



LinuC レベル 2 Version10.0 学習教材
(新規追加出題範囲)

v1.0.2 版

【改訂履歴】

版	主な改訂内容（概要）
v1.0.0 (2020年5月1日)	新規作成
v1.0.1 (2020年9月3日)	2.06.2 Docker コンテナとコンテナイメージの管理 ・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修正 ・問い合わせ先の変更

本学習補助教材の位置づけ

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 2 Version 10.0 の出題範囲の中で過去の LinuC のバージョン（レベル 2：Version 4.5）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（Version 10.0 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利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、エンドユーザー企業、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事や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 2 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に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 エンジニアにとどまらず、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 2 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補助教材の位置づけです。

- 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
- 2.13：システムアーキテクチャ

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。クラウドや仮想化など、新しい技術要素が含まれるようになりましたが、これらの技術を使いこなす技術者をめざす方にお役に立てれば幸いです。

井上 博樹（執筆担当）

ソースコードからのビルドとインストール、監視ツール、システム構成ツール、仮想化サーバー、コンテナ、システムアーキテクチャについて執筆を担当しました。Git, Ansible による構成自動化、Zabbix による監視、KVM や Docker による仮想化など、システム管理者の負担を軽減する仕組みを紹介しています。みなさんのシステム構築・運用に役立てていただければ幸いです。

著作権

本教材の著作権は特定非営利活動法人エルピーアイジャパンに帰属します。

Copyright© LPI-Japan. All Rights Reserved.

使用に関する権利

本教材は、クリエイティブ・コモンズ・パブリック・ライセンスの「表示 - 非営利 - 改変禁止 4.0 国際 (CC BY-NC-ND 4.0)」でライセンスされています。



●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い合わせ先：

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 4.5)	7
201 試験出題範囲 新旧対照表	7
202 試験出題範囲 新旧対照表	8
新出題範囲 学習補助教材	10
2.04.1 make によるソースコードからのビルドとインストール (Git 部分のみ)	10
2.04.5 死活監視、リソース監視、運用監視ツール	15
2.04.6 システム構成ツール	26
2.05.1 仮想マシンの仕組みと KVM	34
2.05.2 仮想マシンの作成と管理	41
2.06.1 コンテナの仕組み	48
2.06.2 Docker コンテナとコンテナイメージの管理	53
2.13.1 高可用システムの実現方式	70
2.13.2 キャパシティプランニングとスケーラビリティの確保	77
2.13.3 クラウドサービス上のシステム構成	84
2.13.4 典型的なシステムアーキテクチャ	90
Appendix	98
A.1 Zabbix の概要	98
A.2 Pacemaker, Corosync のインストールと主な設定	115

出題範囲 新旧対照表 (バージョン 10.0 vs バージョン 4.5)

201 試験出題範囲 新旧対照表

バージョン 10.0			バージョン 4.5		
出題範囲		v.4.5 の範囲	出題範囲		v.10.0 の範囲
2.01 : システムの起動と Linux カーネル			202 : システムの起動		
2.01.1	ブートプロセスと GRUB	202.2 102.2	202.2	システムのリカバリ	2.01.1
2.01.2	システム起動のカスタマイズ	202.1	202.1	システムの起動をカスタマイズする	2.01.2
			202.3	その他のブートローダ	【削除】
			201 : Linux カーネル		
2.01.3	Linux カーネルの構成要素	201.1	201.1	カーネルの構成要素	2.01.3
2.01.4	Linux カーネルのコンパイル	201.2	201.2	Linux カーネルのコンパイル	2.01.4
2.01.5	Linux カーネル実行時における管理とトラブルシューティング	201.3	201.3	カーネル実行時における管理とトラブルシューティング	2.01.5
2.02 : ファイルシステムとストレージ管理			203 : ファイルシステムとデバイス		
2.02.1	ファイルシステムの設定とマウント	203.1	203.1	Linux ファイルシステムを操作する	2.02.1
2.02.2	ファイルシステムの管理	104.2 203.2 203.3	203.2	Linux ファイルシステムの保守	2.02.2
			203.3	ファイルシステムを作成してオプションを構成する	【削除】
			204 : 高度なストレージ管理		
			204.1	RAID を構成する	【削除】
			204.2	記憶装置へのアクセス方法を調整する	【削除】
2.02.3	論理ボリュームマネージャの設定と管理	204.3	204.3	論理ボリュームマネージャ	2.02.3
2.03 : ネットワーク構成			205 : ネットワーク構成		
2.03.1	基本的なネットワーク構成	205.1	205.1	基本的なネットワーク構成	2.03.1
2.03.2	高度なネットワーク構成	205.2	205.2	高度なネットワーク構成	2.03.2
2.03.3	ネットワークの問題解決	205.3	205.3	ネットワークの問題を解決する	2.03.3

2.04：システムの保守と運用管理			206：システムの保守		
2.04.1	makeによるソースコードからのビルドとインストール	206.1 【git追加】	206.1	ソースからプログラムをmakeしてインストールする	2.04.1
2.04.2	バックアップとリストア	206.2	206.2	バックアップ操作	2.04.2
2.04.3	ユーザーへの通知	206.3	206.3	システム関連の問題をユーザーに通知する	2.04.3
			200：キャパシティプランニング		
2.04.4	リソース使用状況の把握	200.1	200.1	リソースの使用率の測定とトラブルシューティング	2.04.4
2.04.5	運用管理ツールによる死活監視とリソース監視	200.2 【大幅改定】	200.2	将来のリソース需要を予測する	2.04.5 2.13.1
2.04.6	システム構成ツール	【新設】			
2.05：仮想化サーバー					
2.05.1	仮想マシンの仕組みとKVM	【新設】			
2.05.2	仮想マシンの作成と管理	【新設】			
2.06：コンテナ					
2.06.1	コンテナの仕組み	【新設】			
2.06.2	Dockerコンテナとコンテナイメージの管理	【新設】			

202 試験出題範囲 新旧対照表

バージョン 10.0			バージョン 4.5		
出題範囲	v.4.5の範囲		出題範囲	v.10.0の範囲	
2.07：ネットワーククライアント管理			210：ネットワーククライアントの管理		
2.07.1	DHCPサーバーの設定と管理	210.1	210.1	DHCPの設定	2.07.1
2.07.2	PAM認証	210.2	210.2	PAM認証	2.07.2
2.07.3	LDAPクライアントの利用方法	210.3	210.3	LDAPクライアントの利用方法	2.07.3
2.07.4	OpenLDAPサーバーの設定	210.4	210.4	OpenLDAPサーバの設定	2.07.4
2.08：ドメインネームサーバー			207：ドメインネームサーバ		
2.08.1	BINDの設定と管理	207.1	207.1	DNSサーバの基本的な設定	2.08.1
2.08.2	ゾーン情報の管理	207.2	207.2	DNSゾーンの作成と保守	2.08.2
2.08.3	セキュアなDNSサーバーの実現	207.3	207.3	DNSサーバを保護する	2.08.3

2.09 : HTTP サーバーとプロキシサーバー			208 : HTTP サービス		
2.09.1	Apache HTTP サーバーの設定と管理	208.1	208.1	Apache の基本的な設定	2.09.1
2.09.2	OpenSSL と HTTPS の設定	208.2	208.2	HTTPS 向けの Apache の設定	2.09.2
2.09.3	nginx の設定と管理	208.4	208.4	Web サーバーおよびリバースプロキシとしての Nginx の実装	2.09.3
2.09.4	Squid の設定と管理	208.3	208.3	キャッシュプロキシとしての Squid の実装	2.09.4
2.10 : 電子メールサービス			211 : 電子メールサービス		
2.10.1	Postfix の設定と管理	211.1	211.1	電子メールサーバの使用	2.10.1
			211.2	電子メール配信を管理する	【削除】
2.10.2	Dovecot の設定と管理	211.3	211.3	メールボックスアクセスを管理する	2.10.2
2.11 : ファイル共有サービス			209 : ファイル共有		
2.11.1	Samba の設定と管理	209.1	209.1	Samba サーバの設定	2.11.1
2.11.2	NFS サーバーの設定と管理	209.2	209.2	NFS サーバの設定	2.11.2
2.12 : システムのセキュリティ			212 : システムのセキュリティ		
2.12.1	iptables などによるパケットフィルタリング	212.1	212.1	ルータを構成する	2.12.1
2.12.2	OpenSSH サーバーの設定と管理	212.3	212.3	セキュアシェル (SSH)	2.12.2
2.12.3	OpenVPN サーバーの設定と管理	212.5	212.5	OpenVPN	2.12.3
2.12.4	セキュリティ業務	212.4	212.4	セキュリティ業務	2.12.4
			212.2	FTP サーバの保護	【削除】
2.13 : システムアーキテクチャ					
2.13.1	高可用システムの実現方式	【新設】			
2.13.2	キャパシティプランニングとスケーラビリティの確保	【新設】	200.2	将来のリソース需要を予測する	2.09.5 2.13.2
2.13.3	クラウドサービス上のシステム構成	【新設】			
2.13.4	典型的なシステムアーキテクチャ	【新設】			

新出題範囲 学習補助教材

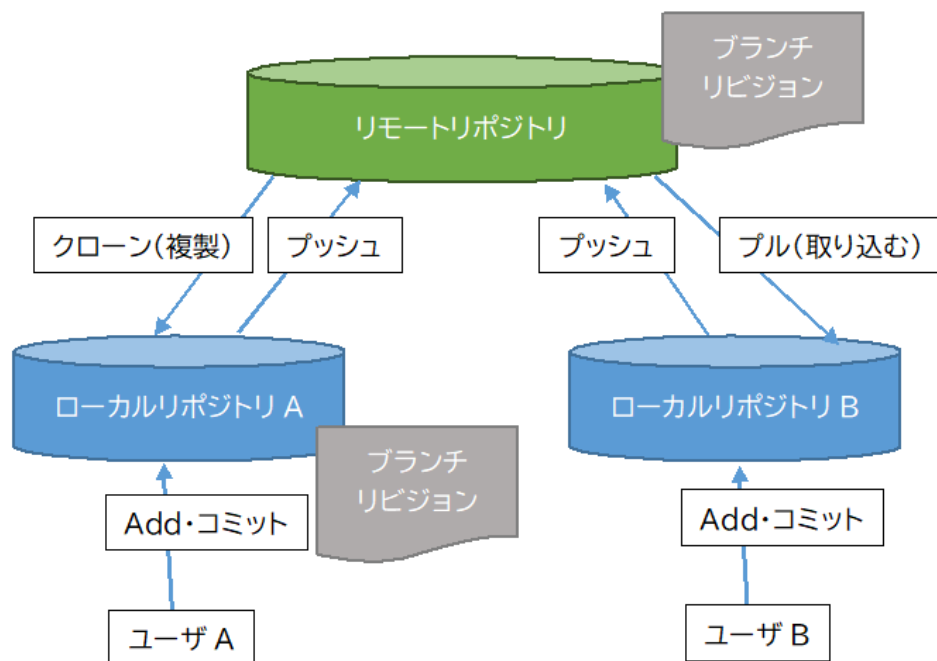
2.04.1 make によるソースコードからのビルドとインストール

1. Git の概要

Git は Linux カーネルを開発しているリーナス・トーバルズ氏が開発した分散型のバージョン管理システムです。現在、多くのプロジェクトで採用され、バージョン管理やチームによる共同開発などで使用されています。

Git では、単一のリポジトリ (サーバー上の特定のフォルダ) を全員で共有するのではなく、ユーザーごとにリポジトリのローカルコピー (ローカルリポジトリ) を作成し、すべての編集・更新履歴を管理します。このため、ユーザーごとに、オフラインでもバージョン管理が可能です。

Git 以外のバージョン管理システムとしては CVS (Concurrent Versions System) や SVN (Subversion) などがあります。これらは単一のリポジトリを使用するところが Git と異なります。



Git リポジトリの連携イメージ
(リモートリポジトリで変更履歴を共有する)

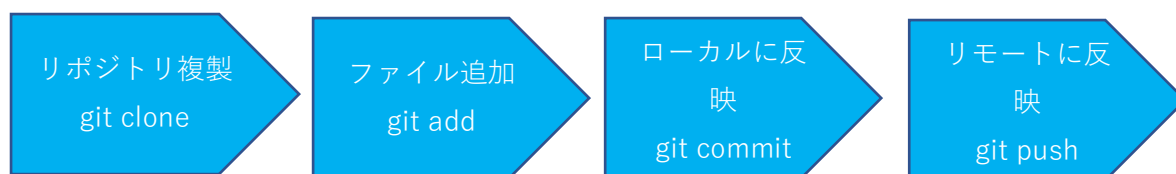
2. Git でできること

Git を使うと、以下のようなことができます。

- ファイルの変更履歴の管理ができる
- 変更履歴を参照してバージョンを過去のバージョンに戻ることができる
- ソースコード以外のデータファイルも管理できる
- チームで変更履歴を共有し、衝突を防ぐことができる

そのためソフトウェア開発だけではなく、ドキュメントや Web デザインなどの分野でも使用されています。

全体的な操作の流れは以下のようなイメージになります。



3. 主要な Git コマンド

3.1 ローカルリポジトリの作成 (git init)

Git を使用するにはリポジトリの作成が必要です。以下の `git init` コマンドを実行すると、ローカルホスト上に「.git」ディレクトリが追加されて、履歴などを管理するためのファイル群が作成されます。

```
$ cd <リポジトリを新規に追加するディレクトリ>  
$ git init
```

3.2 リモートリポジトリのクローニング (git clone)

既存のリポジトリにあるプロジェクトをコピーして開発に参加したり、成果物を使用したりするには、まずプロジェクトのリポジトリをコピー（クローニング）します。

```
$ cd <リポジトリを新規に追加するディレクトリ>  
$ git clone リポジトリの URL
```

とすると指定した URL に存在するリポジトリ内のファイル群が、フォルダごとローカルマシン上のカレントディレクトリにコピーされます。

コピーしたローカルリポジトリ (フォルダ) 内のファイルを更新したり、追加したりします。

3.3 変更履歴の保存・コメントの追加 (git add, git commit, git tag)

編集を終えたら、

```
$ git add ファイル名
$ git commit -m "コメント"
```

としてローカルリポジトリに変更履歴を反映します。

また、

```
$ git tag <タグ名>
```

とすると直前のコミットに対してわかりやすい別名をつけることができます。例えば、ソフトウェアの場合だと、アルファ版、ベータ版、リリース候補 (Release Candidate)、公開版 (General Availability)、Stable (安定版)、あるいはバージョン番号などのタグをつけておくと、他のチームメンバーがバージョン情報やステータスをチェックしやすくなるでしょう。

3.4 公開リポジトリに変更を送信する (git push)

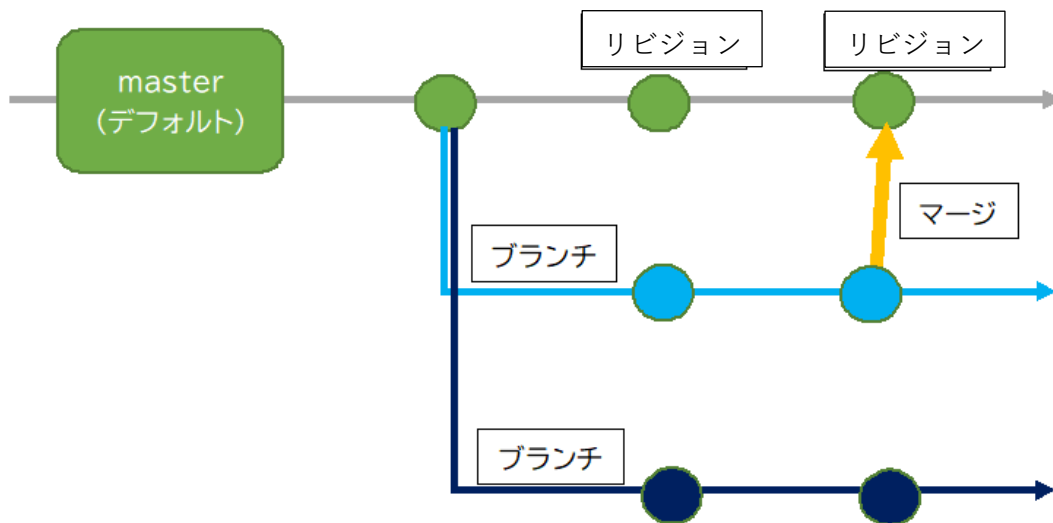
commit した変更をリモートリポジトリに反映するには push コマンドを使用します。

```
$ git push 反映先 (リモートリポジトリ) ブランチ
```

リポジトリには、バージョン管理のためにブランチとリビジョンという仕組みがあります。ブランチはメインのリポジトリから分岐 (フォーク) したもので重複しない名前をつけます。

ブランチ内ではさらにリビジョンをつけて、ブランチ内でのバージョン管理を行います。

ブランチ名を指定することで、リモートリポジトリ上のブランチを切り替えてクローニング (複製) することが可能です。



master ブランチを分岐して、ブランチ内でリビジョン管理をするイメージ

3.5 ブランチを切り替える (git checkout)

```
$ git checkout <ブランチ名>
```

3.6 タグで絞り込んで表示する (例えばタグに v1.9.5 を含むものだけを表示)

```
$ git tag -l "v1.9.5*"
```

オプションをつけないと、すべてのタグが表示されます。

4. リモートリポジトリからのクローニングとソースコードのビルド

それでは、実際にリポジトリをクローニング（複製）して、ソフトウェアをビルドしてみましょう。ここでは、コンピュータービジョンのソフトウェアで有名な darknet のビルドにトライします。

```
$ git clone https://github.com/pjreddie/darknet.git
$ cd darknet
```

