキュリティ		
1.10.1	セキュリティ管理業務の実施	110.1	110.1	セキュリティ管理業務の実施	1.10.1
1.10.2	ホストのセキュリティ設定	110.2	110.2	ホストのセキュリティ設定	1.10.2
1.10.3	暗号化によるデータの保護	110.3	110.3	暗号化によるデータの保護	1.01.1, 1.10.3
1.10.4	クラウドセキュリティの基礎	【新設】			
1.11：オープンソースの文化					
1.11.1	オープンソースの概念とライセンス	【新設】			
1.11.2	オープンソースのコミュニティとエコシステム	【新設】			

1.01.2 仮想マシン・コンテナの概念と利用

仮想化とは、コンピューターをはじめとした種々の物理的なハードウェア環境を抽象化し、ソフトウェア化することです。現在は、コンピューティング、ストレージ、クライアント環境、ネットワークなどが仮想化出来るようになっています。
コンピューティング環境では、仮想化を行うため、「仮想マシン」と、「コンテナ」という技術が使われています。ここでは、それぞれの技術の基本と、違いについて学びます。

1. コンピューティング環境の仮想化の利点、欠点

一台のサーバーで一つの業務処理を行えば、複数の業務同士が干渉することはありません。しかし、業務の負荷次第では、サーバーのリソースを無駄にしてしまうこともあります。そこで、一台の物理的なサーバー上で複数の論理的な、「仮想の」サーバーを用意し、おのおので業務処理を動かすようにすれば合理的です。そのための技術がコンピューティングにおける仮想化技術です。

コンピューティング環境に仮想化技術を導入すると、以下のようなメリットが得られます。
(1) 物理的なサーバー台数を削減できる。
(2) コストと運用管理の負担を軽減できる。
(3) サーバーの設置面積を節約できる。
(4) サーバーを迅速に用意できる。
(5) サーバーを柔軟に増減できる。
(6) サーバーの利用効率を高められる。

ただし、以下のようなデメリットもあります。
(1) 処理速度が低下する場合がある。
(2) 障害発生時の影響範囲が大きくなる場合がある。
(3) 導入・運用管理の専門知識が必要になる。

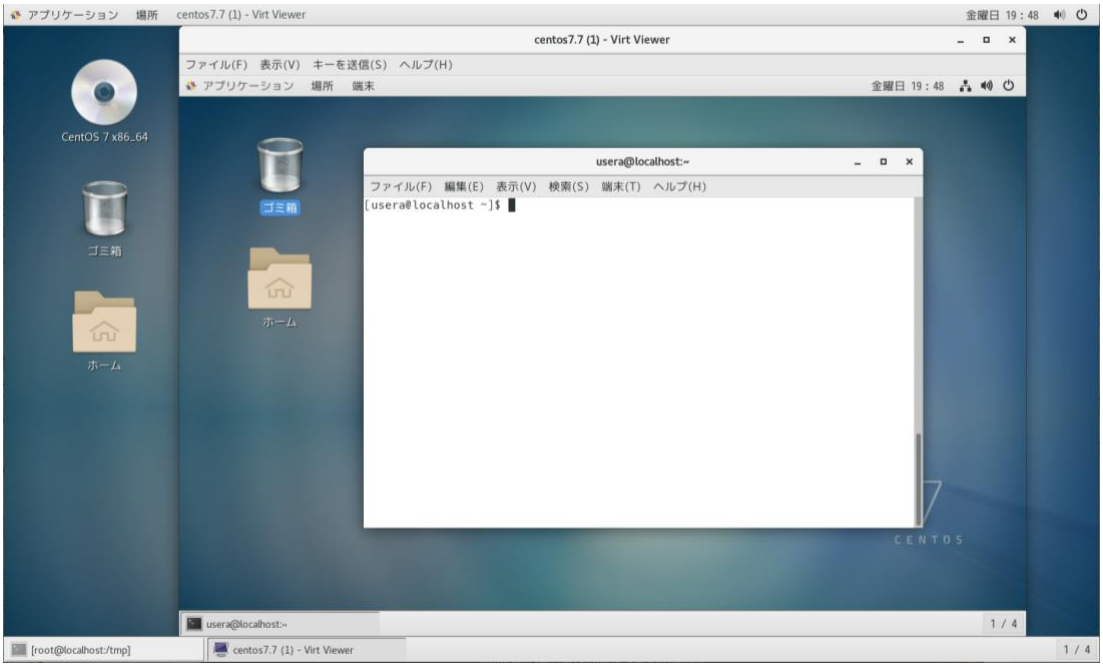
2. 仮想マシンとコンテナ

2.1. 仮想マシンとは

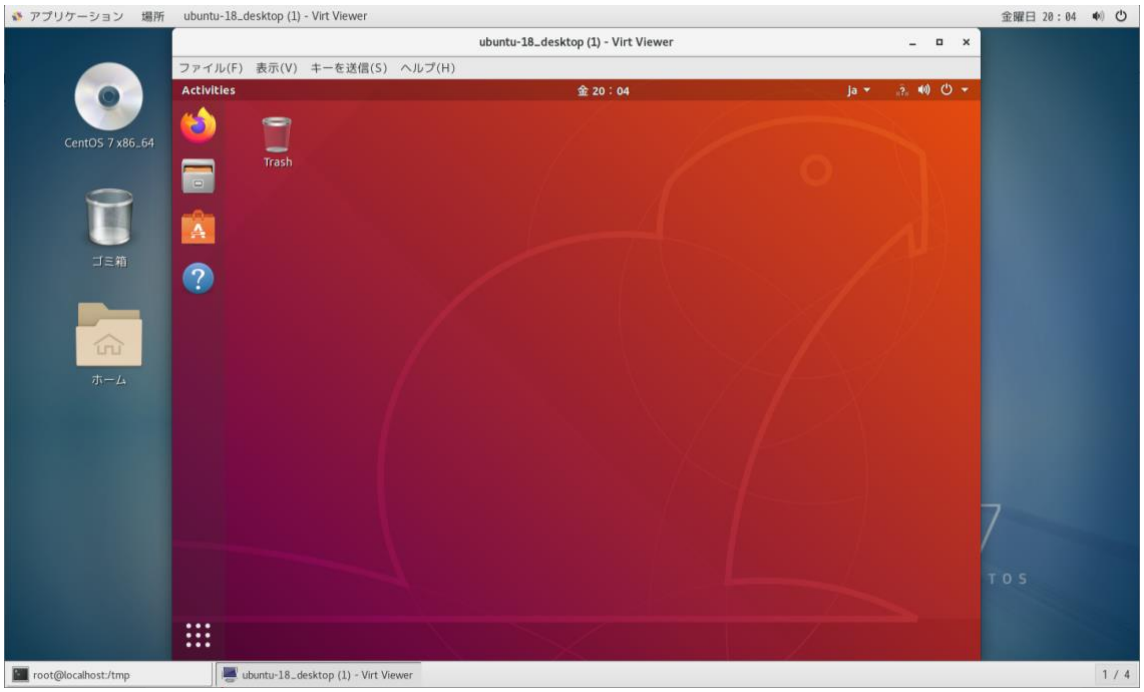
コンピューティングにおける仮想化技術の一つに「仮想マシン」があります。仮想マシンは、物理的なサーバーと同等の機能・動作を再現（エミュレート）した仮想的なハードウェア環境のことです。この機能を実現するのが、「ハイパーバイザー」と呼ばれるソフトウェアです。
ハイパーバイザーは、ハードウェア上で直接動作する「Type1（ネイティブ型）」と、ハードウェアの OS（ホスト OS）上で稼働するアプリケーションが仮想マシンを再現する「Type2（ホスト型）」の二種類に分けられます。代表的なものとしては以下があります。

Type1 の例：KVM、VMware ESX/ESXi、Microsoft Hyper-V、Citrix Xen Server など
Type2 の例：VMware Workstation Player、Oracle VM VirtualBox、Microsoft Virtual Server など

Linux では Kernel-based Virtual Machine（KVM）が OS の一部として組み込まれています。これは、Linux カーネルにハイパーバイザーを埋め込んだもので、仮想化機能としては Type1 に属します。



仮想マシンの実行イメージ（CentOS7 上で CentOS7 を実行）



仮想マシンの実行イメージ（CentOS7 上で Ubuntu18 を実行）

仮想マシンには以下のような特徴があります。

- それぞれの仮想マシンには個別に OS（ゲスト OS）、ライブラリ（実行環境）、ミドルウェア、アプリケーションをインストールして、独立したコンピューティング環境が構築できます。
- ハードウェアをエミュレートするため、CPU やメモリのリソースを多く消費します。
- 仮想マシンは、ホスト OS とは独立しているため、実行環境が変わっても、そのまま動作させることができます。
- ホスト OS とは異なる種類の OS を仮想マシン上で実行することが可能です。