リポジトリのブランチを指定しないと、デフォルトでは master ブランチ（最新安定版）が選択されます。もしも、リリースブランチを指定してダウンロードしたい場合には、checkout コマンドを使います。

```
$ git checkout <ブランチ名> # r1.10 など
```

リポジトリをクローニングしたら、ローカルリポジトリのディレクトリに移動して make コマンドを実行します。

```
$ cd darknet
$ make
```

無事にコンパイルが完了したら

```
$ ./darknet
```

としてコマンドを実行してみましょう。

```
usage: ./darknet <function>
```

とコマンドの使用方法（usage）が表示されたら、ビルドに成功しています。

2.04.5 死活監視、リソース監視、運用監視ツール

現在、Web サイトや SNS、サーバーと連携するスマホアプリなどインターネット上で提供されるサービスや、企業内のファイルサーバー、メールサーバー、グループウェアなどのシステムは、24 時間 365 日いつでも問題なく使用できることが当然だと考えられています。

しかし、問題なくシステムを連続稼働させることは当然ではありません。それなりの対策が必要です。例えば、ハードウェア、ネットワーク、OS など、システムを構成するさまざまな要素で障害が発生する可能性があるため、何らかの障害が発生してもシステム全体を停止せずに運用を継続させる仕組みが必要となります。

そのため、障害発生時のシステムやデータ復旧、サービス再稼働までの時間を短縮し、サービスのダウンタイムをできるだけ短くしてユーザーの利便性を損なわないためには、システムやサーバーを的確に監視してサービスが停止しないよう、障害をできるだけ早期に検知して対応することが重要です。

1. システム障害と予兆

システムの障害を引き起こす原因には以下のようなものが考えられます。

1.1 リソースの枯渇

メモリ消費量が実メモリ容量を超えてスワップ領域へのアクセスが多発すると、OS のパフォーマンスが大幅に落ちます。

また、ディスク容量を使い果たしてしまうと、それ以上データを書き込んだりできなくなるためにシステムが停止してしまうトラブルが発生します。

1.2 過負荷

アプリケーションの処理などで CPU 使用率が 100% に達するとシステムが応答しなくなったり、アプリケーションの処理が遅延/停止して、ブラウザやアプリへの応答がなくなり、処理結果が表示されない、などのトラブルを引き起こします。

1.3 異常停止

システム上のどこかで障害が発生した結果、システムやサービスが停止する場合があります。

1.4 結果異常

サービスが返す処理結果が異常な値を示していたり、データの破損により、システム障害が発生する可能性があります。

1.5 OOMKiller

Linux はメモリが不足 (Out-Of-Memory : OOM と略す) してシステム停止が発生してしまう状況になると、メモリリソースを大量に消費しているプロセスを強制終了 (kill) します。

停止処理をしていないプロセスが存在していない場合は、OS によって強制終了された可能性があります。Centos では、/var/log/messages、Ubuntu なら/var/log/syslog の各 OS のシステムログに、強制終了した情報が記録されます。

強制終了される可能性があるプロセスは、dstat コマンドの top-oom オプションで確認できます。dstat コマンドは、/proc/PID/oom_score を参照して結果を表示します。

```
$ dstat --top-oom
--out-of-memory---
  kill score
sshd          594
sshd          594
```

強制終了させるプロセスの優先度を設定しておくこともできます。

/proc/PID/oom_adj に、-16 から+15 までの優先度を設定することができます。

-17 を設定すると強制終了の対象リストから外されます。

2. 死活監視の対象と手法

死活監視の対象としては、

- サーバー
- ネットワーク機器
- サーバー上の特定のサービス

などがあります。

システムに対しての死活監視は、例えばサーバーのネットワークインターフェースに対して疎通確認を行い、応答 (レスポンス) が返ってくるかどうかを確認します。

疎通確認が取れない場合には、サーバーが何らかの原因で停止しているか、ネットワークインターフェースに障害が発生しているなどのケースが考えられます。

また、各種サービスの死活監視については、Web サーバー、アプリケーションサーバー、メールサーバー、ファイルサーバーなどのプロセスが起動しているかを確認します。プロセスが起動していないとサービスが提供されません。また、CPU やメモリ、ディスクなどが不足するとサービスの提供に影響が出るので、次のリソース監視もあわせて実行します。

3. リソース監視の対象と手法

リソース監視の対象には、

- ログ（サーバー、サービス、プロセス、ネットワーク）
- 使用率（CPU、メモリ、ストレージ、通信量）

などがあります。

3.1 ログの監視

サーバーやネットワーク機器などのログファイルを `tail` コマンドなどを用いてコンソール上でリアルタイムに表示して確認することもできますが、ログ監視ツールや統合監視ツールなどを用いる方が管理者の負担が軽減できます。

ログ監視ツールを用いると、指定したディレクトリに保存されているログファイルをリアルタイムに監視して、特定のパターンにマッチしたら、メールでの通知を送信したり、スクリプトを実行することができます。

監視する対象としては、以下のようなログがあります。

- サーバーのシステムログ
- サービスごとのログファイル
例えば、`httpd` や `nginx` のログファイルにはサービスの起動、停止、エラーのログが記録されます。
- プロセスやネットワークの監視ログ
監視ツールから、定期的にはプロセスの起動をチェックしたり、ネットワークの疎通確認を行った記録がログファイルに記録されます。

3.2 SNMP によるリソース監視

サーバーやネットワーク機器を監視するには、SNMP（Simple Network Management Protocol）を用いたリソース監視も使われます。これは、IP アドレスが設定されているさまざまなデバイスから情報を取得し、動作の状況や、障害の状況など、デバイスの状態を監視するための仕組みです。

監視対象としては、

- CPU、メモリ、ネットワークインターフェース、サーバー筐体の温度などの状態
- HDD、電源、ファンなどのハードウェアの稼働状況
- ネットワークの稼働状況（L2 ループ、マルチキャストやブロードキャストパケットの多重ループ、が発生していないか）

などがあります。

また、監視には、

- SNMP マネージャから SNMP エージェントへのポーリング(定期的なリクエスト送信)
- SNMP エージェントから SNMP マネージャへの通知

の2タイプの仕組みがあります。

例えば、オープンソースの監視ツールである Nagios には SNMP マネージャの機能が内蔵されています。そして、監視対象のサーバーやデバイスでは SNMP エージェントのプロセスが動作しています。SNMP マネージャは、SNMP エージェントに対して一定間隔（例えば3分おき）にリクエストの送信と、エージェントからのレスポンス受信を行います。

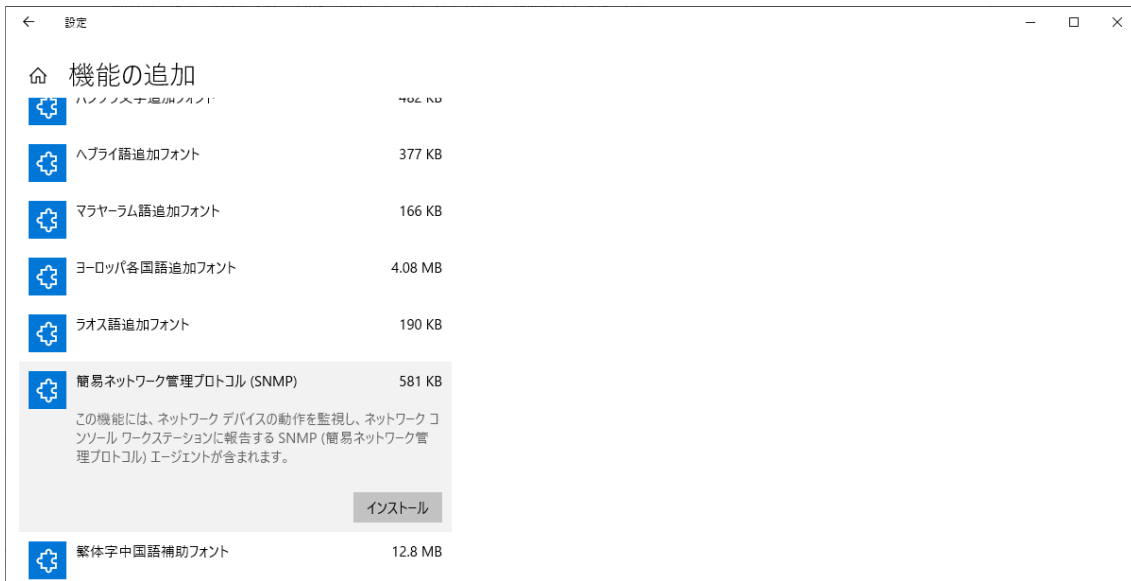
SNMP エージェントは、多くのルーターやスイッチなどのネットワークデバイスには組み込まれていることが多く、設定をすることで利用が可能になります。

Linux では、net-snmp パッケージにエージェントが含まれています。パッケージをインストールし、設定を行うことで利用が可能になります。

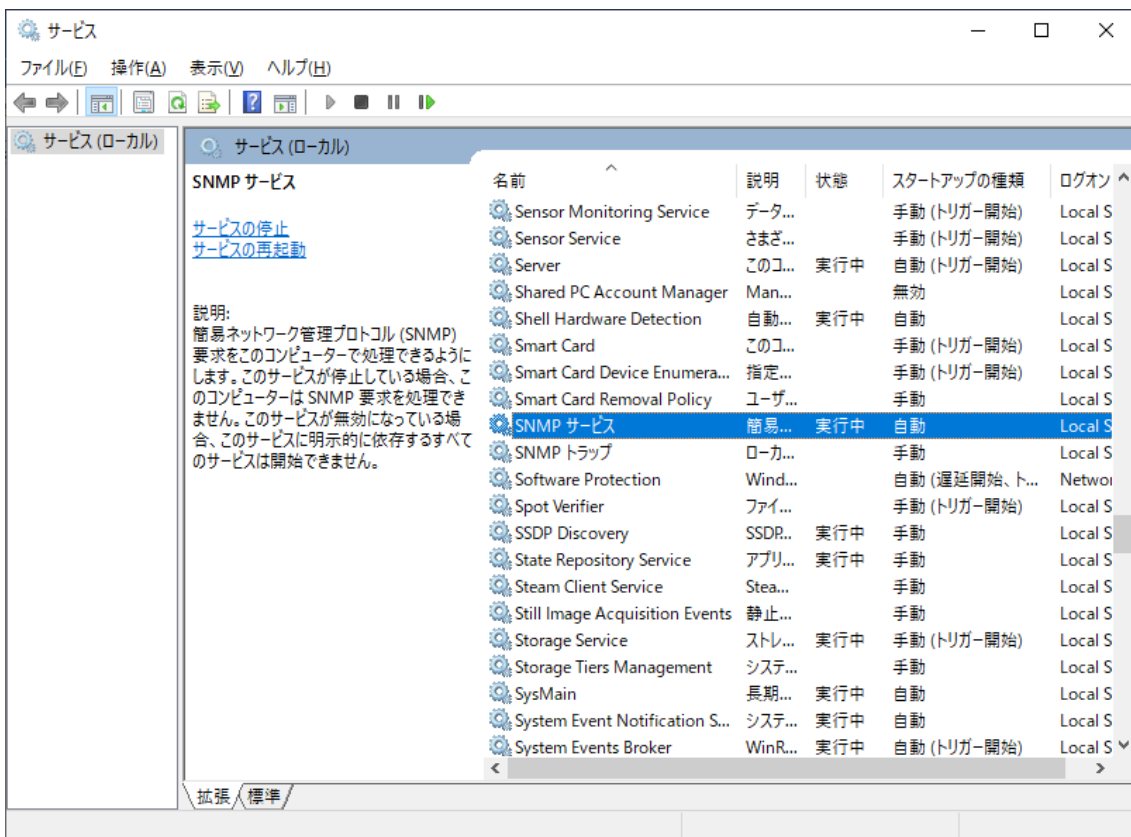
Windows マシン¹では「SNMP Service」を追加・設定して起動すれば SNMP エージェントが使えるようになります。

アプリと機能の追加の設定メニューから「簡易ネットワーク管理プロトコル (SNMP)」を選択して、インストールを実行すると、サービスの一覧に表示されます。

¹ Windows 10 Pro version 1903 で確認



SNMP エージェントの追加



SNMP サービスの起動・停止画面

4. 運用監視ツールによる監視作業の標準化と自動化

サービスやデバイスごとに監視スクリプトを開発・実行し、ログ監視やリソース監視を手動で行うことはシステム規模が大きくなると負担が大きくなり、現実的ではありません。

そこで、監視ツールによる監視作業の標準化や自動化が重要になります。

例えば、オープンソースの監視ツールである Zabbix には以下のような監視、障害検知機能が搭載されています。

- ネットワーク上のノード監視
ICMP (ping) による死活監視
- リソース監視
CPU、メモリ、ディスク使用率
- ネットワークデバイス監視
監視ツールのエージェントプロセスや SNMP エージェントによる監視
- プロセス監視
ホスト上で動作しているプロセスの監視
- ポート監視
指定したポートを通じた通信が可能かどうかの監視
- サービス監視
HTTP, FTP, SMTP, POP, IMAP, LDAP, SSH, NNTP, NTP などのサービスの動作監視
- ログ監視
ログファイル中に指定した文字列パターンが含まれないかを監視
- Windows 監視
Windows 用エージェントを使用すると Windows のサービス起動状況やイベントログを監視できます。複数のエージェント対応、ソフトウェアのインストール・アンインストール、アプリケーションの起動・停止など Windows 版固有の機能が搭載されています。

また、それぞれの項目についてしきい値を設定し、Eメールなどによるアラート通知方式を設定することが可能です。

4.1 監視方式の自動化や RBA によるパッケージ化

さらに、単一のノードの監視だけでなく、複数台のノードをまとめて監視したり、複数の処理を自動的に実行したりする自動化機能が備わっている監視ツールもあります。

こうした自動化機能は、RBA (Run Book Automation) と呼ばれ、問題や障害を検出した際にあらかじめ定義しておいた作業を自動的に実行することができるようになります。

4.2 監視業務の標準化の重要性

システム監視業務プロセスの改善にあたっては、既存の監視業務プロセスが効率的に行われているかどうかをチェックするところからスタートするとよいでしょう。

監視作業の自動化以前に、まず監視業務プロセス全体の見直しと標準化を行う必要があります。無駄な作業はないか、より効率を向上できないかなどをチェックしてから、監視プロセスの標準化を進めることが重要です。プロセスが標準化されれば、監視対象が増えても担当者の負担を軽減したり、作業ミスや運用の負担やコストを軽減したりすることにつながるでしょう。

標準化を進めることで監視ツールを統一したり、RBA による自動化を適用したりしやすくなります。その結果、全体最適化を進めることが可能となるでしょう。

5. 主要な監視ツール

5.1 Icinga2

Icinga2 は、Nagios(後述)から派生（フォーク）して開発されたオープンソースの監視ツールです。Icinga2 を用いると Web インターフェース、もしくはコマンドラインインターフェースから監視設定を行うことができます。

標準的な監視機能（システムやリソースの監視、通知）に加えて、監視結果の可視化(動的なグラフ生成)や、分散構成への対応などの特徴があります。

5.2 Nagios

Nagios もオープンソースの監視ツールの一つです。ネットワークサービスの監視、リソース監視、アラート機能など標準的な機能を備えています。基本的な設定は Web インターフェースから行います。

Nagios はプラグインをユーザーが開発・追加できる点が特徴的です。Bash, C++, Perl, Ruby, Python, PHP, C#などで、監視したい対象についてプラグインを開発して使用することができます。

現在は、機能を強化した商用版と、オープンソース版が存在しています。新機能は商用版に主に追加されるため、Icinga のように分岐して新機能の開発を独自に進めるプロジェクトも出てきています。

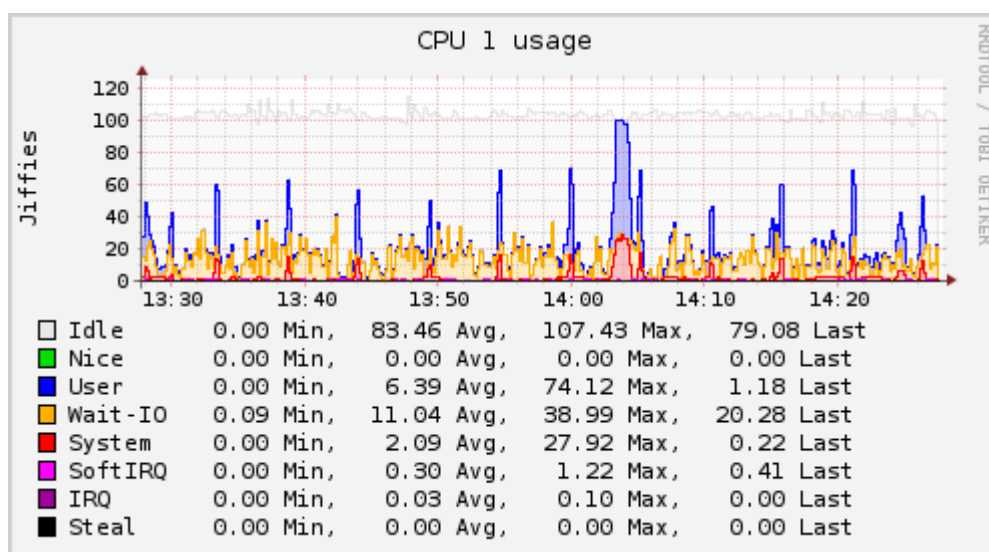
5.3 collectd (<https://collectd.org/>)

collectd は、システムやアプリケーションのパフォーマンス情報を収集するオープンソースの監視ツールです。collectd は、100 個以上の豊富なプラグインを使用して、システ

ムやリソースの監視を行えます。収集したデータは、RRD というフォーマットで保存されます。

collectd は C 言語で書かれているため、パフォーマンスが高く、実行ファイルのサイズが小さいのが大きな特徴です。そのため、一般的なサーバー以外にも組み込み系システムやルーターなどにも搭載可能です。

一方で、グラフ生成機能は含まないため可視化ツールを併用する必要がある、モニタリング機能が限定的である、という短所もあります。



collectd が収集したデータを可視化した例²

5.4 MRTG

MRTG は、「Multi Router Traffic Grapher」の略で、ルータなどネットワークデバイスが送受信したデータトラフィック量を可視化するために使用されるオープンソースの監視ツールです。

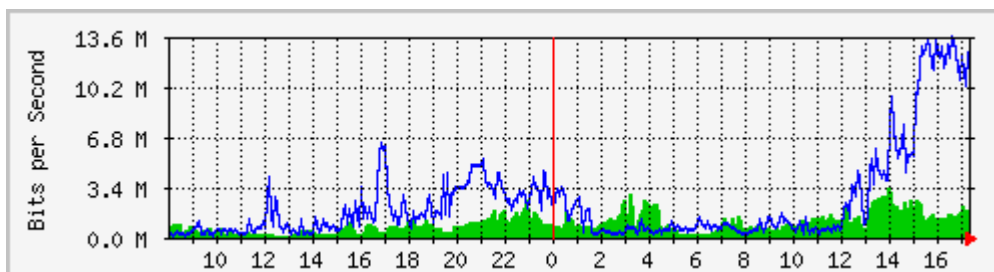
MRTG は Perl 言語で書かれていて、取得した情報を HTML 形式で出力可能なため、Web サーバーと連携して結果を表示させることができます。

もともとはルータを監視対象として開発されましたが、現在は SNMP マネージャとして動作させることが可能で、SNMP エージェントと連携してサーバーやネットワークデバイスの監視が可能となっています。SNMP に対応しているため、ネットワークトラフ

² <https://collectd.org/>

ックだけでなく、サーバーのリソース状況（CPU、メモリ、ディスクなどの使用率）なども監視することが可能です。

欠点としては、二つの系列データの可視化までしかできないなど、可視化機能が弱い点が挙げられます。



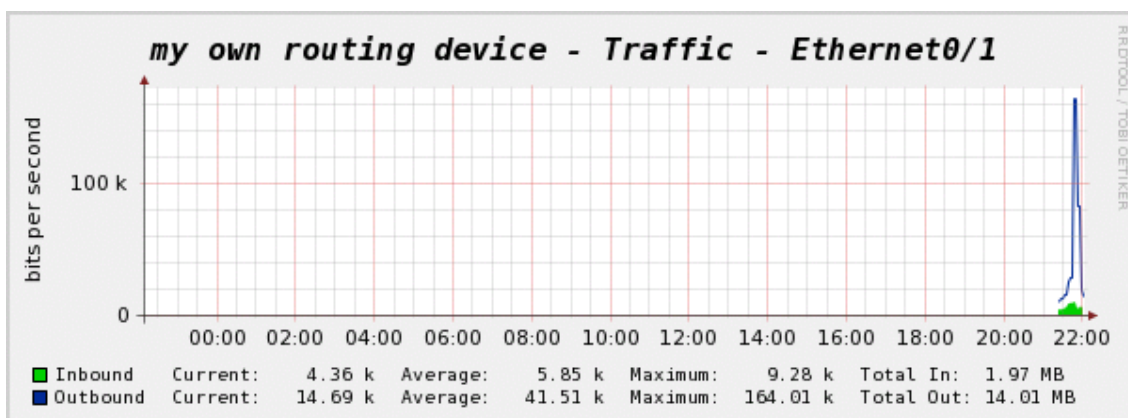
MRTG で送受信のデータトラフィックを可視化した例³

5.5 Cacti

Cacti は、オープンソースの監視ツールの一つで SNMP マネージャとして機能することが可能です。

Cacti は collectd や MRTG が採用している RRD フォーマットのデータを扱うことができるだけでなく、SNMP 対応や Web インターフェースによる設定の容易さなどが特徴です。

CentOS では、EPEL リポジトリを追加すると yum コマンドでインストールすることが可能です。net-snmp, net-snmp-utils の二つのパッケージをインストールすると、`/etc/snmp/snmpd.conf` で設定を行うことができますようになります。



Cacti で生成したグラフ（ルーターのトラフィック）⁴

³ <https://oss.oetiker.ch/mrtg/>

⁴ https://docs.cacti.net/manual:100:2_basics.1_first_graph#creating_the_graphs

5.6 Zabbix

Zabbix は現在非常に人気のあるオープンソースの監視ツールの一つです。バックエンドは C 言語で書かれています。Web インターフェースは PHP 言語で書かれています。2020年4月現在、最新安定版は 4.4.1、LTS（長期サポート版）は 4.0LTS です。

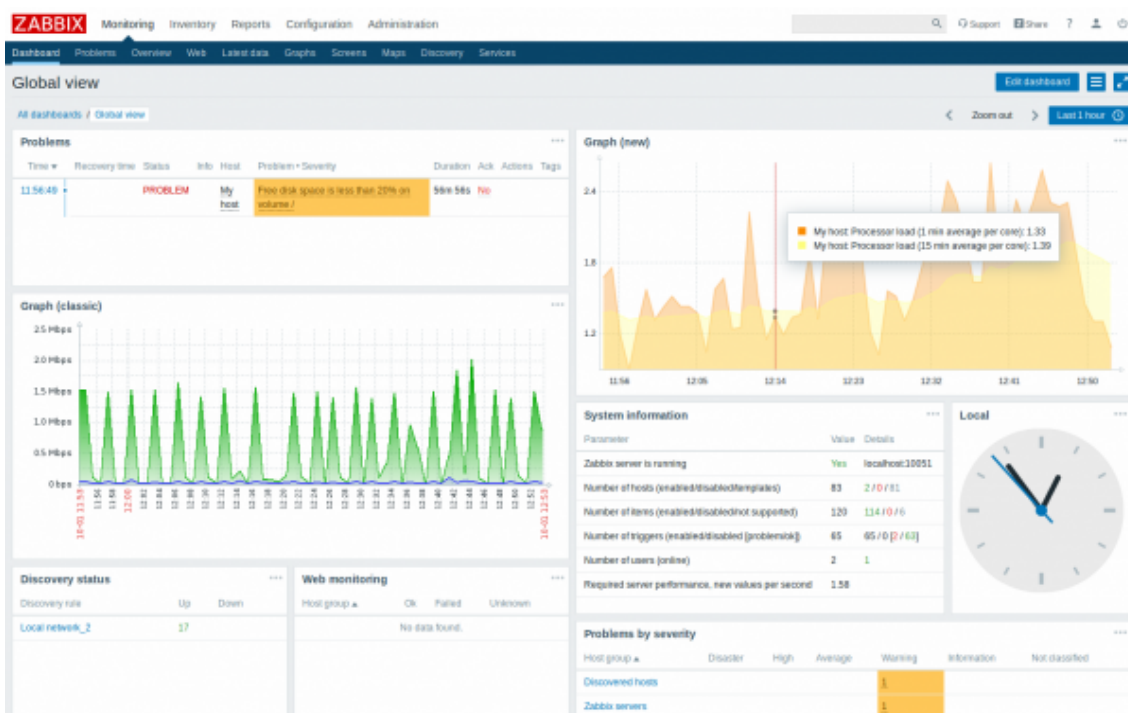
Zabbix はデータベースに収集した情報を格納するため、MySQL や PostgreSQL などのリレーショナルデータベースと連携して動作します。

Zabbix では、

- ICMP(ping)、SMTP、HTTP などによるサービスの稼働状況の確認
- Zabbix エージェントを監視対象にインストールして、ホストのリソース監視 (CPU、メモリ、ディスクなどの使用率) の監視

などの、複数の監視方法に対応しています。エージェントをインストールしなくても多数のホストやデバイスの監視に対応しているところが特徴的です。

Zabbix の詳細については、Appendix で取り上げますが、以下のようなダッシュボード機能を提供し、監視対象を一覧表示することができます。



Zabbix のダッシュボード⁵

⁵ https://www.zabbix.com/documentation/4.0/manual/web_interface/frontend_sections/monitoring/dashboard

また、各ターゲットの詳細情報はウィジェットという四角形の領域に表示されます。例えば、以下は CPU 使用率の監視結果を可視化したグラフです。



Zabbix で CPU 使用率の推移を可視化したグラフの例⁶

⁶ https://www.zabbix.com/documentation/4.0/manual/web_interface/frontend_sections/monitoring/graphs

2.04.6 システム構成ツール

1. システム構成ツールの必要性と導入メリット

IT システムを構築するにあたり、サーバーへの OS インストール、ルーターなどネットワーク機器の設定が、インフラ構築作業では必要です。

しかし近年ではさらにシステムの大規模化やクラウド化が進んだり、マイクロサービスアーキテクチャによる処理の分散化が進行したりしてきました。その中で、アプリケーションのコンテナ化や、DevOps の進化により、アプリケーションの開発からデプロイ（本番環境への配置）が高速になり、短期間にアプリケーションがリリースされるようになってきました。

管理すべきサーバーやアプリケーション、コンテナなどの数が増大し、さらに開発から運用までの期間も短縮されたことで、以前のように手作業で 1 台ずつ設定作業をしていると、サービスの提供や運用管理が間に合わなくなったり、人的エラーが発生したりしてしまうリスクが出てきました。

こうしたニーズやリスク低減に対応するために各種設定作業もコード化しておいて、作業を自動化しようという取り組みが行われるようになります。こうした作業のコード化を、Infrastructure as Code (IaC) と呼んでいます。

作業を自動化することで、

- サーバー設定などのプロビジョニング作業を標準化できる
- より効率よく、ミスなく作業ができる
- 台数が多くなっても短時間で作業が可能になる
- 同じ設定を間違えずに行える

などのメリットが生まれます。

◎ 冪等性（べきとうせい）

設定作業を何度も実行しても同じ結果が得られる特徴のことを「冪等性（べきとうせい）、idempotency」と呼びます。

構成管理ツールには、正規表現によるパターンマッチングを行い無駄な設定を追加しないようにする機能や、設定は行わずテストを行う dry run 機能などがあり、誤った設定や余分な設定を行わないような工夫がされています。

冪等性が確保されることで、システムが大きくスケールした際にも、設定ミス回避したり、作業負荷を軽減することが可能になります。

2. Ansible の概要

作業の自動化を実行するには、構成管理ツールを使用します。

構成管理ツールには、いろいろなソフトウェアが存在します。Puppet, Chef, Salt, Ansible などが広く使用されています。

このうち、Puppet や Chef などは管理対象となるマシンにエージェントと呼ばれるクライアントソフトウェアをインストールし常時稼働させておく必要があります。それに対し、Ansible では (Linux を含む Unix 系 OS の場合) SSH 接続が可能であれば、管理対象のマシンにエージェントを常駐させておく必要がないため、エージェントレスな構成管理ツールと呼ばれています。

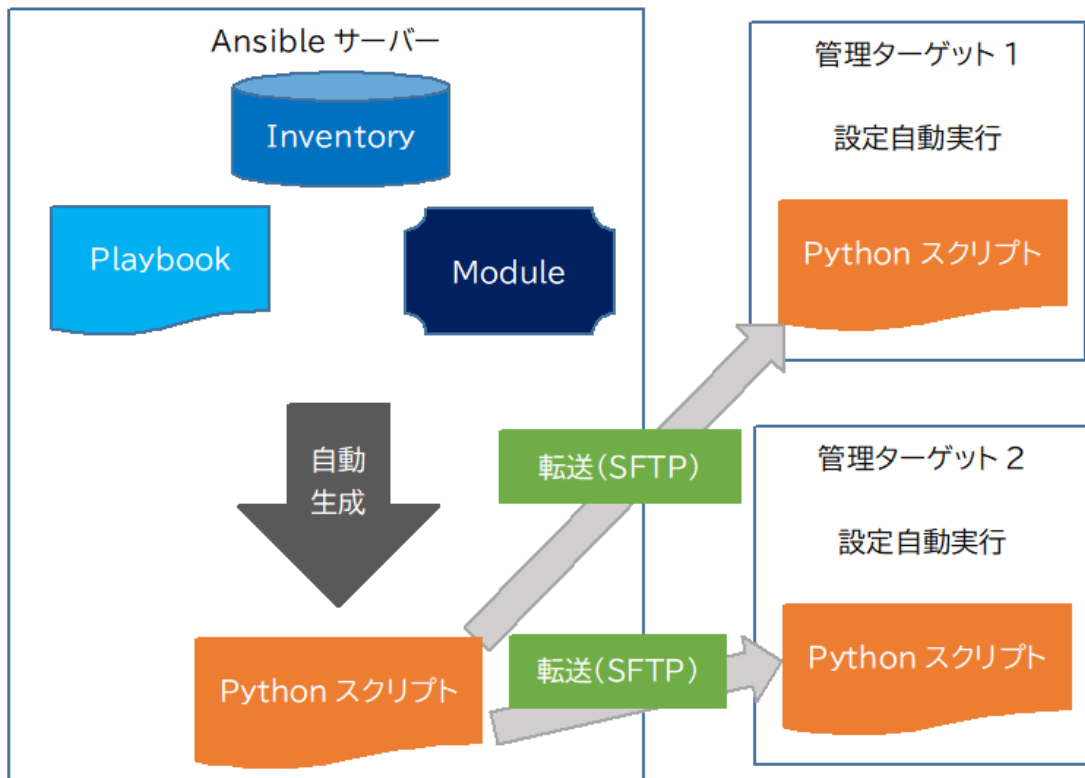
Ansible はオープンソースで開発・公開が進められている構成管理ツールで、Python 言語で記述されています。

YAML という所定のフォーマットで記述された二つの設定ファイル (構成を定義したテキストファイル) に従って、サーバーやネットワークデバイスなどの設定を自動的に実行できます。

Ansible の動作環境は以下のようになっています。

- 設定ターゲットを定義するインベントリファイル (Inventory file)
- 設定作業を記述するプレイブック (Playbook)
- 処理の最小単位 (Module)

Ansible のコマンドを実行すると、インベントリファイルに定義したホストに Playbook が Python スクリプトに変換されます。そして、変換されたスクリプトと、処理を実行するための Module がターゲットのホストに転送され、転送先のホストで Playbook の記述にしたがって、各種処理や設定が実行されます。



Ansible の動作イメージ

さらに構成管理だけでなく、アプリケーションの配置（デプロイ）や、コンテナの構成などにも対応するようになってきています。

3. ローカル環境への Ansible のインストールと対象ホストの設定

Ansible を CentOS7 にインストールするには、まず EPEL リポジトリを追加します。これは Ansible が CentOS の公式リポジトリに含まれていないためです。

```
# yum install epel-release
```

リポジトリを追加したら、yum コマンドで ansible をインストールします。

```
# yum install ansible
```

すると、`/etc/ansible/ansible.cfg` に設定ファイルが生成されます。そして、Ansible を実行するホストを定義するには、この設定ファイルで `inventory` の定義行の先頭の `#` を削除して有効化します。

```
inventory = /etc/ansible/hosts # 先頭のシャープを取ります。
```

続いて、`/etc/ansible/hosts` に、Ansible で設定を行うホストの情報を記述します。

```
[www1]
192.168.1.10
```

複数台を指定するには 1 行に一つのホストの情報を入力します。

4. Ansible Playbook の設定と実行

YAML は各行に、「キー： 値」というフォーマットで設定を記述していきます。

例えば、Ansible で構成を行うホストに `httpd` をインストールするには、`playbook.yml` に以下の記述を行います。

```
- hosts: www1
  remote_user: root

  tasks:
  - name: yum install
    yum: name=httpd
```

設定する先 (hosts) と Playbook ファイルを記述したら、以下のように `ansible-playbook` コマンドを実行して設定作業を実行します。

```
$ ansible-playbook -i inventory playbook.yml
```

設定するターゲットの `root` ユーザーの `ssh` ログインのパスワードを求められるので入力すると Playbook から自動生成された Python スクリプトが転送され、実行されます。

このようにして、設定対象となるホストと、設定処理の手順をあらかじめ記述しておくことで、各種設定を自動的に実行することができます。

5. システムの構成変更の自動化

Ansible などの構成管理ツールを用いると、既存のサーバーの設定変更だけでなく、コンテナの管理や設定変更などにも対応できます。

5.1 仮想サーバー・コンテナの払い出し

Ansible は、Docker コンテナのイメージの作成、Kubernetes への展開などが可能になります。例えば、ansible-bender や Ansible Operator があります。