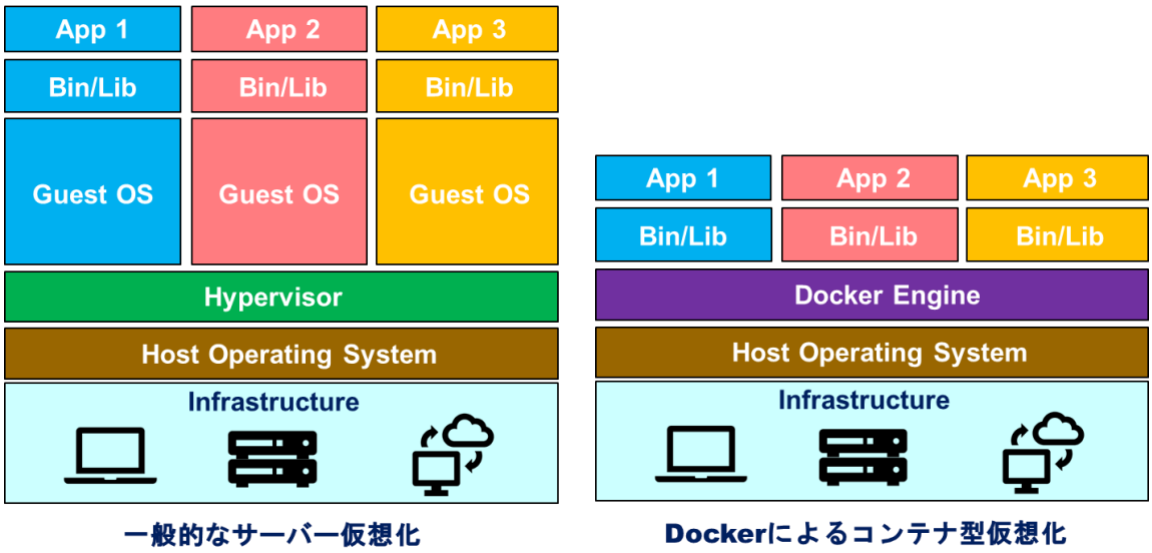
2.2. コンテナとは

もう一つの仮想化技術として「コンテナ」があります。これは、物理的なコンピューター上で稼働する OS（ホスト OS）のリソースの一部を隔離し、仮想的に作り出された実行環境のことです。昔からある chroot の機能を拡張したものと考えるとわかりやすいでしょう。代表的なコンテナ技術に Docker があります。

Docker はもともと、Linux 上で独立した別の Linux システムを起動できるコンテナ管理ソフトウェアとして開発されました。ホスト OS のカーネルを共有しながら、完全に隔離されたアプリケーション実行環境を構築することができます。

コンテナには以下のような特徴があります。

- ホスト OS とは隔離されたプロセスを作成し、その上にライブラリ（実行環境）、ミドルウェア、アプリケーションをインストールして、独立したコンピューティング環境を構築します。
- ホスト OS のプロセスとして動作するので、CPU やメモリのリソース消費は少なく、起動するまでの時間がかかりません。
- コンテナ管理ソフトウェアがハードウェアや OS ごとの違いを吸収するため、他の実行環境へコンテナを容易に移動・配布できます。
- コンテナ内部で実行される OS 環境はホスト OS と同じものに限定されます。



仮想マシンとコンテナの違い

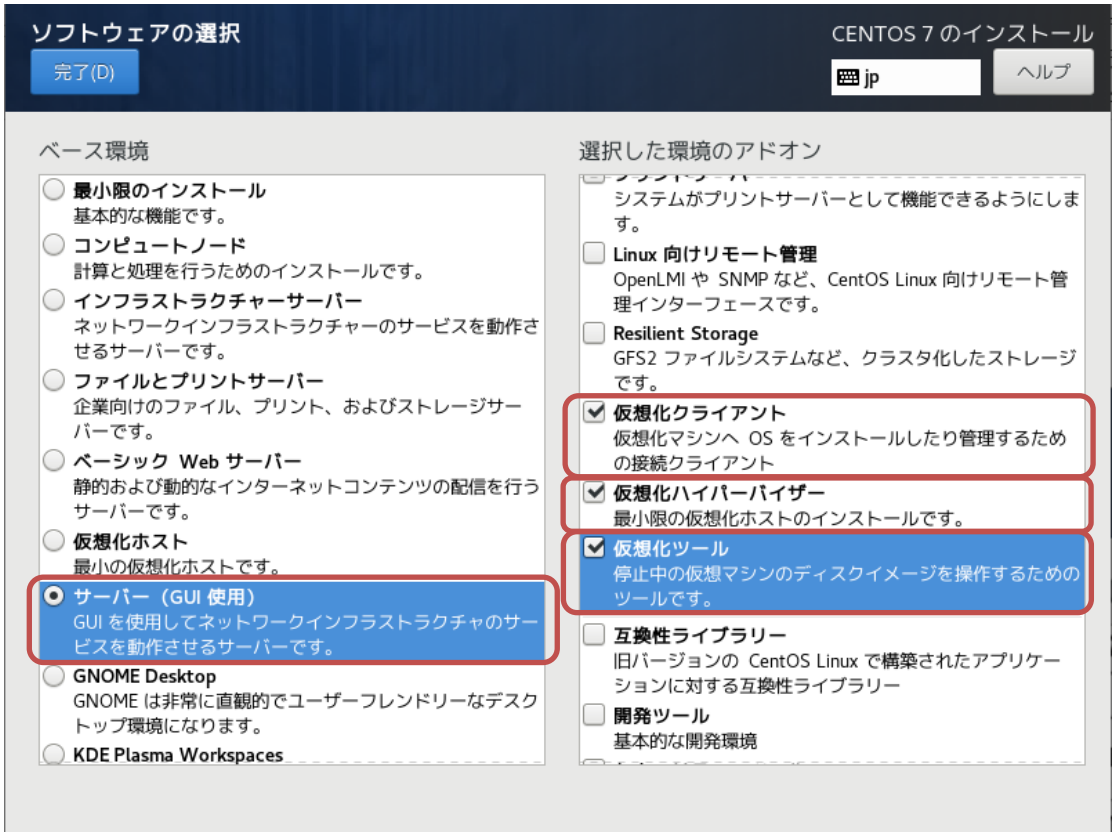
3. 仮想マシンの作成

仮想マシンを実現する技術はいくつかありますが、ここでは、Linux に組み込まれている KVM を使ってみることにします。KVM は、利用する CPU に仮想化支援機能が組み込まれている場合に利用可能です。

※以下の KVM の実行環境については、巻末の「KVM の実行環境に関する注意事項」を参照ください。

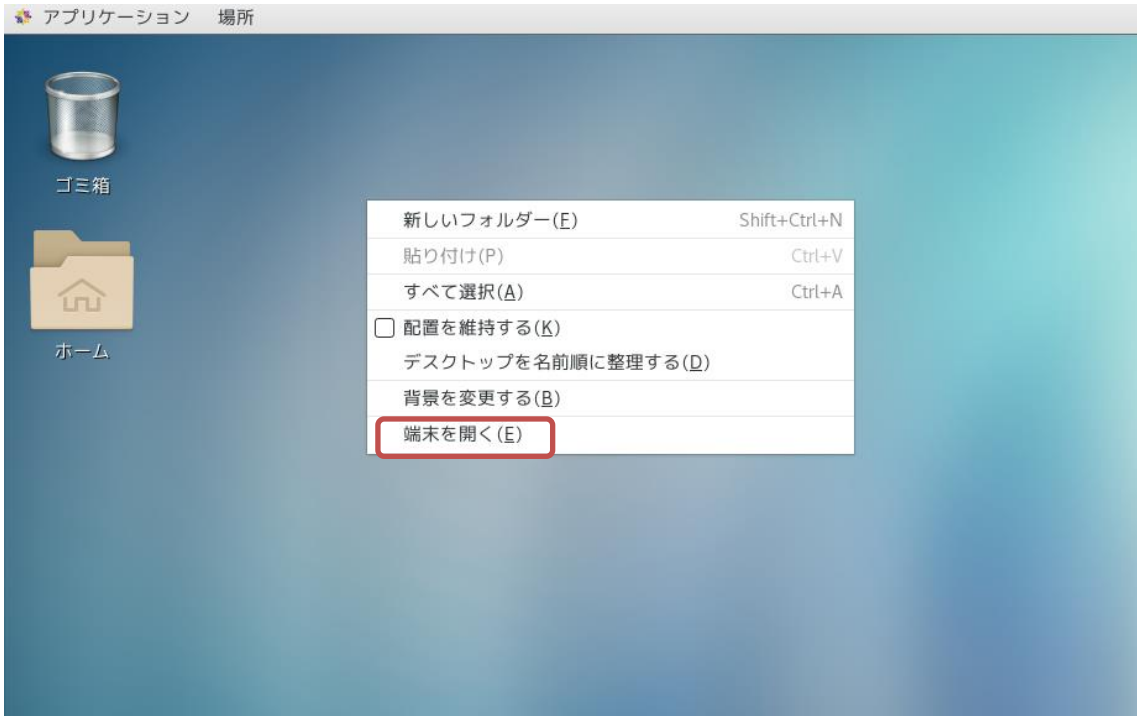
3.1 KVM のセットアップ

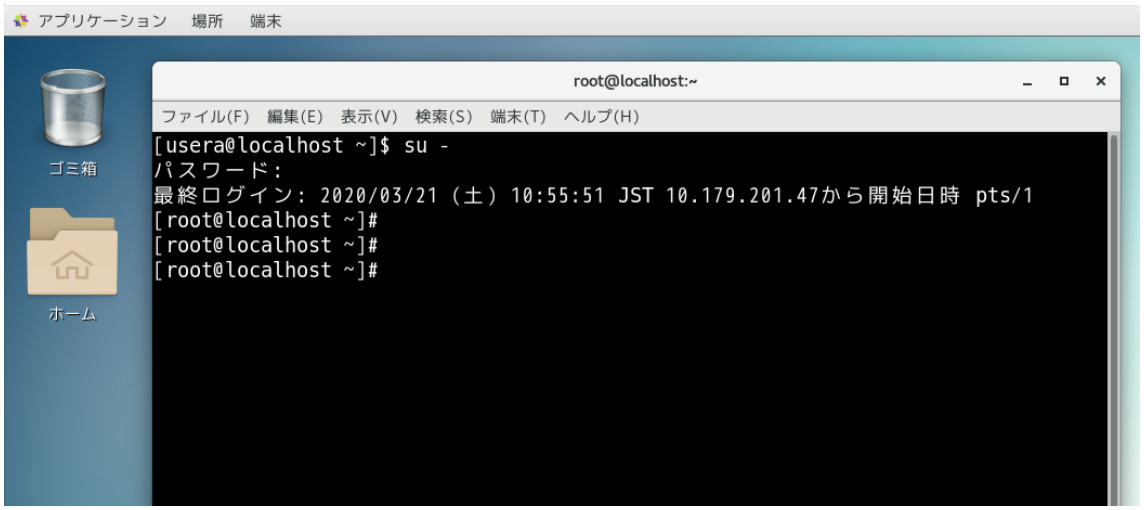
KVM は Linux に組み込まれていますので、KVM の制御に必要なパッケージをインストールすれば、多くのディストリビューションですぐに利用できます。
たとえば、CentOS ではインストール時にベース環境を「サーバー(GUI 使用)」、選択した環境のアドオンに「仮想化クライアント」、「仮想化ハイパーバイザー」、「仮想化ツール」を選択すれば、KVM が GUI の操作環境で利用できます。



3.2 KVM 環境の設定と確認

以下の操作は CentOS 7 を GUI モードで起動後、デスクトップ上で右クリックから「端末を開く」でコンソール端末を開いて、root ユーザーで操作することを想定しています。





まず、CPU が仮想化支援機能に対応していることを確認します。これには/proc/cpuinfo で CPU 情報を確認します。CPU が Intel VT-x/d 対応の場合は vmx、AMD-V 対応の場合は svm が表示されます。

egrep '(vmx|svm)' /proc/cpuinfo

```
[root@localhost ~]# egrep '(vmx|svm)' /proc/cpuinfo
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pg
e mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb
rdtscp lm constant_tsc arch_perfmon nopl xtopology tsc_reliable nonst
op_tsc eagerfpu pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 x
2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hyper
visor lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced tp
```

また、lsmod でカーネルに kvm のモジュールが読み込まれていることを確認します。

lsmod | grep kvm

```
[root@localhost ~]# lsmod | grep kvm
kvm_intel             188644  0
kvm                   621480  1 kvm_intel
irqbypass              13503  1 kvm
```

rpm コマンドで KVM の実行に必要なパッケージの確認を行います。
必要なパッケージは、qemu-kvm、qemu-img、libvirt、virt-install、virt-manager、virt-viewer です。

rpm -q qemu-kvm qemu-img libvirt virt-install virt-manager virt-viewer

```
[root@localhost ~]# rpm -q qemu-kvm qemu-img libvirt virt-install
virt-manager virt-viewer
qemu-kvm-1.5.3-167.el7.x86_64
qemu-img-1.5.3-167.el7.x86_64
libvirt-4.5.0-23.el7.x86_64
virt-install-1.5.0-7.el7.noarch
virt-manager-1.5.0-7.el7.noarch
virt-viewer-5.0-15.el7.x86_64
```

パッケージの確認後、仮想化デーモンの libvirtd の起動状態を確認します。

systemctl status libvirtd

```
[root@localhost ~]# systemctl status libvirtd
● libvirtd.service - Virtualization daemon
   Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled; vendor
 preset: enabled)
   Active: active (running) since 水 2020-05-27 11:24:56 JST; 57min ago
     Docs: man:libvirtd(8)
           https://libvirt.org
 Main PID: 1386 (libvirtd)
    Tasks: 19 (limit: 32768)
   CGroup: /system.slice/libvirtd.service
           ├─1386 /usr/sbin/libvirtd
           ├─1753 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/...
           └─1755 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/dnsmasq/...

 5月 27 11:24:57 localhost.localdomain dnsmasq[1753]: started, version 2...
 5月 27 11:24:57 localhost.localdomain dnsmasq[1753]: compile time optio...
 5月 27 11:24:57 localhost.localdomain dnsmasq-dhcp[1753]: DHCP, IP rang...
 5月 27 11:24:57 localhost.localdomain dnsmasq-dhcp[1753]: DHCP, sockets...
```

起動していれば、Active:active(running)と表示されます。
もし起動していない場合は、libvirtd を起動します。

systemctl start libvirtd

```
[root@localhost ~]# systemctl start libvirtd
```

次回の Linux 起動時に libvirtd が自動的に起動する設定が有効であることを確認します。
「enabled」と表示が出れば libvirtd の自動起動が有効になっています。

systemctl is-enabled libvirtd

```
[root@localhost ~]# systemctl is-enabled libvirtd
enabled
```

「disabled」と表示された場合は下記コマンドで有効にします。

systemctl enable libvirtd

```
[root@localhost ~]# systemctl enable libvirtd
Created symlink from /etc/systemd/system/multi-user.target.wants/libvirtd.se
rvice to /usr/lib/systemd/system/libvirtd.service.
Created symlink from /etc/systemd/system/sockets.target.wants/virtlockd.sock
et to /usr/lib/systemd/system/virtlockd.socket.
Created symlink from /etc/systemd/system/sockets.target.wants/virtlogd.socke
t to /usr/lib/systemd/system/virtlogd.socket.
```

3.3 ゲスト OS の ISO イメージをダウンロード

/tmp に移動して CentOS7 の Minimal の ISO イメージをダウンロードします。

```
cd /tmp
curl -LO http://ftp.riken.jp/Linux/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-2009.iso
```

```
[root@localhost tmp]# curl -LO http://ftp.riken.jp/Linux/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-2009.iso
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1035M  100 1035M    0     0  48.2M      0  0:00:21  0:00:21 --:--:-- 51.3M
```

ダウンロードできたら確認します。

```
ls -l CentOS-7-x86_64-Minimal-2009.iso
```

```
[root@localhost tmp]# ls -l CentOS-7-x86_64-Minimal-2009.iso
-rw-r--r--. 1 root root 1085276160  5月  27 12:28 CentOS-7-x86_64-Minimal-2009.iso
```

3.4 仮想マシンのイメージファイルの作成

仮想マシンも物理マシンと同様、OS を格納するストレージ領域が必要です。そこで、仮想マシンで使用するストレージのイメージファイルを作成します。容量を 10GB とします。

```
qemu-img create -f qcow2 /var/lib/libvirt/images/centos7.img 10G
```

```
[root@localhost tmp]# qemu-img create -f qcow2 /var/lib/libvirt/images/centos7.img 10G
Formatting '/var/lib/libvirt/images/centos7.img', fmt=qcow2 size=10737418240 encryption=off cluster_size=65536 lazy_refcounts=off
```

作成できたことを確認します。

```
ls -l /var/lib/libvirt/images/centos7.img
```

```
[root@localhost tmp]# ls -l /var/lib/libvirt/images/centos7.img
-rw-r--r--. 1 root root 197120  5月  27 12:31 /var/lib/libvirt/images/centos7.img
```

3.5 仮想マシンの作成とゲスト OS のインストール

仮想マシンを作成し、ISO イメージから CentOS7 のインストールを行います。
ゲスト OS の仮想マシン設定は下記とします。

・仮想マシン名　centos7
・CPU 数　1
・メモリ　1024MB
・--disk path で先ほど作成したストレージのイメージファイルを指定します。
・--location でダウンロードした CentOS7 の ISO イメージを指定します。