コンテナの作成、起動だけであれば Docker Composer でも実現できますが、上記のようなツールを用いると、あらかじめ記述しておいた Playbook にしたがって設定作業も実行することができるのが利点です。

他にも、以下のような操作を実行できます。

- 既存イメージ、またはスクラッチからコンテナを生成する
- 既存コンテナまたは Dockerfile からイメージを生成する
- コンテナの root ディレクトリをマウント・アンマウントして操作できるようにする
- コンテナやイメージを削除する
- ローカルコンテナをリネームする（別名をつける）

5.2 アプリケーションのリリース

構成管理ツールを用いると、仮想マシンの生成だけでなく、アプリケーションを配布したり、もしくはアプリケーションを含むコンテナイメージからコンテナを生成し、アプリケーションを提供したりすることができます。

Ansible を用いる場合は、Playbook に記述した手順で、

- 既存のサーバーにアプリケーションを配布して実行する
- 新たに仮想マシンを生成や起動し、アプリケーションの配置や設定ファイルの読み込みなどを行う
- アプリケーションを内包しているコンテナからイメージを生成して実行する

などの手順でアプリケーションを配布、リリースできます。

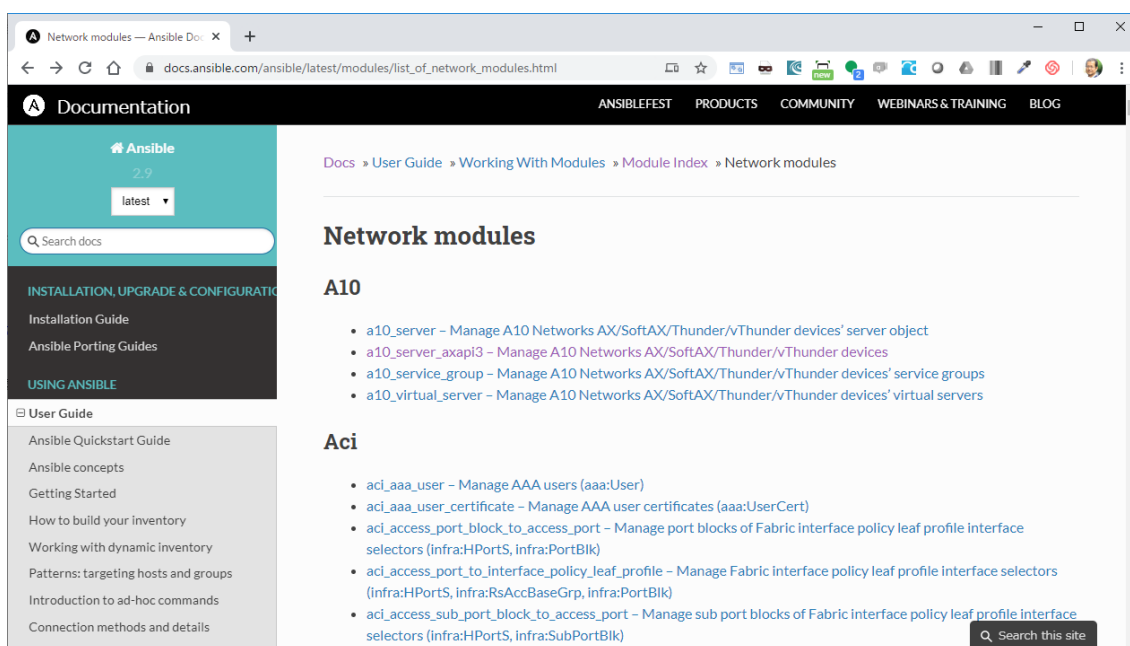
5.3 ネットワーク機器のステータス取得・設定変更

Ansible などの構成管理ツールはネットワーク機器の設定や、ステータス取得などにも対応しています。

各デバイスに対応した管理機能は、Ansible Network Module と呼ばれ、Ansible 最新版（執筆時点では 2.9）には数百ものネットワークモジュールが用意されています。

日本でもよく知られているデバイスには、以下のようなネットワークモジュールが提供されています。

- A10（ロードバランサ）
- Aruba（アクセスポイント）
- F5（ロードバランサ） BigIP などの製品
- Fortios（ファイアウォール） FortiGate
- Ios（Cisco 製品、各種スイッチやルーター）
- Junos（Juniper 製ネットワークデバイス）



ネットワークモジュール一覧（Ansible の公式ドキュメントより）

https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html

そして、Playbook に記述する設定には、

- config（設定）
- command（実行する設定コマンド）
- facts（収集された設定情報）

などがあります。

5.4 ネットワーク機器の情報を参照する例

Ansible の hosts ファイルでは一つのネットワークデバイスを定義します。

```
[router]
10.0.10.110
```

次は Playbook を作成します。以下の例では、ルーターのバージョン情報を取得し、テキストファイルに出力しています。ファイル名を「show_ver.yml」としています。

```
- hosts: router
  gather_facts: no
  tasks:
    - name: sh ver
      raw : "show ver"
      register: show_ver
    - name: sh ver output
      local_action: shell /bin/echo "{{ show_ver.stdout }}" >
/home/cisco/ansible/shver_{{ inventory_hostname }}.txt
```

ホスト情報と Playbook の作成を終えたら、ansible-playbook コマンドを実行します。

```
$ ansible-playbook -i hosts show-ver.yml --ask-pass
PLAY [router] *****
TASK: [sh ver] *****
ok: [10.0.10.110]
TASK: [sh ver output] *****
changed: [10.0.10.110 -> 127.0.0.1]

PLAY RECAP *****
10.0.10.110 : ok=2    changed=1    unreachable=0    failed=0
```

shver_10.0.10.110.txt のファイルの内容を確認すると、以下のようにルーターの OS のバージョンを確認できます。以上のようにしてサーバーだけでなく、ネットワーク機器の情報を取得したり、設定を実行することができます。

```
Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Experimental  
Version 15.4(20140730:011659) [lucylee-pi25-2 107]
```

2.05.1 仮想マシンの仕組みと KVM

1. 仮想化の概要と二つの方式（ホスト型とハイパーバイザー型）

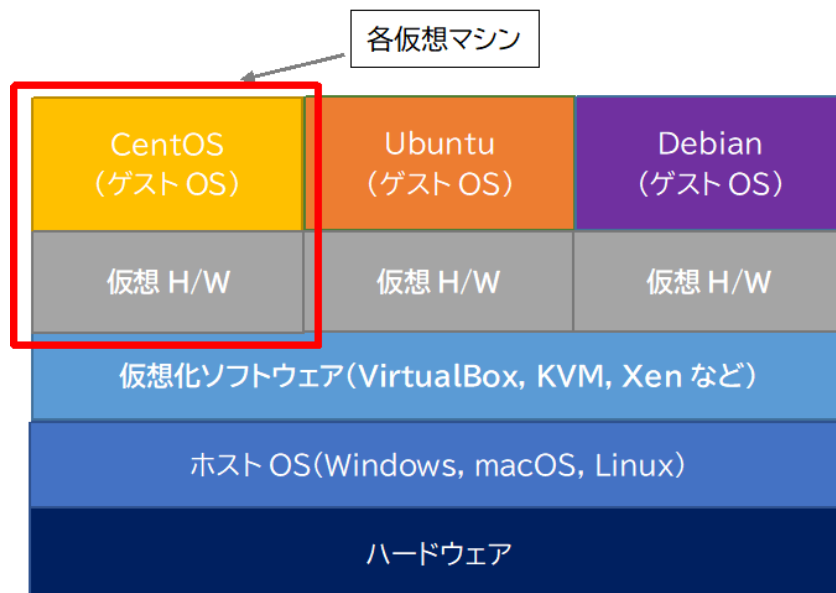
仮想化とは、コンピューターやその他種々の物理的なリソースを抽象化し、ソフトウェア化する技術のことです。コンピューターの場合、単一のハードウェア上で、複数の、ソフトウェア化されたコンピューターを動作させます。この、ソフトウェア化されたコンピューターのことを「仮想マシン (Virtual Machine)」と呼びます。

コンピューターに対して仮想化技術を使用すると、一つのハードウェア上で複数の異なるオペレーティングシステム（以下、OS）を動作させ、物理サーバー台数の節約をしたり、用途ごとのサーバーを高速に構築して提供したり、柔軟に構成を変更してスケールアップやスケールアウトに対応できるなどの利点があります。また、近年普及が拡大しているパブリッククラウドを実現する基盤技術としても使用されています。

コンピューターに対する仮想化方式は、ホスト型とハイパーバイザー型があります。

◎ ホスト型

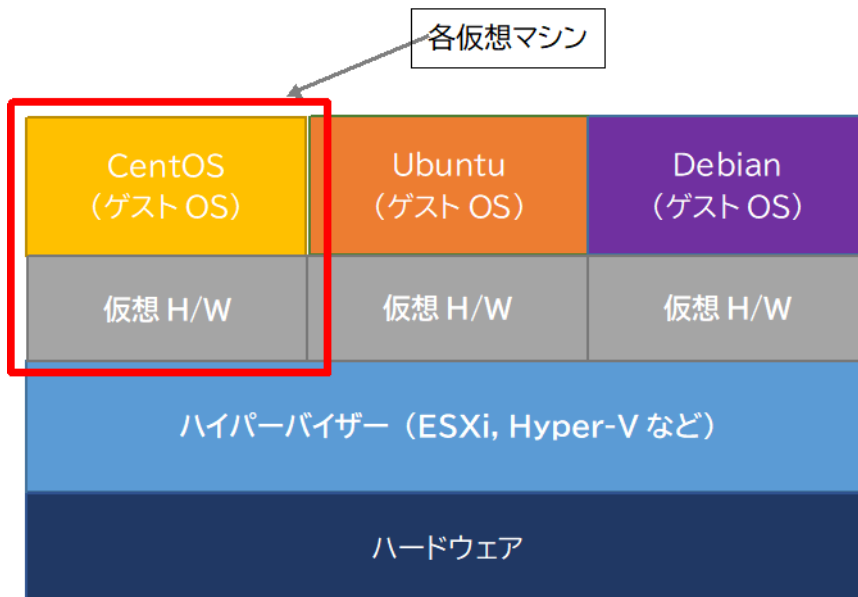
ハードウェアにインストールした OS（「ホスト OS」と呼びます）上に仮想化ソフトウェアをインストールし、その上で（複数の）仮想マシンをインストールして動作させる方式です。ホスト型の仮想化ツールでは、VirtualBox や VMware Player などがよく知られています。



ホスト型の構成イメージ

◎ ハイパーバイザー型

コンピュータ上にハイパーバイザーと呼ばれる仮想化管理ツールを介して、直接 OS をインストールして動作させる方式です。直接ハードディスク上にインストールするため、ホスト型に比べて高速に動作する、ホスト OS の制約のためにゲスト OS を調整する負担がない、などが利点です。ハイパーバイザー型の仮想化ツールとしては VMware ESXi や Hyper-V, KVM などがよく知られています。



ハイパーバイザー型の構成イメージ

2. KVM

KVM は、Kernel-based Virtual Machine の略で、Linux カーネルの拡張モジュールとして動作します。

CPU ごとにモジュールが開発されていて

- kvm-intel (Intel 社製 CPU 用)
- kvm-amd (AMD 社製 CPU 用)

の二種類のカーネルモジュールが提供されています。

CPU が仮想化支援機能に対応しているかどうかは、`/proc/cpuinfo` や `lscpu` コマンドで確認します。

```
$ ls /proc/cpuinfo
```

そして、

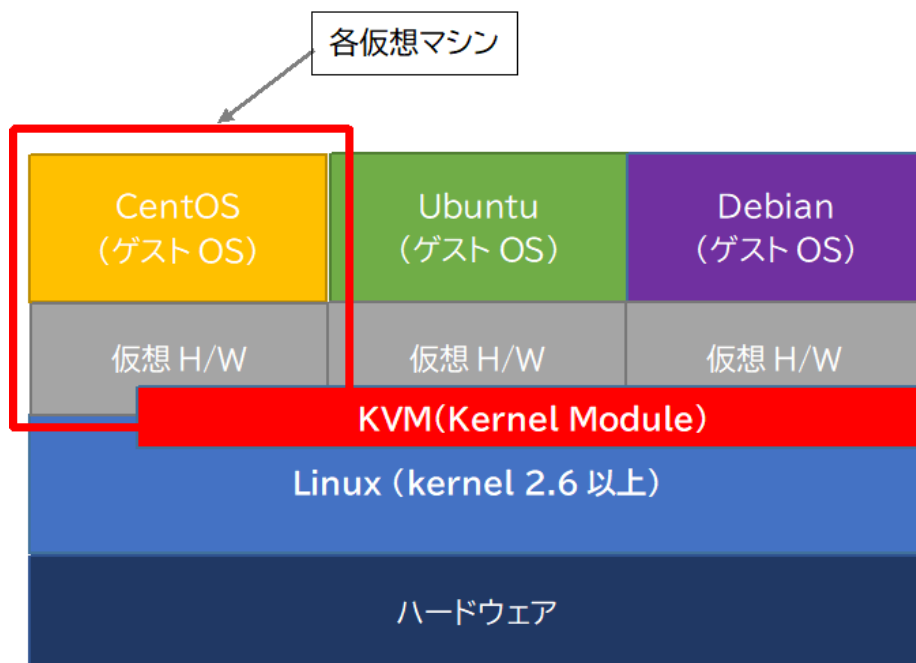
Intel 製 CPU の場合には「`vmx`」

AMD 製 CPU の場合には「`svm`」

があることを確認します。

もし含まれていない場合は仮想化ツールを実行することができません。Intel 製なら Core i3 以降の CPU なら仮想化機能 (VT-x) に対応しています。ただし、出荷時の設定で無効化されている場合があります。その場合は、電源投入直後に BIOS や UEFI の設定メニューを起動して、仮想化機能を有効化 (Enable) にする必要があります。

KVM モジュールは Linux カーネル上で動作するため、上記の分類のハイパーバイザー型のように動作します。そのため、ゲスト OS 側でホスト OS の制限を受けにくく、Linux の拡張機能などを使えます。



KVM の構成イメージ

◎ 完全仮想化と準仮想化

仮想化を実行する上では、以下の二つの仮想化方式があります。

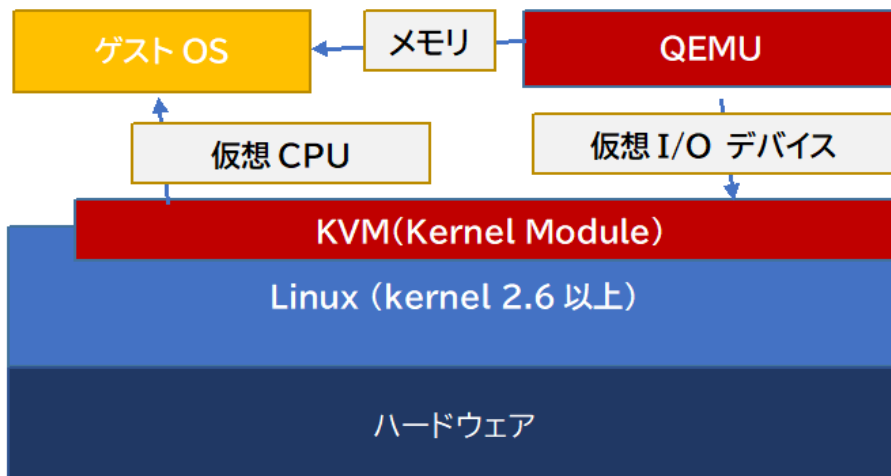
- BIOS レベルからハードウェア全体を仮想化し、ディスク I/Oなどをすべてエミュレートして、OS に手を入れないで仮想マシンを構成する「完全仮想化」
- ゲスト OS にハイパーバイザーとの連携の仕組みを取り入れる「準仮想化」

準仮想化は完全仮想化よりも高パフォーマンスを期待できますが、ハイパーバイザーへの依存性が高まるため、ハイパーバイザー間の移植性は低くなります。

KVM では virtio という準仮想化ドライバパッケージを用いることで、より高速なディスク I/O や高スループット、CPU 処理のオーバーヘッドの減少を実現します。

3. KVM を高速動作させるための QEMU

QEMU は、ゲスト OS からの入出力 I/O(ディスク、ネットワーク)を仮想化する機能を提供するエミュレータです。KVM がメモリや仮想 CPU を、QEMU がディスクやネットワークインターフェースなどの仮想ハードウェアを提供します。処理をそれぞれ分担することでパフォーマンスの向上を図っています。



KVM と QEMU の動作イメージ

4. KVM の導入と設定

KVM の導入・仮想マシンの追加は以下のようなステップで行います。

- (1) ホスト OS に KVM (カーネルモジュール) や QEMU をインストールする
- (2) ブリッジインターフェースを設定する
- (3) 仮想マシンを追加する