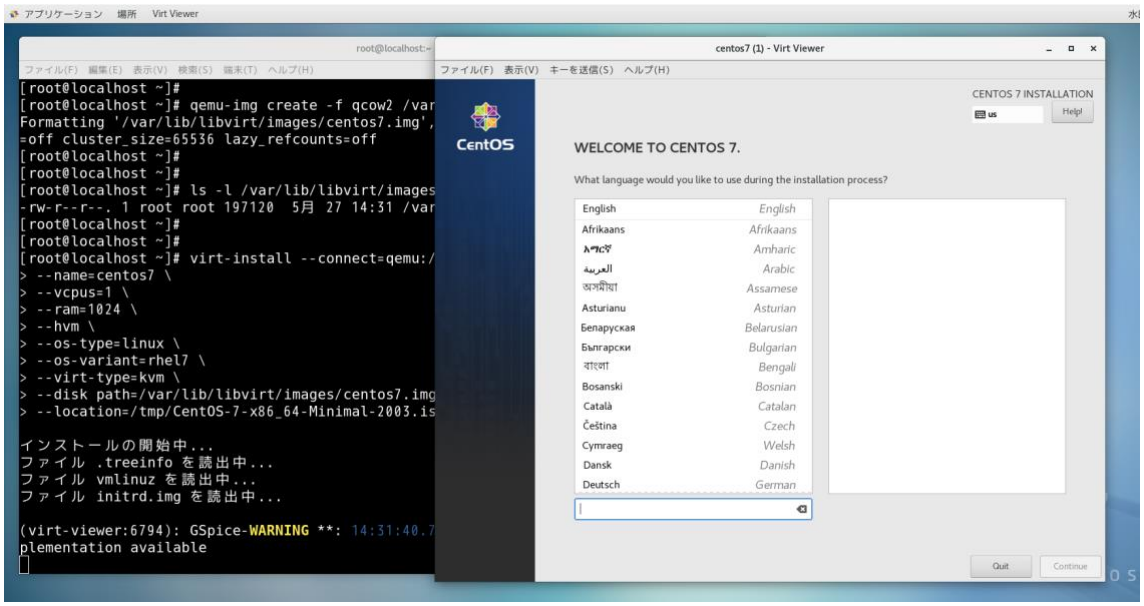
```
virt-install --connect=qemu:///system ¥
--name=centos7 ¥
--vcpus=1 ¥
--ram=1024 ¥
--hvm ¥
--os-type=linux ¥
--os-variant=rhel7 ¥
--virt-type=kvm ¥
--disk path=/var/lib/libvirt/images/centos7.img ¥
--location=/tmp/CentOS-7-x86_64-Minimal-2009.iso
```

（virt-install コマンドを 1 行で記述すると長くなるので、見やすいように適宜バックスラッシュで改行しています。）

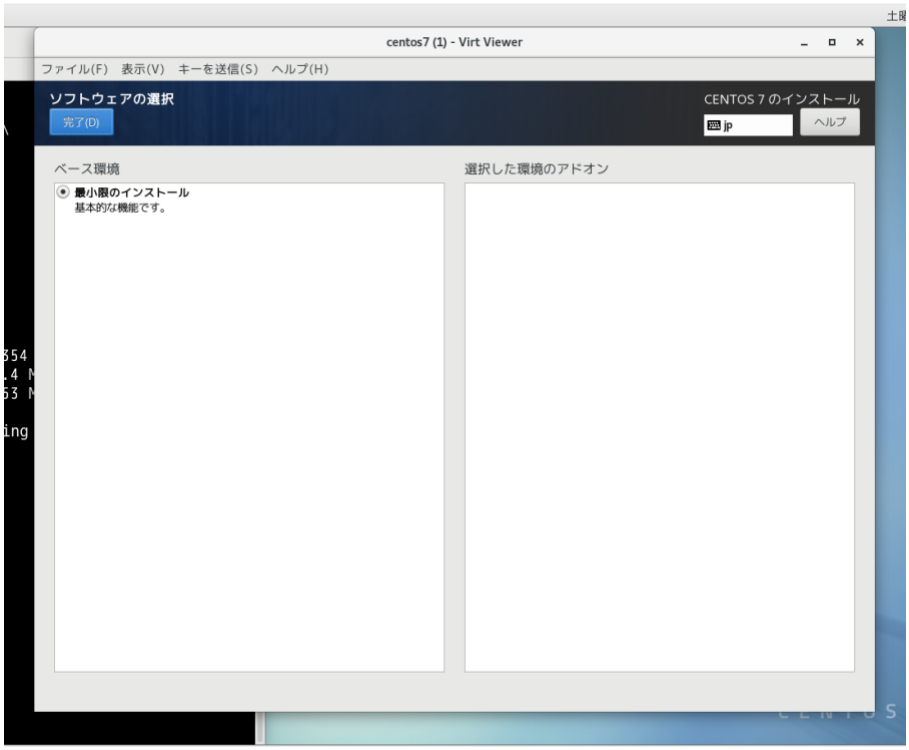
```
[root@localhost ~]# virt-install --connect=qemu:///system \
> --name=centos7 \
> --vcpus=1 \
> --ram=1024 \
> --hvm \
> --os-type=linux \
> --os-variant=rhel7 \
> --virt-type=kvm \
> --disk path=/var/lib/libvirt/images/centos7.img \
> --location=/tmp/CentOS-7-x86_64-Minimal-2009.iso

インストールの開始中...
ファイル .treeinfo を読出中...                                    |  353 B  00:00:00
ファイル vmlinuz を読出中...                                      | 6.4 MB  00:00:00
ファイル initrd.img を読出中...                                   |  53 MB  00:00:00
```

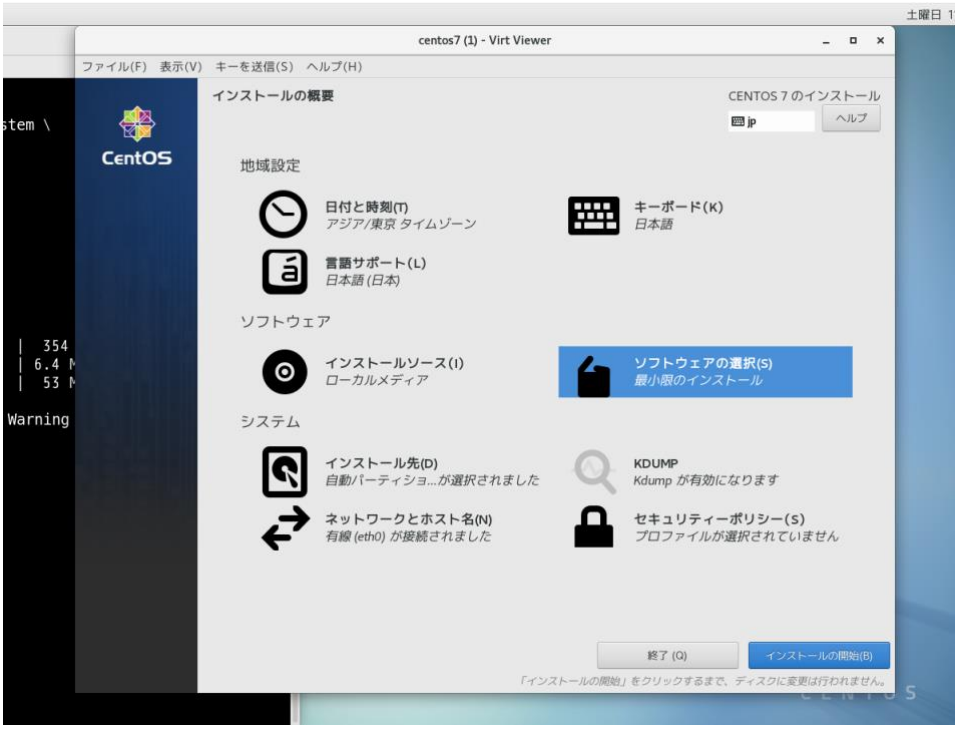
別のウィンドウが開き CentOS7 のインストールが開始されます。



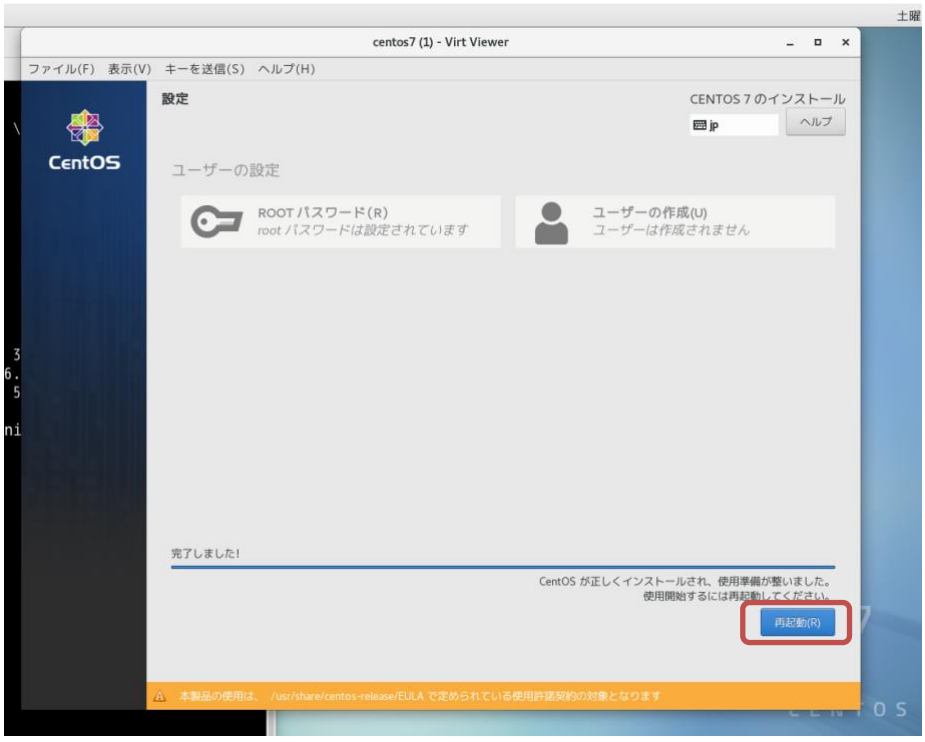
「ソフトウェアの選択」は Minimal の ISO イメージなので「最小限のインストール」のみです。



その他、「インストール先」と「ネットワークとホスト名」を設定して「インストールの開始」をクリックします。



インストールが完了したら「再起動」をクリックします。



再起動後、ゲスト OS にログインして CentOS のバージョンを確認します。

cat /etc/redhat-release

以上でゲスト OS のインストールが完了しました。

4. 仮想マシンの起動・停止のコマンド

KVM では、virsh コマンドを使って仮想マシンを操作します。以下に起動・停止に関する主なコマンドを記します。

※ ドメインとは、CPU、メモリ、仮想ディスク、仮想マシンを実行するネットワークなど一連のリソースを含む仮想マシンの構成を指します。

virsh list --all

定義されているドメインの一覧を表示します。--all オプションは、停止状態および動作中のドメインの一覧を表示します。

```
[root@localhost ~]# virsh list --all
 Id    名前                         状態
----------------------------------------------------
 -     centos7                      シャットオフ
```

virsh start [ドメイン名]

[ドメイン名] の仮想マシンを起動します。

```
[root@localhost ~]# virsh start centos7
ドメイン centos7 が起動されました
```

virsh shutdown [ドメイン名]

[ドメイン名] の仮想マシンをシャットダウンします。

```
[root@localhost ~]# virsh shutdown centos7
ドメイン centos7 はシャットダウン中です
```

virsh reboot [ドメイン名]

[ドメイン名] の仮想マシンを再起動します。

```
[root@localhost ~]# virsh reboot centos7
ドメイン centos7 を再起動しています
```

virsh suspend [ドメイン名]

[ドメイン名] の仮想マシンを一時停止します。

```
[root@localhost ~]# virsh suspend centos7
ドメイン centos7 は一時停止されました
```

virsh list で仮想マシンが一時停止中であることを確認できます。

```
[root@localhost ~]# virsh list
 Id    名前                         状態
----------------------------------------------------
 6     centos7                        一時停止中
```

virsh resume [ドメイン名]

一時停止中の [ドメイン名] の仮想マシンを再開します。

```
[root@localhost ~]# virsh resume centos7
ドメイン centos7 が再開されました
```

virsh destroy [ドメイン名]

[ドメイン名] の仮想マシンを強制停止します。

(物理マシンの電源を切る動作と同様です)

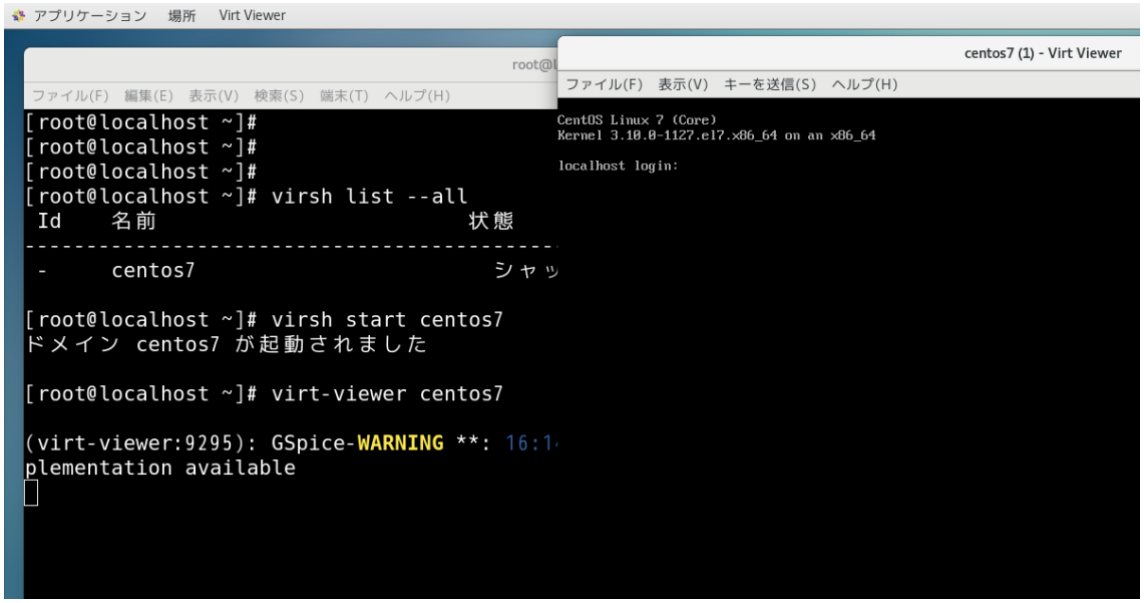
```
[root@localhost ~]# virsh destroy centos7
ドメイン centos7 は強制停止されました
```

virt-viewer [ドメイン名]

[ドメイン名]で指定した起動中の仮想マシンの操作画面を開きます。

```
[root@localhost ~]# virt-viewer centos7
```

実行すると下記のように仮想マシンの操作画面が開きます。



5. コンテナの作成

コンテナ管理ソフトウェアとして一般的な Docker を使ってみることにします。

5.1 Docker のセットアップ

以下の手順で Docker をセットアップします。ホスト OS は CentOS7 を使用します。以下の操作は root ユーザーで操作することを想定しています。

yum コマンドで docker をインストールします。

yum install docker

```
[root@localhost ~]# yum install docker
読み込んだプラグイン:fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: ftp.iij.ad.jp
 * extras: ftp.iij.ad.jp
 * updates: ftp.iij.ad.jp
```

rpm コマンドで docker がインストールされたことを確認します。

rpm -q docker

```
[root@localhost ~]# rpm -q docker
docker-1.13.1-109.gitcccb291.el7.centos.x86_64
```

また、下記のコマンドで docker のバージョンを確認できます。

docker -v

```
[root@localhost ~]# docker -v
Docker version 1.13.1, build cccb291/1.13.1
```

docker を起動します。

systemctl start docker

```
[root@localhost ~]# systemctl start docker
```

docker が起動していることを確認します。

systemctl status docker

```
[root@localhost ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: d
isabled)
   Active: active (running) since 火 2020-03-24 12:02:04 JST; 3min 12s ago
     Docs: http://docs.docker.com
 Main PID: 6812 (dockerd-current)
    Tasks: 17
   CGroup: /system.slice/docker.service
           ├─6812 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/...
           └─6817 /usr/bin/docker-containerd-current -l unix:///var/run/docker/lib...
```

起動していれば、Active:active(running)と表示されます。

次回の Linux 起動時に docker が自動的に起動するように設定します。

systemctl enable docker

```
[root@localhost ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service to
/usr/lib/systemd/system/docker.service.
```

自動起動が有効になったことを確認します。「enabled」と表示が出れば docker の自動起動が有効になっています。

systemctl is-enabled docker

```
[root@localhost ~]# systemctl is-enabled docker
enabled
```

以上で Docker のセットアップが完了しました。

5.2 OS イメージのダウンロードとコンテナの基本操作

コンテナはコンテナ用の OS イメージから作られます。次に Docker 公式のリポジトリである DockerHub から OS イメージをダウンロードします。ここでは CentOS の最新版のイメージをダウンロードします。

docker pull centos:latest

```
[root@localhost ~]# docker pull centos:latest
Trying to pull repository docker.io/library/centos ...
latest: Pulling from docker.io/library/centos
8a29a15cefae: Pull complete
Digest: sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
Status: Downloaded newer image for docker.io/centos:latest
```

取得したコンテナイメージを確認します。

docker images

```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker.io/centos    latest              470671670cac        2 months ago        237 MB
```

REPOSITORY に「docker.io/centos」、TAG に「latest」と表示されていればイメージがダウンロードできています。

次に、イメージからコンテナを作成して bash を起動してログインします。
-it オプションで作成したコンテナへのコマンド入力と実行結果の出力ができるようになります。

docker run -it centos:latest /bin/bash

```
[root@localhost ~]# docker run -it centos:latest /bin/bash
[root@df219e6e8390 /]#
```

実行後、プロンプトのホスト名がコンテナ ID に変化することでコンテナにログインしていることが分かります。

コンテナログイン後、コンテナの OS のバージョンを確認します。
CentOS の最新版のコンテナであることが分かります。

cat /etc/redhat-release

```
[root@df219e6e8390 /]# cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)
```

exit を実行するとコンテナは停止してホスト OS に操作が戻ります。

```
[root@9b3c1c0d4d83 /]# exit
exit
[root@localhost ~]#
```

6. コンテナの起動・停止のコマンド

Docker では、docker コマンドを使ってコンテナを操作します。以下にコンテナの起動・停止に関する主なコマンドを記します。

docker ps

起動中のコンテナ一覧を表示します。

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
  PORTS               NAMES
df219e6e8390        centos:latest       "/bin/bash"         18 minutes ago      Up 17 minutes
                      competent_noether
6f2a28e56ef1        ubuntu              "/bin/bash"         3 minutes ago       Up 3 minutes
                              priceless_dijkstra
242eec2998a5        centos:latest       "/bin/bash"         57 minutes ago      Up 51 minutes
                              frosty_kirch
```

docker ps -a

停止中も含めたコンテナ一覧を表示します。STATUS の Exited(0)はコンテナが正常に停止したことを示します。

```
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
            PORTS               NAMES
df219e6e8390        centos:latest       "/bin/bash"         18 minutes ago      Exited (0) About a
 minute ago                             competent_noether
6f2a28e56ef1        ubuntu              "/bin/bash"         4 minutes ago       Up 4 minutes
                                       priceless_dijkstra
242eec2998a5        centos:latest       "/bin/bash"         57 minutes ago      Up 51 minutes
                                       frosty_kirch
```

docker start [コンテナ ID]

停止中のコンテナを起動します。