それでは順に実行していきましょう。ここでは CentOS 7 を使います。

4.1 必要なパッケージのインストール

以下のパッケージを yum コマンドでインストールします。

- yum のモジュール機能を使用して仮想環境パッケージ (virt)
- 仮想マシンの OS インストールツール (virt-install)
- ブラウザーインターフェースから仮想マシン (VM) を管理するツール (cockpit, cockpit-machines)

```
# yum -y module install virt
# yum -y install virt-install cockpit cockpit-machines
```

4.2 仮想化サポート機能の確認

```
# virt-host-validate
```

を実行して仮想化サポート機能の確認を行きましょう。

4.3 仮想化をサポートする libvirtd サービスの自動起動を有効化・起動

```
# systemctl enable --now libvirtd
```

続いてはブラウザーインターフェースを提供する cockpit ソケットの自動起動を有効化し、起動しましょう。

```
# systemctl enable --now cockpit.socket
```

4.4 インストール用 ISO イメージの用意

インストールしたい OS のイメージをダウンロードしておきます。今回の例では、

Ubuntu18.04LTS を使用します。

4.5 仮想マシンを追加する準備（ネットワークの設定）

KVM をインストールすると、デフォルトでは virbr0（仮想のブリッジインターフェース）と仮想ネットワーク（default）が設定されています。default は、ルーターとして機能し、DHCP サーバー機能を提供します。それぞれ設定を確認しておきましょう。

まずはインターフェースの確認をしてみましょう。

```
$ nmcli con | grep bridge
virbr0    dfeb02ec-554a-473e-b416-71334d164cbd  bridge    virbr0
```

次に仮想ネットワークの確認を行いましょう。

```
$ virsh net-list
名前                状態    自動起動  永続
-----
default            動作中  はい (yes)  はい (yes)
```

4.6 ブリッジインターフェースを追加

上記の NAT 接続ですと外部への接続はできますが、外部からの（インバウンド）接続ができません。そこでブリッジインターフェースを追加して、外部から接続できるようにしましょう。KVM には下記の Network Manager を使用する以外に、ブリッジ設定を行うブリッジユーティリティ（bridge-utils）も用意されています。

```
$ nmcli con add type bridge con-name br0 ifname br0
$ nmcli con mod br0 bridge.stp no
$ nmcli con mod br0 ipv4.method manual ipv4.addresses "192.168.1.5/24"
ipv4.gateway "192.168.1.1" ipv4.dns 192.168.1.1 （アドレスは各自の LAN に合わせる）
```

追加したら、もう一度インターフェースを確認してみましょう。

```
$ nmcli con | grep bridge
br0                bde3f90a-58ba-4d69-8992-5ba493f5d604  bridge    br0
virbr0            dfeb02ec-554a-473e-b416-71334d164cbd  bridge    virbr0
```


次に br0 の接続先として、イーサネットインターフェース (enp0s3 など、マシン上で確認しましょう) を追加します。

```
$ nmcli con add type bridge-slave ifname enp0s3 master br0
```

ブリッジインターフェースを有効化します。

```
$ nmcli con up br0
```

接続が正常にアクティベートされました (master waiting for slaves) (D-Bus アクティブパス: /org/freedesktop/NetworkManager/ActiveConnection/7)

ブリッジ接続を確認したら、enp0s3 の接続設定を削除しましょう。

```
$ nmcli con del enp0s3
```

2.05.2 仮想マシンの作成と管理

続いては、仮想マシンの追加や OS のインストールを実行していきます。以下のように `virt-install` コマンドで仮想マシンの追加が行えます。例えば、以下では、64bit CPU、2 コア、メモリ (RAM) 1GB、ディスク 20GB の Ubuntu 18 の仮想マシンを作成しています。

```
$ VM_NAME="ubuntu18"
$ virt-install ¥
--name ${VM_NAME} ¥
--hvm ¥
--arch x86_64 ¥
--os-type linux ¥
--os-variant OS 種類 ¥
--vcpus 2 ¥
--ram 1024 ¥
--disk
path=/var/lib/libvirt/images/${VM_NAME}.img,format=qcow2,size=20 ¥
--network bridge=br0 ¥
--graphics vnc,keymap=ja ¥
--noautoconsole ¥
--location ISO イメージファイルのパス
```

OS の種類に指定できるオプションは、`osinfo-query` コマンドで確認できます。まずはこのコマンドを実行して使用可能なバージョンを確認してから、ISO イメージをダウンロードしてもいいでしょう。

```
$ osinfo-query os
```

`virt-install` コマンドを実行したら、`virsh` コマンドで仮想マシンが表示されることを確認しましょう。

```
$ virsh list
```

続いては、仮想マシンとブリッジインターフェースの接続を確認しましょう。

```
$ virsh domiftest ubuntu18
```

1. 仮想マシンへの OS インストール

仮想マシンが追加できたら、OS のインストールを進めていきましょう。virt-install コマンドを実行すると、新しく追加された仮想マシン上で OS のインストーラが起動しています。

CentOS のブラウザインターフェースを起動して、仮想ディスプレイに接続し、インストーラを実行してみましょう。

<https://>ホストマシンの IP アドレス:9090

にアクセスして、root 権限を持つユーザーでログインします。

左側ペインのメニューで「仮想マシン」をクリックし、先頭の「>」をクリックしましょう。コンソールタイプを選択して仮想ディスプレイに接続します。GUI インターフェースを使用するためには「Graphics Console(VNC)」を選択しましょう。

あとはインストーラの指示にしたがって、OS のインストールを進めていきましょう。

以上の手順で CentOS 上に KVM (カーネルモジュール) を追加して、仮想マシンを追加し、異なる OS をホスト OS 上で実行できるようになります。

2. 代表的な virsh コマンドのオプション

KVM で生成した仮想マシンを操作するには、virsh コマンドを用います。主なコマンドには以下のようなものがあります。

仮想マシン (VM) の一覧を表示します。

```
$ virsh list all
```

VM を起動します。

```
$ virsh start <vm名>
```

VM を停止します。

```
$ virsh shutdown <vm 名>
```

VM を強制終了します。

```
$ virsh destroy <vm 名>
```

VM を再起動します。

```
$ virsh reboot <vm 名>
```

VM を削除します。

```
$ virsh start <vm 名>
```

VM の設定を変更します。(VM の停止が必須)

```
$ virsh edit <vm 名>
```

VM の CPU 情報を取得します。

```
$ virsh vcpuinfo <vm 名>
```

3. GUIで仮想マシンを管理するツール (virt-manager)

前節では virt-install コマンドを用いて仮想マシンの追加を行いました。デスクトップ環境を使用している場合には、GUI ベースの virt-manager を使用することも可能です。

```
# yum -y install virt-manager
```

(中略)

インストール済み:

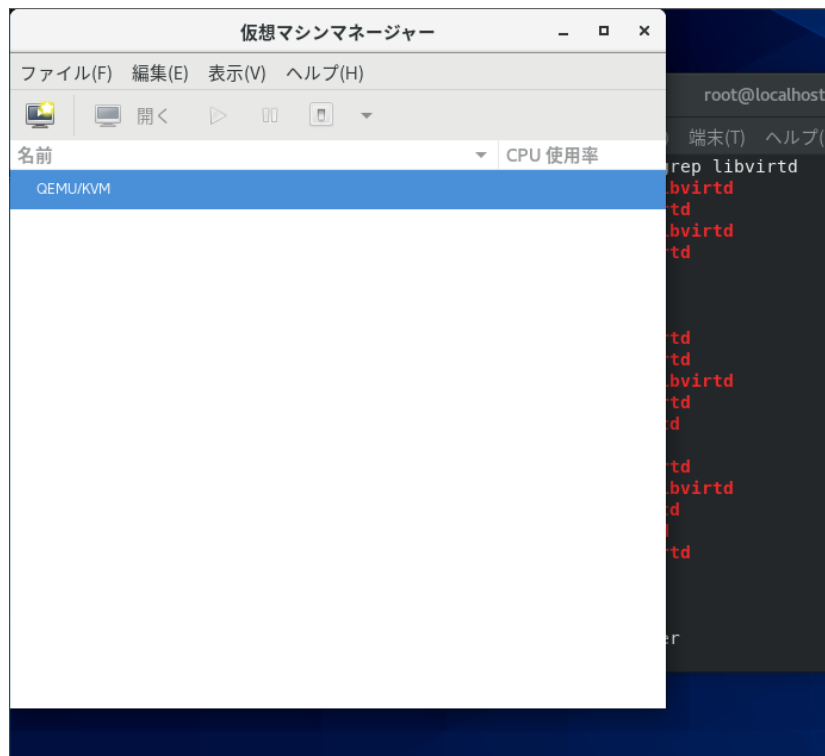
```
virt-manager-2.2.1-2.el8.noarch
```

完了しました!

インストールが完了したら、

- virt-manager コマンドを実行する
- ナビゲーションメニューから、「アプリケーション > システムツール > 仮想マシンマネージャー」をクリックする

のいずれかの方法で起動します。

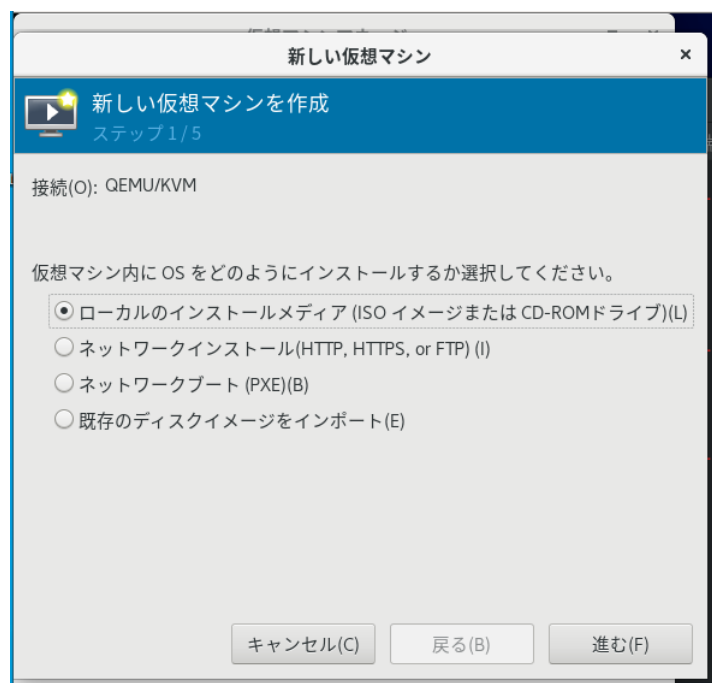


仮想マシンマネージャを起動したところ

この時、libvirtd が起動していないと接続エラーが出ますので、自動起動にしていな場合は以下のように libvirtd を起動します。

```
# systemctl start libvirtd
```

左上のディスプレイのアイコンをクリックすると仮想マシンを追加することができます。デフォルトでは、「ローカルのインストールメディア (ISO イメージまたは CD-ROM ドライブ)」が選択されています。あらかじめ仮想マシンにインストールしたい OS の ISO イメージを用意しておきましょう。



新しい仮想マシンの追加ウィザード



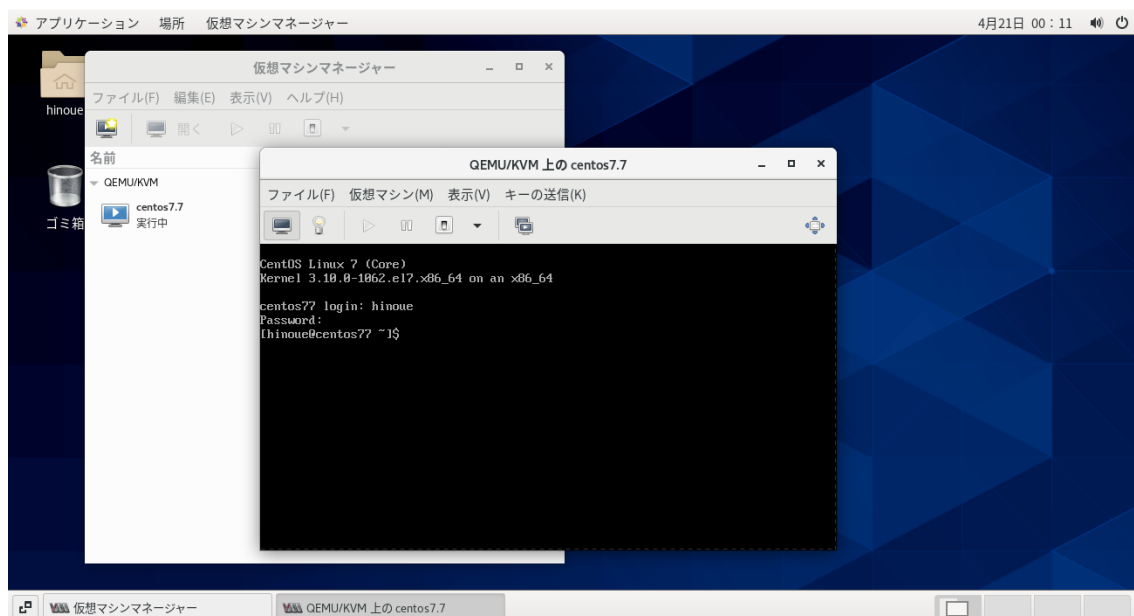
ISO イメージを選択して仮想マシンを作成するメニュー

このようにして、仮想マシンマネージャー (virt-manager) を使用すると、GUI ベースで仮想マシンの追加を実行できます。



仮想マシンマネージャーを使って CentOS7.7 のインストーラを実行しているところ

インストーラの実行が完了したら、OS を再起動します。今回は最小インストールパッケージを選択したので、下記のようにコマンドラインのログイン画面が表示されます。

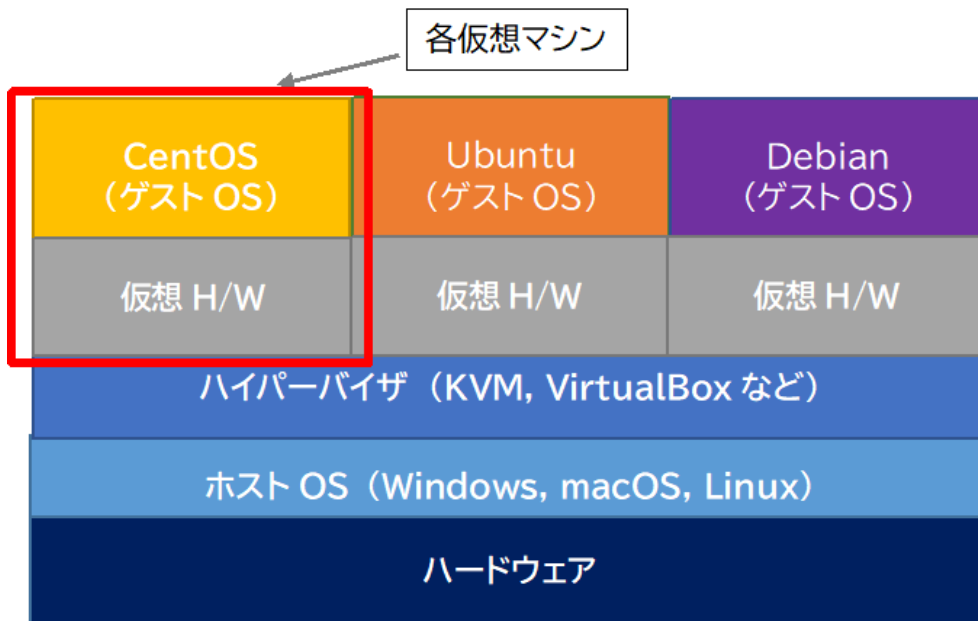


インストールを完了して KVM 上で CentOS 7 が動作している様子

2.06.1 コンテナの仕組み

1. 仮想マシンとコンテナ型仮想化の違い

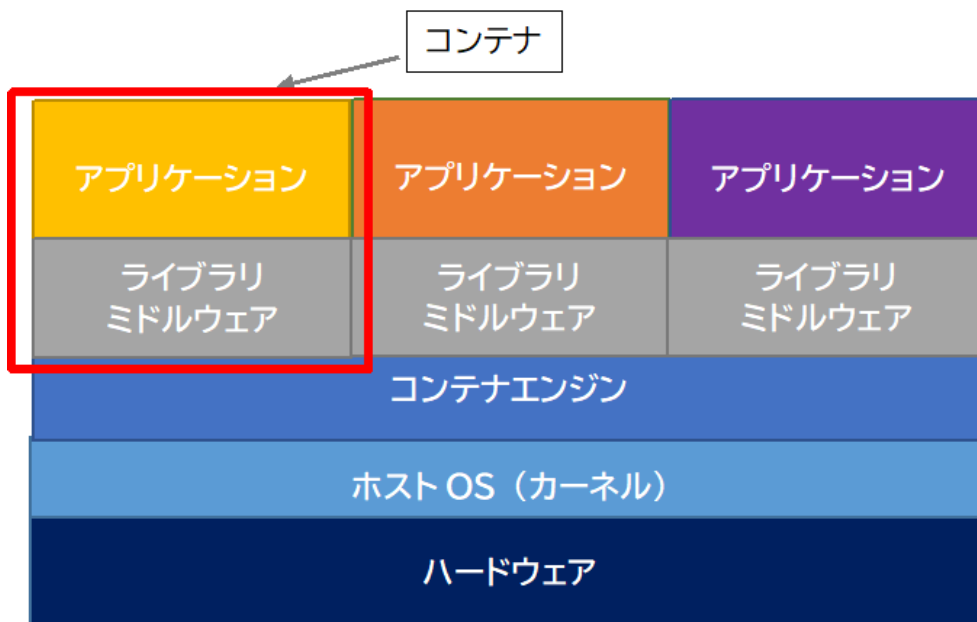
前節では物理マシンに直接 OS をインストールするのではなく、ホスト型・ハイパーバイザー型などの仮想化技術を利用した仮想マシンにインストールする方法について学びました。この結果、複数の OS を一つのハードウェア上で稼働させることができました。



仮想マシン型の仮想化イメージ

しかし、ハードウェアを丸ごと仮想化し、仮想マシンごとに OS をインストールするため、ホストマシンのメモリーやディスクの消費量が増え、仮想マシンの制御を行う分だけ CPU の負荷が高くなります。さらに、仮想マシンのバックアップに必要なディスク容量が増大したり、起動や複製に時間がかかったりするなどの問題があります。

そこで近年普及してきているのが、Docker などホスト OS 上のプロセスとして動作するコンテナ型仮想化技術です。コンテナ型仮想化技術を用いると、ウェブアプリケーションや各種サーバーなどの動作に必要なコンポーネントだけを切り出してコンテナというパッケージを構成し、より高速に起動や複製を実行できます。



コンテナ型仮想化イメージ

2. コンテナを実現する技術

コンテナ型仮想化は、

- コンテナエンジンを動作させる「ホスト OS」
- コンテナの動作を管理する「コンテナエンジン」
- サーバーのプロセスやリソースを区画化した「コンテナ」

から構成されます。

ホスト上に区画化された空間を実現することを「コンテナ型仮想化」、実現方法を「コンテナ技術」と呼んでいます。コンテナは、アプリケーションだけでなく、動作に必要なライブラリや設定ファイルなど、実行環境をまとめたものです。

コンテナ型仮想化の特徴としては、以下のような点が挙げられます。

- コンテナを起動するために、仮想マシンやゲスト OS を起動する必要がない。
- コンテナエンジンがホスト OS に常駐し、コンテナの操作・管理を行う。
- 各コンテナはコンテナエンジンが動作しているホスト OS のカーネルを共有している
- 各コンテナの機能を構成するプロセスをプロセス群として扱うことができ、コンテナ間で互いのプロセスは参照できない。

上記のようにコンテナ型仮想化は、仮想マシンやゲスト OS のリソースや負荷がないので、

- 仮想マシン型に比べるとコンテナのサイズは小さく、リソース消費量を抑えられる。
- ゲスト OS の起動停止が不要なので、起動が高速である
- 仮想マシンは数 GB のサイズになることが多いが、コンテナは数百 MB 以内の容量であることが多く、すべてのユーザーに同じ環境を提供することが容易になる。

などのメリットが得られます。

3. 名前空間 (namespace)

コンテナ型仮想化では、名前空間 (ネームスペース) という技術を使用して、コンテナとよばれるワークスペース間の分離 (isolation) を実現しています。コンテナエンジンはコンテナごとに名前空間の集合を生成します。

名前空間を扱う API (関数) には以下のようなものがあります。

- clone: 新たなプロセスを生成する
- setns: 呼び出したプロセスを既存の名前空間に参加させる
- unshare: 呼び出したプロセスを新しい名前空間に移動させる

コンテナ型仮想化で使用される名前空間には以下のようなものがあります。

- IPC 名前空間 (プロセス間通信のリソースを分離)
- マウント名前空間 (ファイルシステムツリーを分離)
- ネットワーク名前空間 (ネットワークインターフェースを分離)
- PID 名前空間 (プロセス ID 空間を分離)
- ユーザー名前空間 (UID/GID を分離)
- UTS 名前空間 (nodename, domainname のシステム識別子を分離)

各名前空間は、/proc/{pid}/ns/以下で確認できます。

```
$ ls -l /proc/$$/ns/
合計 0
lrwxrwxrwx. 1 root root 0  4月 28 11:46 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 root root 0  4月 28 11:46 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx. 1 root root 0  4月 28 11:46 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx. 1 root root 0  4月 28 11:46 net -> 'net:[4026531992]'
lrwxrwxrwx. 1 root root 0  4月 28 11:46 pid -> 'pid:[4026531836]'
lrwxrwxrwx. 1 root root 0  4月 28 11:46 pid_for_children ->
'pid:[4026531836]'
lrwxrwxrwx. 1 root root 0  4月 28 11:46 user -> 'user:[4026531837]'
lrwxrwxrwx. 1 root root 0  4月 28 11:46 uts -> 'uts:[4026531838]'
```

ホスト OS で namespace 一覧を表示した結果

一方、コンテナ上でも namespace を確認してみます。

```
root@fe4ef9552b2e:/# ls -l /proc/$$/ns/
total 0
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 ipc -> 'ipc:[4026532311]'
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 mnt -> 'mnt:[4026532309]'
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 net -> 'net:[4026532314]'
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 pid -> 'pid:[4026532312]'
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 pid_for_children ->
'pid:[4026532312]'
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 user -> 'user:[4026531837]'
lrwxrwxrwx. 1 root root 0 Apr 28 02:59 uts -> 'uts:[4026532310]'
```

コンテナ上で namespace を表示した結果

例えば、pid の結果を比較すると異なるノード番号が割りあてられています。

- ホスト： pid:[4026531836]
- コンテナ： pid:[4026532312]

これにより、コンテナはホスト OS のカーネルを使用していますが、コンテナ毎に異なる namespace 番号をつけることでプロセス群を分離して認識しています。

コンテナ上で ps コマンドを実行すると、ホスト OS で動作しているプロセスの情報は表示されません。これは、コンテナとホスト OS が異なる名前空間を使用しているためです。

```
# docker run -it ubuntu bash
root@fe4ef9552b2e:/# ps
  PID TTY          TIME CMD
    1 pts/0        00:00:00 bash
   10 pts/0        00:00:00 ps
```

ubuntu コンテナに接続してから ps コマンドを実行した結果

```
[root@centos7 ~]# ps
  PID TTY          TIME CMD
 3716 pts/1        00:00:00 su
 3724 pts/1        00:00:00 bash
 3746 pts/1        00:00:00 ps
```

ホスト OS 上で ps コマンドを実行した結果

4. cgroups (コントロールグループ)

コンテナ型仮想化では、cgroups を用いてリソース (CPU やメモリ) などの割り当てを行います。これに対して namespace (名前空間) では、名前空間ごとに識別用のノード番号を割り当てて、リソースの分離を実行していました。

cgroup は Linux カーネルの機能で、システムリソースの割り当て、優先度の変更、拒否、管理や監視レベルの制御などを行えます。

5. コンテナとイメージ

各コンテナはコンテナエンジン上で動作します。各コンテナのテンプレートはイメージと呼ばれ、コンテナからイメージを作ることができますし、既に構成済みのイメージファイルを公開リポジトリからダウンロードして使用することもできます。

イメージから生成されたコンテナのインスタンスにはそれぞれユニークな ID が振られ、起動や停止、接続や削除などができます。

2.06.2 Docker コンテナとコンテナイメージの管理

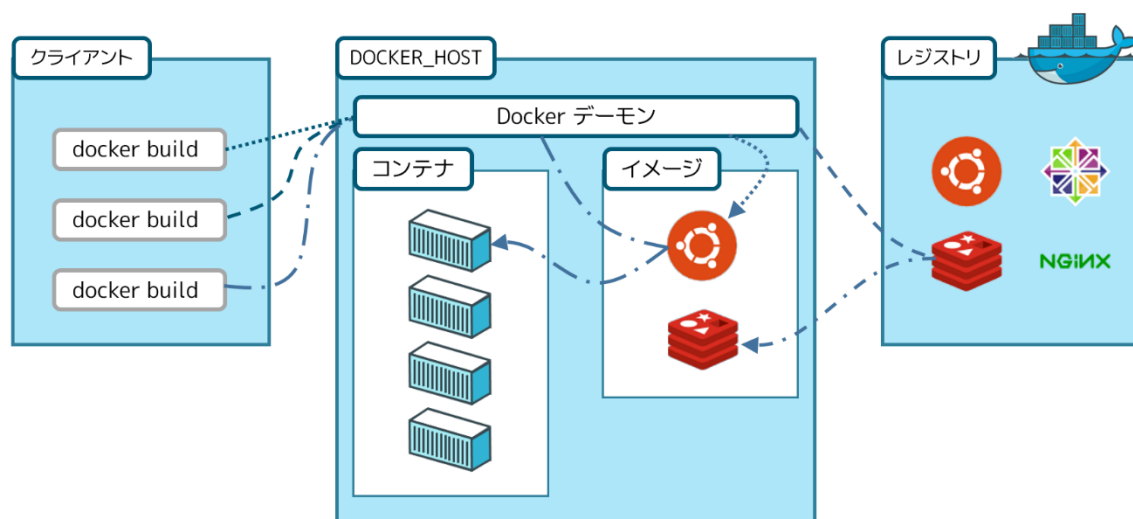
1. Docker の概要

コンテナ型仮想化で広く使用されているのが Docker です。Docker の各コンテナは、コンテナの動作プラットフォーム（Docker Engine など）上で動作します。

Docker は Go 言語で書かれていて、Apache 2.0 ライセンスの元でオープンソースソフトウェアとして公開されています。

さらに Kubernetes などコンテナオーケストレーション技術を用いると、複数コンテナを連動させて可用性の高いシステムを短時間に構築し稼働させることができるようになってきています。

ただし、Docker を使いこなすためにはシステム構成を考え、それに適した構成定義ファイル（Dockerfile）などを作成するスキルが必要で、ある程度の知識が必要です。



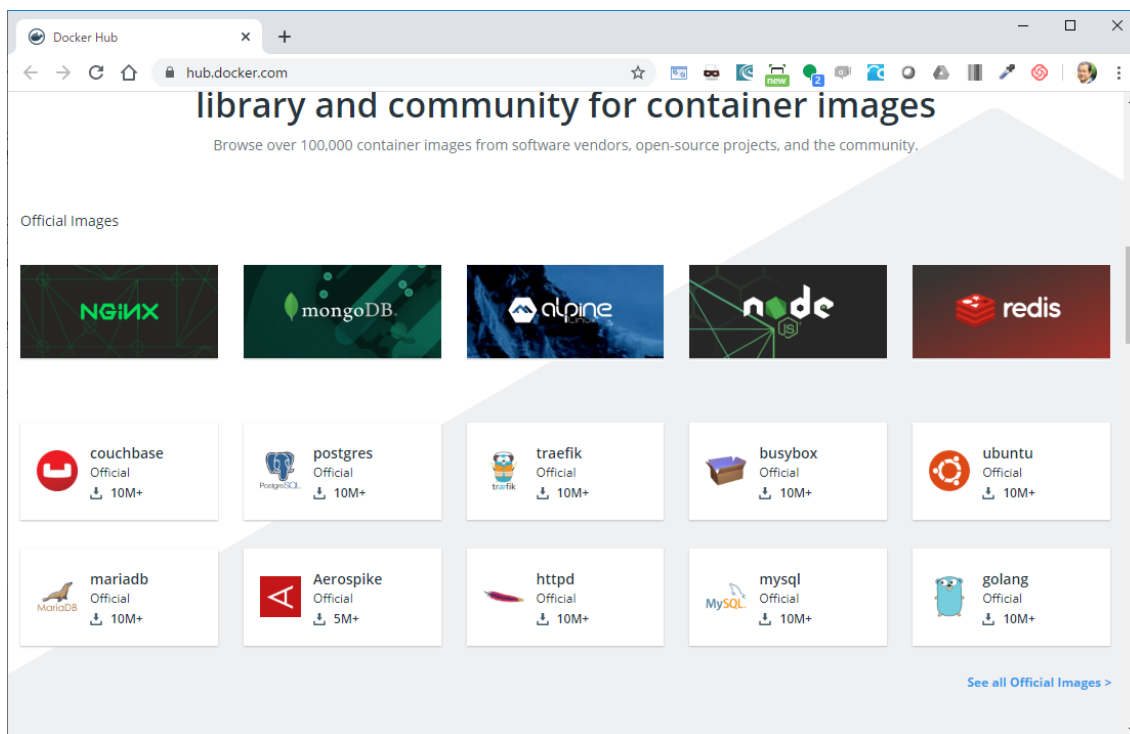
Docker の動作イメージ⁷

1.1 コンテナのファイルシステムとイメージの関係

各コンテナはイメージと呼ばれるコンテナのテンプレートファイルから生成されます。ユ

⁷ http://docs.docker.jp/_images/architecture.png

ユーザーは独自にコンテナに含めるコンポーネントを定義してイメージを作成することもできますし、Docker Hub (<https://hub.docker.com/>) などの公開リポジトリなどから、公開されている既成のイメージを取り込んでコンテナを生成することもできます。



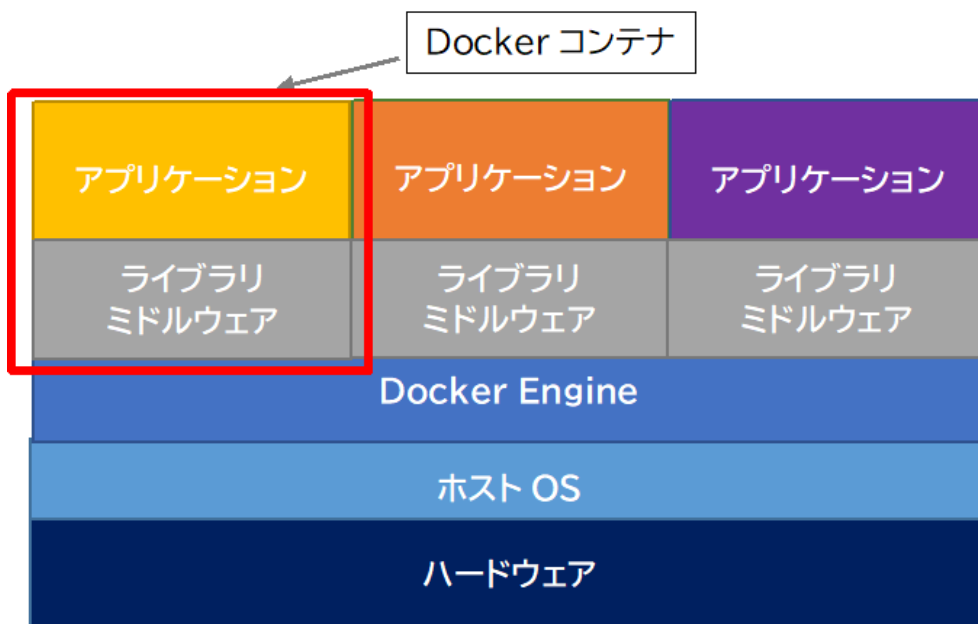
Docker Hub のサイト

Docker Hub では ubuntu などの OS、mysql, mariadb, httpd, nginx など各種アプリケーションやミドルウェアを含んでいるイメージが公開されています。

そして、`docker pull` コマンドを用いてイメージをダウンロードし、`docker create` コマンドでコンテナを生成、そして `docker run` コマンドでコンテナを実行できます。

1.2 Docker の動作イメージ

Docker の各コンテナは Docker Engine 上で動作し、ホストマシンのカーネルを使用します。そのため、ハイパーバイザーや仮想マシンの負荷がないため、より軽量で高速な動作が期待されます。



コンテナ型仮想化 (Docker) の動作イメージ

Docker は各種ミドルウェアライブラリのインストールや環境設定をコード化 (テキストファイルに定義を記述) して管理します。

構成ファイルを共有することで、

- 誰でも同じ仮想環境を構築できる
- 作成したシステム環境を配布しやすい (構成ファイルを提供すればよい)
- コンテナの追加・削除が容易にできる

などの利点が得られます。

1.3 構成自動化ツール利用の利点

構成自動化ツールのセクションでも扱いましたが、各種設定をコード化して管理する Infrastructure as Code を実現しています。コード化することにより、環境構築を自動化し、短時間に同じ環境を確実に再現できるため、システムの運用管理の負担軽減や、チーム開発時の環境構築の負荷軽減などに役立ちます。

開発段階からコンテナ上で開発を進め、本番環境に同じ環境を構築してデプロイ (本番環境に配置) することができるため、DevOps、開発 (Development) と運用 (Operation) の連携がスムーズになります。

開発メンバー間で設定ファイルを共有することで短時間に開発環境を生成し、バージョン

調整やインストールなどの負担を減らせます。

2. Docker の導入

Docker コンテナを動作させる Docker Engine には、有償の Enterprise Edition (EE) と無償の Community Edition (CE) の 2 系統があります。初めて Docker を導入する場合には Community 版を使用するといいいでしょう。

ここでは公式のインストール手順にしたがって、Docker CE をインストールしていきます。まずはホスト名の確認を行います。

hostname コマンドでホスト名が表示されることを確認しましょう。

```
$ hostname
centos7
```

続いては root に切り替えてから、Docker CE のインストールに必要なパッケージ、yum-utils、device-mapper-persistent-data、lvm2 をインストールします。

- yum-utils は yum のユーティリティーで yum-config-manager を含みます。
- device-mapper-persistent-data と lvm2 (論理ボリュームマネージャ) は、デバイスマッピングストレージドライバの動作に必要です。

```
# yum install -y yum-utils device-mapper-persistent-data lvm2
```