```
[root@localhost ~]# docker start df219e6e8390
df219e6e8390
```

[コンテナ ID]は全コンテナで一意となればよいので始めの 2,3 文字の入力でも実行可能です。以後のコマンドの[コンテナ ID]も同様です。

```
[root@localhost ~]# docker start df
df
```

docker ps で起動したことを確認します。STATUS に Up と表示されます。

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
    PORTS               NAMES
df219e6e8390        centos:latest       "/bin/bash"         4 minutes ago       Up 6 seconds
                        competent_noether
```

docker stop [コンテナ ID]

起動中のコンテナを停止します。

```
[root@localhost ~]# docker stop df219e6e8390
df219e6e8390
```

docker restart [コンテナ ID]

起動中のコンテナを再起動します。

```
[root@localhost ~]# docker restart df219e6e8390
df219e6e8390
```

docker pause [コンテナ ID]

起動中のコンテナを一時的に停止します。

docker ps の STATUS は Up(Paused)となります。

```
[root@localhost ~]# docker pause df219e6e8390
df219e6e8390
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
df219e6e8390        centos:latest       "/bin/bash"         47 hours ago        Up 46 seconds (Paused)
```

docker unpause [コンテナ ID]

一時的に停止したコンテナを再開します。

```
[root@localhost ~]# docker unpause df219e6e8390
df219e6e8390
```

docker kill [コンテナ ID]

起動中のコンテナを強制的に停止します。

```
[root@localhost ~]# docker kill df219e6e8390
df219e6e8390
```

強制終了したため **docker ps -a** の STATUS は Exited(137)となります。

```
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
7e6f97165739        centos              "/bin/bash"         8 minutes ago       Exited (0) 8 minutes ago
7fc191e63cd8        centos:latest       "/sbin/sshd"        47 hours ago        Created
df219e6e8390        centos:latest       "/bin/bash"         47 hours ago        Exited (137) 6 seconds ago
```

docker attach [コンテナ ID]

/bin/bash 実行中のコンテナにアタッチ(接続)します。

```
[root@localhost ~]# docker attach df219e6e8390
[root@df219e6e8390 /]#
```

ps コマンドを実行すると PID1 で/bin/bash が実行中です。

```
[root@df219e6e8390 /]#ps ax
   PID TTY      STAT   TIME COMMAND
     1 ?        Ss     0:00 /bin/bash
    23 ?        R+     0:00 ps ax
```

この状態で exit すると、PID1 の bash が exit されてコンテナは停止します。コンテナからデタッチ(切断)してホスト OS に操作を切り替えるには、Ctrl+p、Ctrl+q を連続して押します。
すると、プロンプトがホスト OS に切り替わります。**docker ps** で確認するとコンテナの STATUS は Up であり、コンテナは起動中であることが分かります。

```
[root@df219e6e8390 /]#
[root@df219e6e8390 /]# [root@localhost ~]#   <-- Ctrl+p,Ctrl+q を押すことでホスト OS に戻りました
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
    PORTS               NAMES
df219e6e8390        centos:latest       "/bin/bash"         14 minutes ago      Up 9 minutes
                    competent_noether
```

docker exec -it [コンテナ ID] /bin/bash

起動中のコンテナで新規に bash を起動してログインします。

```
[root@localhost ~]# docker exec -it df219e6e8390 /bin/bash
[root@df219e6e8390 /]#
```

ps コマンドで確認すると PID1 の bash とは別の bash が起動していることが分かります。

```
[root@df219e6e8390 /]#ps ax
   PID TTY      STAT   TIME COMMAND
     1 ?        Ss+    0:00 /bin/bash
    24 ?        Ss     0:00 /bin/bash
    38 ?        R+     0:00 ps ax
```

この状態で exit すると PID24 の bash が exit されるため attach の場合と異なり、コンテナは停止しません。**docker ps** で確認すると STATUS は Up であり、コンテナは起動中であることが分かります。

```
[root@df219e6e8390 /]#exit
exit
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
  PORTS               NAMES
df219e6e8390        centos:latest       "/bin/bash"         18 minutes ago      Up 17 minutes
                      competent_noether
```

1.10.4 クラウドセキュリティの基礎

近年、システムの稼働基盤としてパブリッククラウドが広く利用されるようになりました。事業者が用意したハードウェアインフラをインターネット経由で利用するクラウド環境と、ハードウェアを自社で調達・保有・運用するオンプレミス環境では、セキュリティ対策を講じるべき範囲に差があります。ここでは、パブリッククラウドを利用するときに必要なセキュリティ対策の基礎を学びます。

1. クラウドにおけるセキュリティの責任範囲

オンプレミス環境では、サーバー、ストレージ、ネットワーク機器などのハードウェアを自社のデータセンター（データセンター事業者が提供するコロケーションサービス／ハウジングサービスを含む）に設置し、ハードウェア上で稼働するソフトウェア、データ、さらには利用者すべてを自社で運用・管理します。
セキュリティに関しては、以下のすべてを自社の責任において対策する必要があります。

(1) 入退室管理、設備監視、ラック管理などのデータセンターセキュリティ
（ただし、コロケーションサービス／ハウジングサービスを使う場合は、データセンター事業者が対応を行います。）
(2) BIOS/UEFI 管理、ファームウェア管理、デバイス管理などのハードウェアセキュリティ
(3) ファイアウォール、IPS/IDS、Web フィルタリング、ルーティングなどのネットワークセキュリティ
(4) OS・アプリケーションの脆弱性対応、パッチ適用などのソフトウェアセキュリティ
(5) 認証／ID 管理などのユーザセキュリティ
(6) データの暗号化、アクセス管理、アンチウイルス,バックアップなどのデータセキュリティ

それに対しクラウド環境では、(1)のデータセンターや(2)のハードウェアのセキュリティはクラウド事業者が責任を持って対策を講じます。それ以外の部分（上記の(3)~(6)）については、クラウドサービスの種類によっては、自社の責任範囲が変わってきます。

(1) IaaS（Infrastructure as a Service）
IaaS の場合、ゲスト OS の管理（更新やセキュリティパッチなど)、サーバーインスタンスにインストールしたアプリケーションの管理、各インスタンス向けに提供される

ファイアウォールの構成、データの保護などに自社で責任を負います。

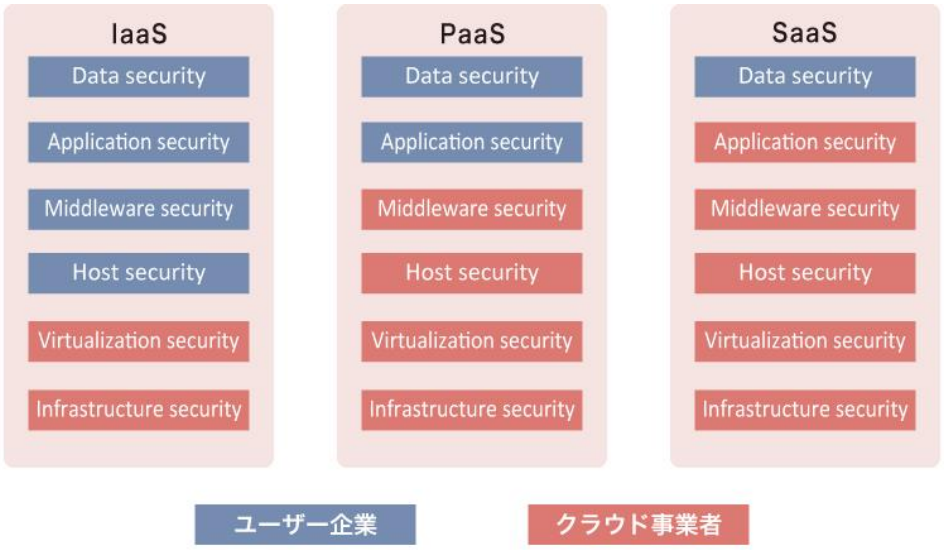
(2) PaaS（Platform as a Service）

PaaS の場合、抽象化されたサービス（データベースなど）を利用するアプリケーションの管理、データの保護、アクセス管理などに自社で責任を負います。

(3) SaaS（Software as a Service）

データの保護のみに責任を自社で負います。

クラウド事業者の責任範囲については、例えば米国の調査会社・ガートナーでは以下のような「セキュリティ責任共有モデル」を公開しています。



ガートナーのセキュリティ責任共有モデル

出典：クラウドにおけるセキュリティサービスの効果的な管理のガイドライン

https://www.cloudsecurityalliance.jp/site/wp-content/uploads/2019/09/Guideline-on-Effectively-Managing-Security-Service-in-the-Cloud-06_02_19_J_FINAL.pdf

2. クラウドセキュリティの留意点

パブリッククラウドを利用する場合、オンプレミス環境とは違った対応が種々必要となります。セキュリティ対策の面からは、以下の点について留意する必要があります。

(1) リージョン

クラウドサービスでは、IaaS 環境やデータを格納するストレージは、クラウド事業者が管理するどこかのデータセンターにあります。そして、データセンターがある地域のことを「リージョン」と言います。「リージョン」は、それぞれが地理的に離れていて、独立して管理されています。大手クラウド事業者では世界各地にリージョンがあり、さらに 1 つの国の中でも複数のリージョンを用意しています。もちろん日本国内にもリージョンはあります。