次に Docker Engine のインストールは、公式リポジトリを使用して行います。

まずは Docker 社の提供するリポジトリを追加します。

```
# yum-config-manager \
  --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo
```

リポジトリを追加したら、Docker Engine CE のインストールを実行します。

```
# yum install docker-ce docker-ce-cli containerd.io
```

Docker Engine CE がインストールされたら、起動してみましょう。

```
# systemctl start docker
```

動作確認のために hello-world イメージを動作させてみましょう。Docker Hub から hello-world (テスト用) イメージをダウンロードしてインスタンスを生成し、確認用のメッセージを出力します。run コマンドは、イメージの取得 (pull)、コンテナの生成 (create)、コンテナの起動 (start) を連続して実行します。

```
# docker run hello-world
```

ホスト OS の起動時に Docker を起動するには、systemd (CentOS や Ubuntu 最新安定版など) を使用しているのであれば以下のコマンドを実行します。

```
# systemctl enable docker
```

3. Docker によるネットワークの構築

Docker の Linux イメージを使用してコンテナを作成すると、通常は NAT (自動ポート変換)機能でホスト OS のポートを自動的に変換してコンテナに IP アドレスを割り当てます。

しかしこの方式だと、名前解決ができずコンテナ名で相互にアクセスできないので、ブリッジ接続に変更して同一ネットワークに異なる IP アドレスで接続するように設定を変更し、コンテナ間の通信を可能にする場合があります。

まずは ubuntu1,ubuntu2 というコンテナを起動してネットワーク設定を調べてみます。最初に ubuntu1 という名前のコンテナを起動します。d は起動時に接続しない (detouch) というオプションです。

```
$ docker run -itd --name ubuntu1 ubuntu /bin/bash
```

コンテナの起動に成功したら設定を inspect コマンドで確認します。

```
$ docker inspect ubuntu1
```

Networks という項目の内容をチェックしてみましょう。

```
[
{
    (中略)
    "Networks": {
        "bridge": {
            '''
            "Gateway": "172.17.0.1",
            "IPAddress": "172.17.0.2",
            '''
        }
    }
}
'''
]
```

すると 172.17.0.2 という IP アドレスが設定されています。

同様に ubuntu2 というコンテナも起動します。

```
$ docker run -itd -name ubuntu2 ubuntu /bin/bash
```

ubuntu2 も設定を inspect コマンドで確認します。

```
$ docker inspect ubuntu2
```

```
[
    (中略)
    "Networks": {
        "bridge": {
            '''
            "Gateway": "172.17.0.1",
            "IPAddress": "172.17.0.3",
            '''
        }
    }
]
```

すると、ubuntu2 には、172.17.0.3 という異なる IP アドレスが割り当てられています。

次に ubuntu1 から ubuntu2 に ping で疎通確認をしてみたいと思います。まずはコンテナを起動して、bash シェルを起動します。exec は指定したコマンドを実行するオプションです。

```
$ docker exec -it ubuntu1 /bin/bash
```

次に ping コマンドを使うために、iputils-ping をインストールします。

```
root@d3ce49845646:/# apt-get update && apt install iputils-ping
```

インストールが成功したら、ubuntu1 から ubuntu2 へ ping を実行してみましょう。

```
root@d3ce49845646:/# ping -c 3 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=1.46 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=1.70 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=1.63 ms
```

続いて、コンテナ名で ping を送信してみましょう。

```
root@d3ce49845646:/# ping -c 5 ubuntu2
ping unknown host ubuntu2
```

デフォルトのネットワーク構成ではコンテナ名でアクセスできないので、ユーザー自身でブリッジネットワークを定義して、コンテナ名でアクセスできるようにしましょう。

まずは、独自のネットワークを定義します。ここでは nw1 とします。

```
$ docker network create nw1
```

続いて、ubuntu1,2 のコンテナを nw1 に接続します。

```
$ docker network connect nw1 ubuntu1
$ docker network connect nw1 ubuntu2
```

もう一度 inspect コマンドを実行して nw1 の情報を確認します。

```
$ docker inspect ubuntu1
```

するとネットワークのセクションに、"nw1"が表示されます。

続いては、ubuntu1 に接続して ping を実行していきます。まずは、docker exec コマンドで ubuntu1 コンテナに接続します。

```
$ docker exec -it ubuntu1 /bin/bash
```

コンテナに接続してコマンドプロンプトが切り替わったら、コンテナ名で ping を実行してみましょう。

```
root@d3ce49845646:/# ping -c 3 ubuntu2
PING ubuntu2(172.19.0.3) 56(84) bytes of data.
64 bytes from 172.19.0.3: icmp_seq=1 ttl=64 time=1.46 ms
64 bytes from 172.19.0.3: icmp_seq=2 ttl=64 time=1.70 ms
64 bytes from 172.19.0.3: icmp_seq=3 ttl=64 time=1.63 ms
```

これで、コンテナ名で通信ができるようになりました。

このようにして、独自に定義したブリッジネットワークを使用すると、同じホストにつながるコンテナ間で、シンプルなネットワークを構築することができます。

さらに異なるホスト上で動作するコンテナ間で通信を行うためには、各ホスト OS にブリッジインターフェースを追加し、異なるホスト上で動作するコンテナを同一の L2 ブリッジネットワークに接続して、通信を可能にするフラットなネットワークを構成することもあります。

4. Docker コンテナの管理コマンド

Docker のインストール直後に、`docker run` コマンドを実行して、`hello-world` イメージをダウンロードしてコンテナを生成、動作させました。

Docker にはこれ以外にもコンテナの操作をするコマンドが用意されています。

- `docker ps/stats` コマンドを使うと、稼働中のコンテナ一覧や詳細を表示できます。
- `docker run/create/restart` コマンドを使うと、コンテナの実行、イメージからの作成、再起動ができます。
- `docker pause/unpause` コマンドを使うと、実行中の Docker コンテナのプロセスの一時中断、および再開を実行できます。
- `docker stop/kill` コマンドを使うと、実行中のコンテナを停止したり、強制終了したりできます。
- `docker rm` を使うと、既存のコンテナを削除できます。

4.1 コンテナに接続してプロセスを実行する

`docker attach` や `exec` コマンドを使うと、実行中のコンテナに接続して、コマンドを実行することができます。

という書式で使用します。

```
$ docker exec <コンテナ名> <コマンド名>
```

例えば、CentOS が動作しているコンテナに接続して bash (シェル) を起動してみましょう。

```
$ docker exec -it ubuntu /bin/bash
```

- -i オプションは標準出力を開き続ける
- -t オプションは疑似ターミナルを割り当てる

というオプションで、この二つを指定すると、ホスト OS からコンテナ内のシェルにコマンドを投入できるようになります。例えば、以下のようにホスト OS のコマンドプロンプトがコンテナのシェルのコマンドプロンプトに切り替わります。コンテナ上で exit コマンドを実行するとホスト OS のコマンドプロンプトに戻ります。

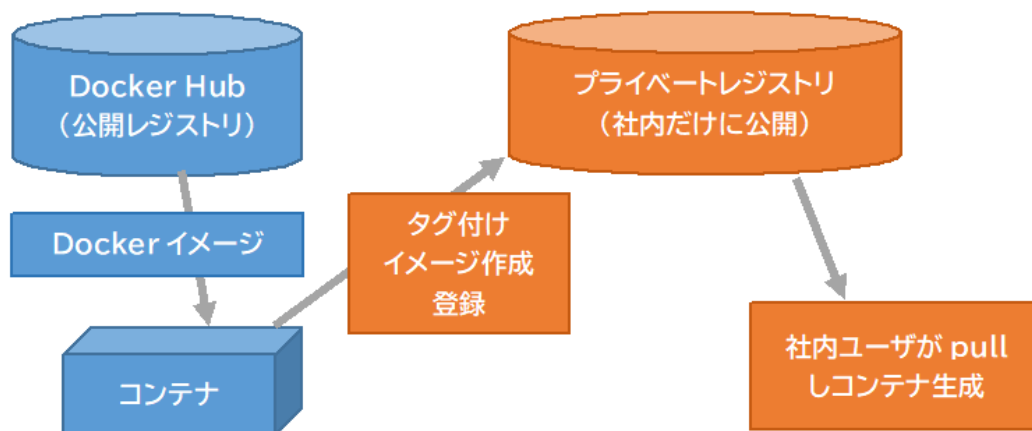
```
# [root@centos ~]# docker run -it ubuntu bash
root@d3ce49845646:/#
```

5. コンテナイメージの管理

Docker レジストリは、Docker イメージの管理（保管や提供）をする仕組みです。複数のリポジトリを持つことができ、それぞれ複数のイメージを格納しておけます。

リポジトリは、イメージを保存しておく領域のことで、ユーザーが名前をつけてイメージを識別したり、リポジトリ内のイメージを参照してコンテナを生成することもできます。

Docker Hub のように公開のレジストリも存在しますが、会社や組織内だけで公開せずにイメージを使いたい場合には、プライベートリポジトリを運用するのが適しています。



Docker レジストリのイメージ

- Docker レジストリの開始方法

Docker レジストリ（コンテナ）を開始するには、以下のように `docker run` コマンドを使用します。

```
$ docker run -d -p 5000:5000 --name registry registry:2
```

- `-d` は、デタッチオプション、コマンド実行時にコンテナに接続しないオプションです。
- `-p` は、ホストのポートをコンテナのポートに関連付けるオプションです。
- `name` オプションでレジストリ名を指定します。
- `registry:2` は取得するレジストリのイメージの名前を指定しています。

- `docker images` コマンドを使用すると、ローカルリポジトリ内のイメージ一覧を表示できます。

```
$ docker images
```


- `docker pull` コマンドを使用すると、Docker Hub（公開リポジトリ）からイメージを取得します。例えば、下記は `ubuntu` のイメージを取得します。

```
$ docker pull ubuntu
```

- `docker tag` コマンドを使うと、自分のレジストリ上にタグをつけて保存できます。例えば、先ほど取得した、ローカルにあるイメージにタグをつけるには、以下のようにします。

```
$ docker tag ubuntu localhost:5000/ubuntu
```

- 作成したイメージをローカルレジストリに登録するには、以下のように `push` コマンドで送信します。

```
$ docker push localhost:5000/ubuntu
```

登録したイメージをローカルのレジストリから取得するには `pull` コマンドを用います。

```
$ docker pull localhost:5000/ubuntu
```

- Docker レジストリを停止・削除するには以下のコマンドを用います。

```
$ docker stop registry && docker rm -v registry
```

➤ `-v` オプションで削除するレジストリ名を指定します。

- `docker import` コマンドを使用するとアーカイブ（圧縮された）ファイルからイメージを生成します。例えば、以下のようなフォーマットでコマンドを実行すると、`tgz` ファイルからイメージを生成します。ただし、タグ付けされていないイメージが生成されます。

```
$ docker import http://example.com/exampleimage.tgz
```

URL ではなく、ローカルファイルシステムから読み込むことも可能です。その場合は

URL の部分をファイルへの PATH にします。

より詳細なオプションは、Docker の公式ドキュメントを参照してください。

<http://docs.docker.jp/engine/reference/commandline/import.html>

- docker commit コマンドを使うと、既存のコンテナに名前をつけてイメージとして保存することができます。

```
$ docker commit default httpd
```

イメージの一覧を表示するには、docker images コマンドを使用します。

```
$ docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
httpd              
4f0aa49f1e67  48 seconds ago  197MB
```

上記の例では default が httpd という名前のイメージとして保存されます。また、既存のイメージにタグをつけて保存することも可能です。

例えば、centos のイメージに httpd をインストールし、centos:httpd という名前（前者がイメージ名で後者がタグ）でイメージを保存できます。

```
$ docker commit centos centos:httpd
```

● イメージの削除

イメージを削除するには、docker rmi (remove image) コマンドを使います。ただし、削除を実行するにはイメージを使用しているコンテナを停止しておく必要があります。

```
$ docker rmi httpd (イメージ名)
```

6. Dockerfile

ここまでは、docker pull/run コマンドで Docker Hub からイメージを取得したり、commit コマンドでコンテナからイメージを作成したりする手順を紹介しました。

しかし、この方法だとイメージを転用する際に不便な場合があります。例えば、イメージの

構成内容をドキュメント化したり、イメージから生成したコンテナに対して手作業で設定を変更したりしなくてはならないケースが出てきます。

そこでプログラムのビルドに使用される Makefile のように、コンテナの構成内容をドキュメントではなく、テキストファイルに定義しておいて、できるだけ手作業を減らして構成を自動化するための仕組みが Dockerfile です。

Dockerfile を作成しておくことで、テキストファイル内でイメージに含むアプリケーションやミドルウェアを定義して、`docker build` コマンドで Docker コンテナのイメージをコマンドライン (`docker build` コマンド) から自動生成できます。そのため、作業負担や操作ミス、作業時間などを減らせます。

● Dockerfile の書式

Dockerfile は、以下のフォーマットで記述します。

```
# コメント  
<命令> <引数>
```

主な命令には以下のようなものがあります。FROM だけでも実行できますが、主に RUN と ADD を用いてカスタマイズされたイメージを作成します。

命令	操作内容
FROM	元となる Docker イメージの指定
MAINTAINER	作成者の情報
RUN	コマンドの実行
ADD	ファイル/ディレクトリの追加
CMD	コンテナの実行コマンド 1
ENTRYPOINT	コンテナの実行コマンド 2
WORKDIR	作業ディレクトリの指定
ENV	環境変数の指定
USER	実行ユーザーの指定
EXPOSE	ポートのエクスポート
VOLUME	ボリュームのマウント

- カスタマイズイメージ作成の例

例えば、Ubuntu のイメージをベースにして、nginx をインストールしたイメージを生成してみましょう。

```
$ mkdir mynginx
$ cd mynginx
$ vi Dockerfile
```

として vi エディターを起動し、以下のような内容を記述します。

```
FROM ubuntu
MAINTAINER Foo Bar foo@example.jp
RUN apt-get install -y nginx
ADD index.html /usr/share/nginx/html/
```

また、別途 index.html ファイルを作成しておきましょう。例えば、メッセージを 1 行だけ含むファイルを生成します。

```
$ echo "Hello Nginx on Docker!" > index.html
```

Dockerfile と index.html が用意できたら、docker build コマンドを実行してイメージを生成してみます。

```
$ docker build -t <イメージ名>:<タグ名> <Dockerfile のディレクトリ>
```

という書式で Dockerfile からイメージを生成します。

```
$ docker build -t foo/mynginx:1.0 .
```

イメージを生成したら、images コマンドでイメージが追加されたことを確認しましょう。リポジトリ名とタグが Dockerfile と一致していることを確認しましょう。

```
$ docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
foo/mynginx     1.0         4f0aa49f1e67  48 seconds ago 198MB
centos          7           2d194b392dd1  2 weeks ago   195MB
```

イメージからコンテナを生成・起動し、動作確認をしてみます。

```
$ docker run -d -p 80:80 -name foo/mynginx:1.0 /usr/sbin/nginx -g
'daemon off;' -c /etc/nginx/nginx.conf
```

起動したら、コマンドラインから、コンテナの http ポートにアクセスして動作を確認してみましよう。

```
$ curl localhost
Hello Nginx on Docker!
```

というように index.html ファイルに記述した内容が表示されたら成功です。

もしもこのコンテナが不要な場合は、コンテナの停止、削除を実行しましょう。

```
$ docker stop mynginx
mynginx
$ docker rm mynginx
mynginx
```

2.13.1 高可用システムの実現方式

このセクションでは、高可用システムを実現する方式について学びます。

システムに障害が発生すると、提供しているサービスが停止してしまい、社会的信用を失ったり、損害賠償を請求されたりするリスクがあります。

例えば、2016年3月にはANA（全日本空輸）の航空券予約・販売・搭乗手続きのシステムに障害が発生し、日本各地の空港で搭乗手続きや発券処理ができなくなってしまった事例などがあります。このケースでは、国内線のデータを格納しているデータベースサーバーを接続するイーサネットスイッチの故障が原因でした。

システム障害発生によるリスクを最小限にするために、バックアップや冗長化（構成要素の多重化や負荷分散）などの対策をあらかじめ講じておくことが重要です。

1. 可用性に影響のある事象

システム障害が発生する原因には、以下のようなさまざまな要素があります。

- ハードウェアの老朽化・経年劣化、落下などの衝撃による物理障害（ハードウェア障害）
- システムの構成ミス、アプリケーションの設計ミスや不具合などの論理障害（ソフトウェア障害）
- ネットワークの障害
- オペレーションミスなどの人的要因
- 不正アクセス、コンピューターウイルス、DOS(悪意のある集中アクセス)などのセキュリティ攻撃
- 地震や水害、台風などの自然災害