国内から海外のリージョンも利用できますが、格納されるデータの種類によっては、法令により国外持ち出しが制限されている場合があります。そのようなデータを扱う場合には、利用条件に応じてリージョンを選択する必要があります。また、海外リージョンを利用する場合、その国の捜査機関などに機密データが流出するリスクも考えられます。このようなことを考え、リージョンを選択する必要があります。

(2) クラウドサービス事業者の制約

クラウドでは、計画的または予期しないタイミングにおいて**メンテナンス**が行われる場合があります。また、ハードウェアやネットワークの障害により、予期しないタイミングでインスタンスが停止、もしくは再起動する可能性もあります。こうした自社で調整ができない、事業者の制約を考慮し、サービスの継続性や、データを保護するための手段を講じておく必要があります。

(3) 揮発性ストレージ

クラウドのサーバーインスタンスでは、恒久的にデータが保持可能なストレージのほかに、一時的なデータボリュームとして使用する揮発性ストレージが使用できます。このような揮発性ストレージを使うと、インスタンスが停止すれば保存されていたデータはすべて削除されます。そのため、揮発性ストレージの誤使用は、データの喪失というリスクが発生します。

3. クラウドセキュリティの利用

クラウド上でのセキュリティを設定するには、パブリッククラウド事業者が提供している

セキュリティ設定機能を使います。通常は、クラウドの設定と監視を行うためのインターフェイスを備えた管理者向けツールである管理コンソールを使用します。
また、オンプレミス環境のサーバーの外側でセキュリティを確保するための仕組み、例えばファイアウォール機能を包含した UTM（統合脅威管理）アプライアンスのような機能を使う場合は、クラウドサービス内で同等の機能を設定して利用します。

(1) ファイアウォール

多くのパブリッククラウドでは、IaaS 環境におけるファイアウォールの設定だけではなく、種々の場面でファイアウォールを設定できるようになっています。クラウド環境では、オンプレミス環境とは異なったシステム構成になりますが、オンプレミス環境と同じようなことができるように、クラウド上でのファイアウォール機能を設定します。

(2) アクセス制御

パブリッククラウドでは、サービスやリソースへのアクセスを制御する**認証機能**、アクセス管理機能が用意されています。ユーザーやグループを作成し、それぞれにアクセス権限を設定してサービスやリソースへのアクセスを許可または拒否します。但し、クラウド上でのアクセス制御は、かなりきめ細かく設定ができるようになっていることが多いです。そのため、ロールと呼ばれる機能のような、アクセス制御をまとめた単位を用意して制御を行うこともあります。

(3) 多要素認証・多段階認証

クラウドへのアクセスは一般的に、ID とパスワードの組み合わせによるログイン認証が使われますが、そこにもう 1 つの認証要素を組み合わせてセキュリティを強化するのが**多要素認証**（二要素認証）です。あらかじめ登録した固有のデバイス（スマートフォンなど）などにワンタイムパスワードを送付するというような認証機能が利用できます。クラウドの管理コンソールへのアクセス時などに使用してセキュリティを向上させます。

多段階認証とは、2 回以上の認証を行う仕組みのことです。多要素認証と似ていますが、多要素認証が 2 つ以上の異なる認証要素を組み合わせるのに対し、多段階認証は同じ認証要素（ID とパスワードの組み合わせなど）を繰り返し使う場合になります。
当然のことながら、複数の認証要素を組み合わせる多要素認証のほうが安全性は高まります。
クラウドサービス事業者がこれらの認証方法を提供していれば、その機能を使います。

1.11.1 オープンソースの概念とライセンス

Linux をはじめとして、すべてのソフトウェアは著作物です。そのため、ソフトウェアは、著作者がソフトウェアを使うためのライセンスを定義しています。オープンソースソフトウェアは、ライセンスが、「オープンソース・イニシアティブ（OSI)」によって策定された定義に沿っているものになります。
代表的なオープンソースソフトウェアには Linux、Apache HTTPD server、Apache Tomcat、Postfix、Samba、PostgreSQL などがあります。また、Linux 上で動作するユーティリティ類もオープンソースソフトウェアです。

1. OSI によるオープンソースの定義

OSI によるオープンソースの定義は、ソフトウェアのライセンスそのものではなく、ライセンスが満たす条件を定めています。

OSI が定義する条件は以下の通りです。

(1) 再配布を自由に認める
(2) ソースコードを無償で公開できる
(3) 派生物を自由に利用できる
(4) 原著作者のソースコードを明示する
(5) 特定人物・集団を差別しない
(6) 使用分野を差別しない
(7) ライセンスは平等に分配する
(8) 特定製品に限定したライセンスにしない
(9) 他のソフトウェアを制限するライセンスにしない
(10) ライセンスは技術中立でなければならない

この条件を満たすソフトウェアのライセンスは数多く存在します。ただし、OSI は、主要なライセンスの一覧を公開し、既存のライセンスを選ぶことを推奨してライセンスの氾濫を防いでいます。

2. オープンソースの特徴

OSI の定義に基づいたオープンソースソフトウェアは以下の特徴があります。

(1) 著作権

オープンソースソフトウェアは、著作物であり、設定されているライセンスに従って使えるものです。ライセンスの内容は種々あり、規定されている内容も異なります。

(2) 無保証

オープンソースソフトウェアは、基本的に「無保証」です。著作者は、ソフトウェアが予期した動作をするか、しないかの保証をせず、その動作によって何らかの損害を被ったとしても補償することはありません。

(3) 開発継続性

オープンソースソフトウェアは、そのソフトウェアのコミュニティを通じてメンテナンスや新機能開発を継続して行うことが可能です。商用ソフトのようにサポートの打ち切りによりソフトウェアが使えなくなることはありません。

3. オープンソースのライセンス

OSI の定義に沿ったオープンソースのライセンスは数多くあります。ここでは主なライセンスの内容について紹介します。

(1) GNU General Public License（GPL）

オープンソースの代表的なライセンスです。3 つのバージョンがありますが、現在使われているのはバージョン 2 とバージョン 3 です。Linux はバージョン 2 を採用しています。大きな特徴は、「コピーレフト」という考えを盛り込んでいることです。これは、GPL でライセンスされたソフトウェアを頒布する時には、ソースコードをつけて、再頒布をすることをライセンス上で規定しています。そのため、GPL ライセンスのソフトウェアを受け取った人は必ずソースコードを入手できます。オープンソースの特長である、ソースコードの公開という事を担保しているライセンスと言えます。

(2) GNU Affero General Public License（AGPL）

GNU Affero General Public License（AGPL）は、GPL とほぼ同じライセンスです。違いは、AGPL でライセンスされたソフトウェアをサービスとして使う場合にも、利用者にソースコードを提供する必要があることです。

(3) GNU Lesser General Public License（LGPL）

主にライブラリ用に作成されたライセンスです。LGPL では、利用者自身の利用のための改変ならびにそのような改変をデバッグするためのリバースエンジニアリングを認めるという条件をつければ、LGPL でライセンスされたライブラリを使うソフトウェアを独自のライセンスを設定できるようにしています。こうすることで、LGPL でライセンスされたライブラリを商用プログラムでも利用できるようになります。

(4) Mozilla Public License（MPL）

Mozilla Foundation によって作成されたライセンスです。MPL でライセンスされたソフトウェアは、必ずソースコードをつけて再頒布しなければなりませんが、ソースコードの提供がない、他のライセンスが適用されるソフトウェアと組み合わせることが可能です。

(5) BSD 系のライセンス

カリフォルニア大学が定めたライセンスです。カリフォルニア大学のバークレー校内の研究グループ Computer Systems Research Group が開発したソフトウェアなどで使われています。

「無保証」であることを明記することと、著作権およびライセンス条文自身を表示すれば再頒布することができるライセンスです。GPL のようにソースコードもいっしょに再頒布する必要はありません。

BSD 系のライセンスとしては、カリフォルニア大学が定めたライセンスのほかに、マサチューセッツ工科大学が定めたライセンス(MIT ライセンス)、Apache Software Foundation (ASF) によって作成された Apache License があります。

(6) パブリックドメイン

パブリックドメインは、著作権が発生していないか、失効した状態のことを言います。パブリックドメインというライセンスがあるわけではありません。

1.11.2 オープンソースのコミュニティとエコシステム

Linux をはじめとするオープンソースソフトウェアは、オープンソースコミュニティやオープンソースソフトウェアを利用したビジネスを展開する企業・組織のエコシステム（競争関係）を通じて継続的な開発が行われています。ここではオープンソースコミュニティとオープンソースエコシステムについて学びます。

1. オープンソースコミュニティとは

オープンソースコミュニティとは、オープンソースソフトウェアの開発・改善、情報交換などを目的に、さまざまな立場の有志によって構成された仮想の組織のことです。また、コミュニティには、特定の、あるいは複数のソフトウェアを開発するための、開発者が主に参加している開発コミュニティや、ユーザー同志の情報交換を中心とするユーザーコミュニティがあります。

コミュニティの運営

コミュニティには、あるオープンソースソフトウェアの開発者を中心としたボランティアベースの場合もありますが、企業やコミュニティを維持管理する団体が開発主体のものもあります。特に、大規模なオープンソースソフトウェアである場合、オープンソースソフトウェアを利用してサポートやサービスを提供するような場合、そして複数のオープンソースソフトウェアを抱えているようなコミュニティでは、企業が主体になって開発している場合や、非営利の法人が運用している場合があります。

コミュニティの開発基盤

コミュニティでは、メンバー間でソースコードや各種の情報を共有して共同開発するための開発サイトを使う事が普通です。開発サイトを使うことで、問合せやバグの管理、ソフトウェアやドキュメントの公開、ユーザー間での情報交換などが、簡単にでき、開発者やユーザーの負担を最小限にすることができます。

代表的な開発基盤としては、

　GitHub、Launchpad、GitLab

などがあります。

コミュニティへの参加

オープンソースコミュニティには、基本的に誰でも自由に参加して活動することが可能です。コミュニティへの参加方法は各コミュニティによって異なりますが、ほとんどの場合は

コミュニティの Web サイトから新規メンバーとして登録するだけで、すぐに活動を始められます。コミュニティにおける情報交換は、メーリングリストのほか、掲示板やメッセンジャーシステム(チャットなど)で行います。
但し、コミュニティへの貢献度によって利用可能な機能や権限について制限をしている場合もあります。

オープンソースコミュニティの例

オープンソースコミュニティは、非常に小規模なものから、全世界規模、開発者が数千/数万人というレベルのものまであります。ここでは、大規模なコミュニティの例を紹介します。

(1) Linux Foundation

Linux だけではなく、各種オープンソースコミュニティに対して、種々の施策を行って活動の支援をしている組織です。各種オープンソースプロジェクトの運営、コミュニティへの資金援助、インフラの提供、イベントの開催、トレーニングの提供などを行っています。
また、世界的なソフトウェア企業や通信会社、さらには自動車会社のようなユーザー企業がメンバーとして Linux Foundation を支えています。

(2) Apache Software Foundation

Web アプリケーションサーバー「Apache HTTP Server」をはじめとして、数多くのオープンソースプロジェクトを支援する非営利団体です。代表的なプロジェクトとしては、Java 実行環境「Apache Tomcat」、ロガー「Log4j」、パフォーマンス測定ツール「JMeter」など、数多くのオープンソースソフトウェアをホスティングしています。

(3) GNU プロジェクト

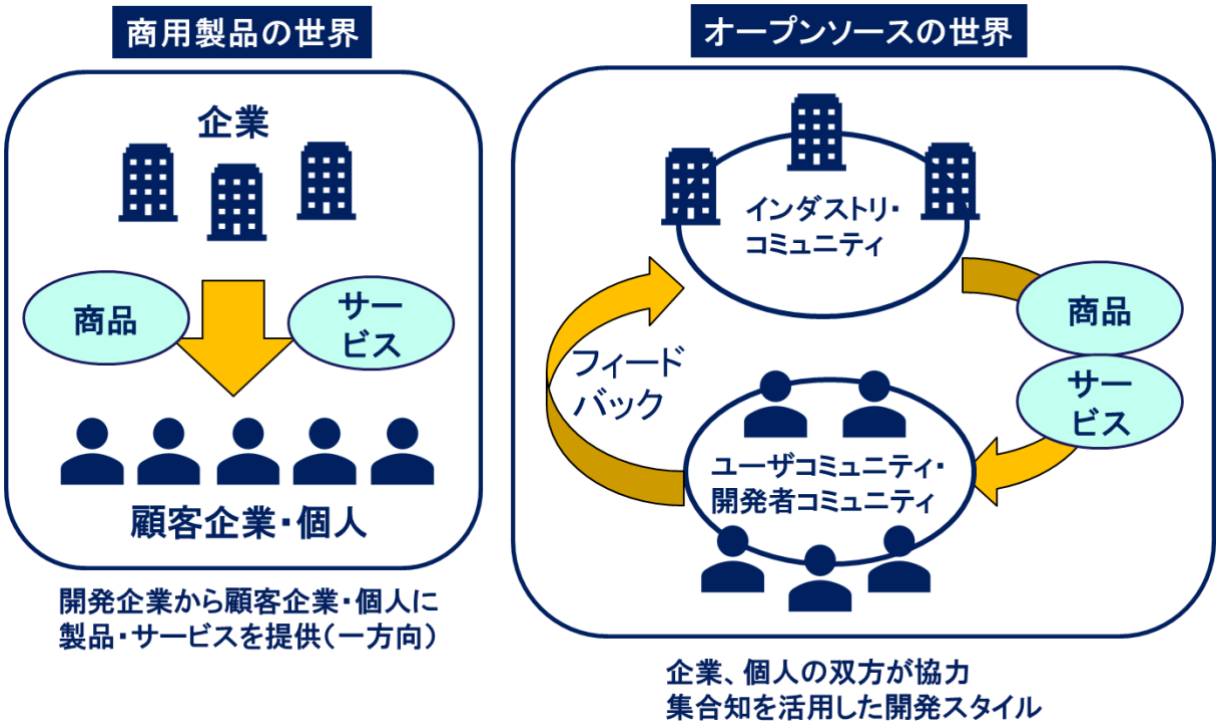
UNIX ライクな OS である「GNU」を開発するためにスタートしたプロジェクトです。OS 本体のほか、ユーティリティやアプリケーションなども開発しています。Free Software Foundation (FSF)が支援しています。

(4) OpenStack Foundation

OpenStack は、クラウドインフラストラクチャを構成するためのソフトウェア群です。その開発を行うコミュニティは、OpenStack Foundation によって管理されています。
OpenStack は、さまざまな機能を持つコンポーネントを持っています。それぞれに開発スキームが構成されていて、多くの人が開発作業に携わっています。

2. オープンソースエコシステムとは

エコシステムとは、複数の企業同士が、商品開発などで協力活動を行いつつ、開発者や流通、消費者、さらには業界や社会全体に渡って共存共栄を図っていく仕組みです。オープンソースソフトウェアの世界では、開発者とユーザーが協力し合ってコミュニティを形成し、開発していくという、開発の仕組みそのものがエコシステムになっています。さらに、オープンソースを使いやすいようにして流通させるディストリビューターや、オープンソースソフトウェアを活用して独自性のあるサポートやソリューションを提供するサービス企業も含めて、現在のビジネスの大きな流れになっています。
オープンソースエコシステムに参加した企業やユーザーが、実際にオープンソースを開発・使用、使用レポートの公表、バグ報告、さらには改良提案を行うといった活動によりエコシステムが好循環に発展し、それによりエコシステムに参加した各企業・ユーザーがメリットを享受できています。



KVM の実行環境についての注意事項

Linux の仮想化機能である、KVM を学習するために、実機ではなく、Windows の VMware Workstation Player や Windows や Mac の Oracle VM VirtualBox などの仮想化ソフトウェアを利用される方が多いと思います。この場合、KVM の実行環境を構築する際には注意事項がありますので下記に記載します。

1. ハードウェアの CPU が仮想化支援機能をサポートしていること

KVM を実行するハードウェアの CPU は、Intel であれば「Intel VT-x」、AMD であれば「AMD-V」と呼ばれる仮想化支援機能をサポートしていることが必要です。この機能は Intel の Core i プロセッサーや AMD の CPU で実装されています。KVM が実行できない場合、仮想化支援機能が BIOS で無効にされている場合がありますので、お持ちのハードウェアの BIOS 設定を確認してください。

2. 仮想化ソフトウェアの仮想マシン設定の注意事項

2.1 仮想マシンの設定について

仮想化ソフトウェアの仮想マシンの設定は下記以上を推奨します。仮想マシン上にさらに KVM の仮想マシンを作成するので、ホスト OS となる仮想マシンの性能が低いと KVM 上の仮想マシンの動作が極端に遅くなりますので、可能な限り下記以上のリソースを割り当ててください。

・CPU 数 1
・メモリ 4GB
・ハードディスク 40GB

2.2 仮想化ソフトウェア上で CPU の仮想化支援機能を有効化

仮想化ソフトウェア上で、KVM の機能を試す場合、仮想化ソフトウェアの CPU 仮想化支援機能が有効になっていないと KVM が動作しません。下記は VMware Workstation Player と Oracle VM VirtualBox の CPU 仮想化支援機能の有効化設定となります。

VMware Workstation Player の場合、仮想マシン設定のプロセッサー->仮想化エンジン->「Intel VT-x/EPT または AMD-V/RVI を仮想化」にチェックを入れます。

Oracle VM VirtualBox の場合、仮想マシン設定のシステム->プロセッサーの「ネステッド VT-x/AMD-V を有効化」にチェックを入れます。Oracle VM VirtualBox は v6.0.0 で AMD の CPU をサポート、v6.1.0 以降で Intel の第 5 世代 Core i プロセッサー以降をサポートしています。

※ただし、2020 年 3 月時点の最新版の v6.14 でも Intel Core i7 第 10 世代でチェックできませんでした。チェックできない場合は VMware Workstation Player で環境構築をしてみてください。

以上で仮想化ソフトウェアの仮想マシン上で CPU の仮想化支援機能が有効となります。

LinuC レベル 1 Version 10.0 学習教材（新規追加出題範囲）
2021 年 4 月 1 日 v1.0.2 版
LPI-Japan 発行