また、メンテナンスのための停止も可用性に影響します。メンテナンスによる停止にも

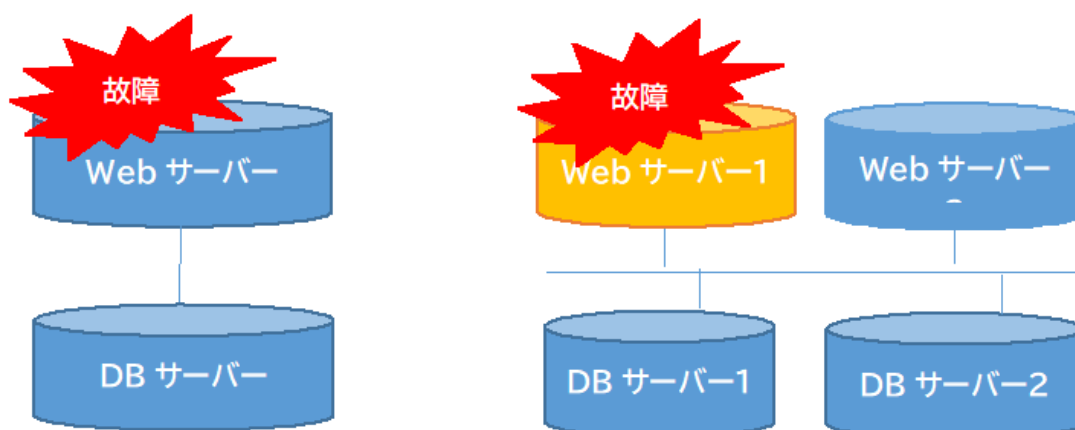
- 定期的なバックアップなどのための計画的なサービス停止
- 過負荷によるサーバーダウンやディスク破損などに対応するための緊急的なサービス停止。

があります。

できるだけシステム停止時間を短くし、可用性を向上させるためには、サーバーや構成機器、アプリケーションの冗長構成などの工夫をします。ただし、無限にコストをかけるわけにもいかないので、さまざまな評価指標を用いて、コストとリスク低減のバランスを考えてシステム構成を工夫する必要があります。

◎ SPoF, 回復性

システム停止を回避するには、その部分で障害が起きてしまうと、全システムが停止してしまう部分をなくす工夫が重要です。そして、システムの中で障害が発生してしまうとシステム全体が機能しなくなる要素は、「SPoF (Single Point of Failure : 単一障害点)」と呼ばれます。



1 台故障したらサービス停止

1 台故障しても、サービス継続

SPoF (単一障害点) のイメージ

また、システム障害が発生してしまった場合でもシステムを復旧させる時間を短縮するためにレプリケーション、つまり複数サーバー間でデータをリアルタイムにコピーしたり、同期したりする仕組みや複数のサーバーを同時に動かし、故障した機械から別の機械に切り替える、クラスタリングという手法が用いられることもあります。

許容される障害時間やコストを考慮し、クラスタリングやレプリケーションなどの工夫をして、システムの停止時間や復旧にかかる時間を限られた予算の中で以下に短縮できるか、ということが重要になります。

次の節ではシステムの稼働率や安定性を評価する指標について見ていきましょう。

2. 可用性の評価方法

可用性を評価するためには以下のような指標がよく用いられます。

- 稼働率と SLA
- RTO と RPO
- MTBF (Mean Time Between Failure)
- MTTR (Mean Time to Repair)

◎ 稼働率と SLA

システムの稼働率を表すには、Availability (可用性) がよく用いられます。

Availability は、

(合意したサービス時間 - システム停止時間) / 合意したサービス時間
をパーセント表示したものです。

クラウドサービスなどの契約には、サービス提供品質の指標として、Availability を用いて SLA (Service Level Agreement) を記述します。稼働率目標レベルを定義し、それを下回った場合に賠償請求ができるなどの項目を含むことがあります。

ISMS (情報セキュリティマネジメントシステム) では、情報の機密性・完全性・可用性を主要な 3 要素としています⁸。その中で可用性について定義をする際には、Availability(可用性)レベルを下表のようなクラスに分類し、目標レベルを定義します。

例えば、Availability レベルが 99.9%だと、ダウンタイムは 1 年間に 8 時間 45 分(24 時間 x 365 日 x 0.1%)となります。

レベル	Availability (%)	年間ダウンタイム(時間)
レベル 1	99.999	5 分
レベル 2	99.99	53 分
レベル 3	99.9	8.8 時間
レベル 4	99-99.5	44~87 時間

Availability レベルとダウンタイム

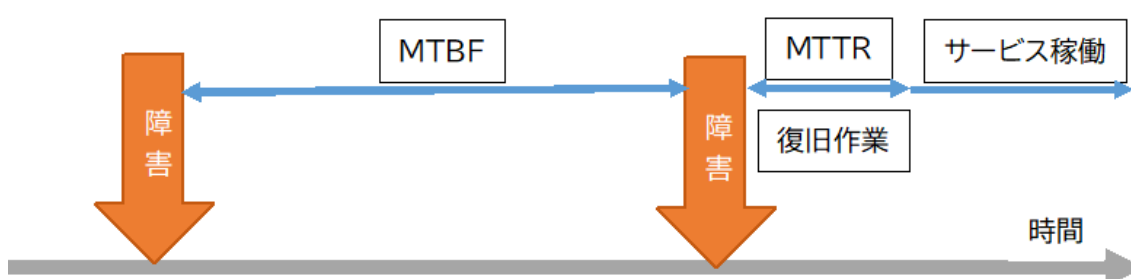
一般的な高可用性クラスタリングソフトウェアは、上記のうちレベル 2 の稼働率を実現するように設計されています。

◎ MTBF と MTTR

⁸ JIS Q 27000:2014

また、システムの平均的な連続稼働時間を MTBF (Mean Time Between Failure、平均故障間隔) と呼びます。Failure はシステム障害のことなので、システム障害が発生するまでの平均的な時間、つまり連続稼働時間の平均値になります。MTBF は長いほど、システムが安定して稼働していると考えられます。

さらにシステムが停止してから再稼働するまでの時間は MTTR (Mean Time To Repair、平均修復時間) と呼ばれます。MTTR が少ないほど、システム障害が発生してから復旧するまでの時間を短く抑えられる、ということの意味します。



MTBF と MTTR のイメージ

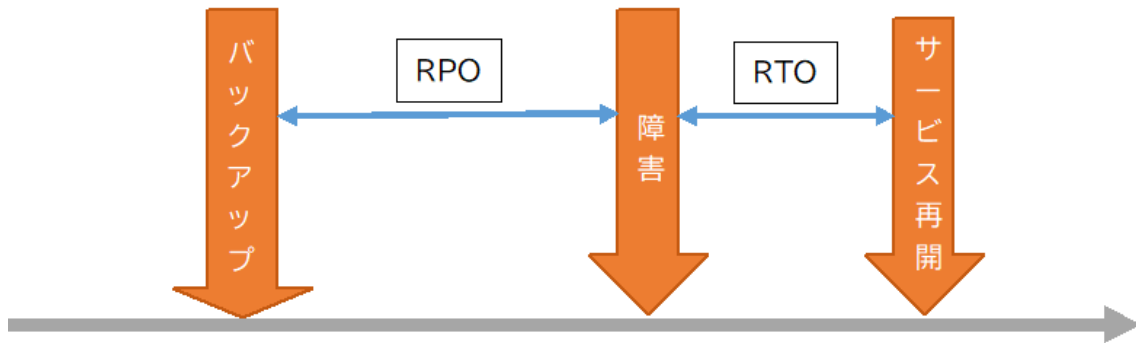
◎ RTO と RPO

そして、高可用性システムを設計する際に用いられる指標には、RTO (目標復旧時間: Recovery Time Objective) と RPO (目標復旧時点: Recovery Point Objective) があります。

RPO はデータを復旧するバックアップデータの古さの目標です。どこまでデータを復元できるかの目標値です。これによりバックアップ手法や保存期間、バックアップ処理のタイミングなどを策定します。

RTO はシステムを再稼働するまでの時間です。障害が発生してから、データ復旧、機器の修理・換装などをして、サービスを再稼働するまでの目標値です。

それぞれを短くするとダウンタイムを短縮することができますが、コストは数値を小さくするほど増大します。



RPO と RTO のイメージ

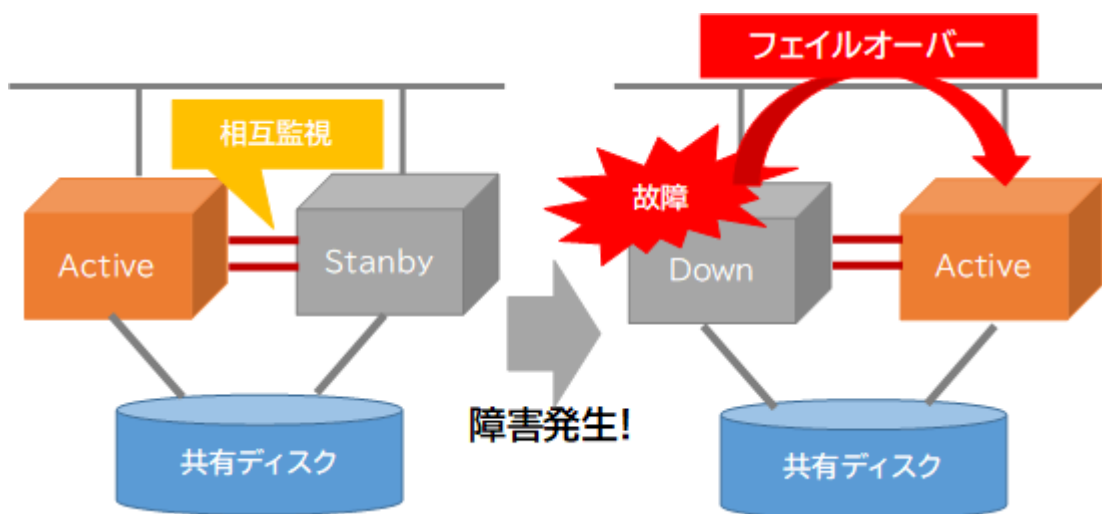
3. 高可用性 (HA: High Availability) を実現するシステム構成

可用性の高いシステムは、一般的に「高可用性のあるシステム」と呼ばれます。または略して HA (High Availability) 構成と表記することもあります。

◎ 冗長化 (HA クラスタ)

高可用性を実現するためには、サービスを提供しているマシンが停止してしまった際に、何らかの方法で、直ちにサービスを継続することが必要です。このためには、あるサービスを提供するシステムを多重化して、待機している予備マシンでサービスを引き継ぎます。引き継ぐ処理のことを「フェイルオーバー」と呼んでいます。こうすることで、ダウンタイムを極力少なくしてサービス提供が継続できるようになります。

こうした現用系(アクティブ)と待機系(スタンバイ)のノードから構成されるシステムは、HA クラスタ(高可用性クラスタ)と呼ばれています。

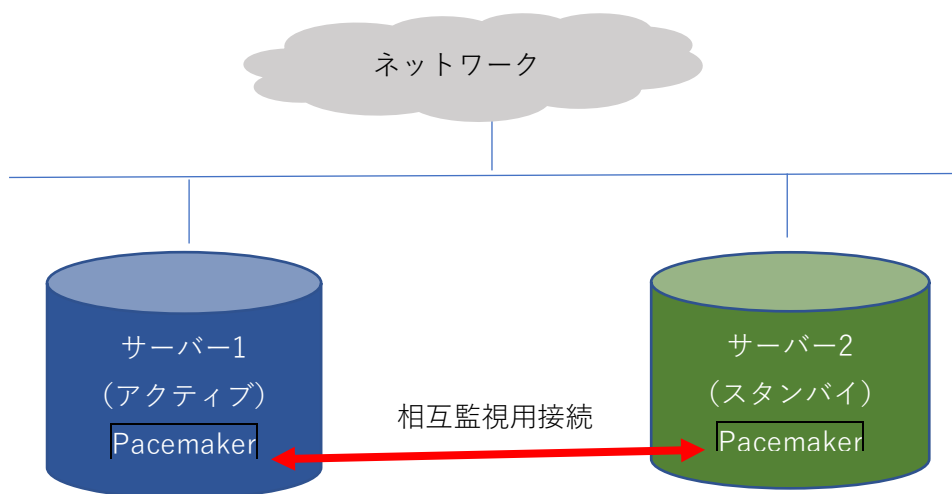


HA クラスタの動作イメージ

高可用性クラスタを構成するには、クラスタリングソフトウェアが使用されます。例えば、オープンソースで開発が進められているクラスタリングソフトウェアでは Pacemaker が有名です。Pacemaker はそれ以前から開発されてきた HeartBeat というソフトウェアのリソース管理機能だけを独立させたものです。

Pacemaker は単独では HA クラスタを稼働させることはできません。Corosync などのクラスタを制御するソフトウェアと組み合わせて動作させます。

Pacemaker のエージェントプロセスは、各マシン上で常時起動し、アクティブ系なノードとスタンバイ系のノードで指定した間隔で死活監視をしています。



また、サービスの監視や制御をするためのプロセスは RA (リソースエージェント) と呼ばれます。RA は、サービスの起動、停止、監視などを行います。

RA には、Web サーバー (Apache や Nginx)、データベースサーバー (PostgreSQL や MySQL)、ネットワークインターフェース、ファイルシステムなどの監視に対応したものがああります。

そして、もしも障害を検出したら、待機系のマシンに処理を引き継ぎます。フェイルオーバーに失敗すると、両方のノードがアクティブになってしまいます。これをスプリットブレインと呼びますが、両方のシステムから共有ディスクにアクセスしてしまい、データの不整合や破損のリスクが発生します。

スプリットブレインを防止するために STONITH (Shoot The Other Node In The Head の頭文字) という仕組みが用いられます。STONITH は両方がアクティブにならないようにフェイルオーバー時に障害が起きた方のサーバーの OS を強制的に再起動したり停止した

りする機能です。

◎ ロードバランシング（負荷分散）

例えば、大学や企業のポータルサイトなどで Web アクセスが集中することで Web サーバーのメモリを使い果たしてしまい、応答時間が長くなったり Web ページがホワイトアウトしたりしてしまう場合があります。

そこで、複数台の Web サーバー間で負荷を分散するために、

- DNS ラウンドロビンを用いてランダムにアクセスを振り分ける
- ロードバランサを用いてサーバー負荷の監視を行い、できるだけ負荷の少ないサーバーノードにアクセスを振り分ける

などの工夫がされます。

4. 物理的、地理的分散による可用性レベルの違い

同一拠点でハードウェアを冗長化するだけでは、自然災害などでデータセンターが機能しなくなった場合にシステム停止が発生してしまいます。

そこで、地理的に離れた複数拠点でロードバランシングしたり、データをレプリケーション（複製）したりしておく、などの工夫がサービス停止の回避に有効です。

前節の HA クラスターリングソフトウェアなどを用いると、複数拠点に配置したサーバー間でフェイルオーバーすることも可能となるでしょう。

2.13.2 キャパシティプランニングとスケーラビリティの確保

このセクションでは、キャパシティプランニングとスケーラビリティの確保に関する対応策と、リソース使用状況の把握について学びます。

1. キャパシティプランを作成するために把握すべきシステムリソース

システムのキャパシティを見積もるためには、主に以下のようなリソースの消費状況を確認する必要があります。

◎ 処理能力

CPU の使用率などを監視ソフトウェアなどで継続的に監視します。

大規模アクセスが発生するサービスを支えるサーバーでは、データベースへの同時アクセスが急上昇すると、CPU 使用率が 100% に達してしまうことがあります。こうなると処理が追いつかず、例えばウェブページが表示されない、スマホアプリにリクエストへのレスポンスが返らないというようなトラブルが発生します。そこで、定期的に CPU 使用率をモニターし、CPU の搭載数を増やす、サーバーの台数を増やして CPU 数を増やす、などの対策が必要となります。

◎ メモリ

オペレーティングシステムが消費するメモリ量と、アプリケーションが消費するメモリ量の合計が、メモリの消費量になります。

そして、この合計がハードウェアメモリの容量を超えると、メモリ容量を物理的なメモリ容量よりも大きく使えるようにする仮想メモリの仕組みによって、ディスクのスワップ領域へのアクセスが発生し、急激にシステムのパフォーマンスが低下します。

システムが快適な動作をするためにはできるだけスワップ領域へのアクセスを少なくすることが必要です。各メモリの消費量は `top` コマンドなどで確認が可能です。

Web サーバーへの集中アクセスが発生すると Web サーバーの子プロセスが多数起動され、メモリを使い果たしてしまうことがあります。この場合は、前述したように、メモリを増設したり CPU を高速化したりするスケールアップや、Web サーバーを複数台設置してロードバランシングするスケールアウトなどの対策が採られます。

◎ ディスク容量

メモリやCPUがいくら高速でもハードディスク領域を使い果たしてしまうと、システムが停止してしまいます。ハードディスクが満杯にならないように、消費状況をモニターし、溢れそうな気配があれば増設する準備を進めておく必要があります。ディスクの空き容量は、df コマンドで確認できます。

◎ ユーザー数

システムを利用するユーザー数と同時接続数を、データベース上のユーザーテーブルや Web サーバーのログから確認しておきましょう。使用している OS やアプリケーションで許諾されているユーザーライセンス数を超えると、利用ができなくなる場合があります。

2. リソースを増減させる方法と必要な対応

サービスが停止しないようにリソースを増減させる方法には、

- メモリを増設する、転送速度の高速なタイプに換装する（スケールアップ）
- サーバーの台数を増やす（スケールアウト）
- CPU を増設する。コア数の多いものに換装する（スケールアップ）
- ディスクを増設する（スケールアップ）

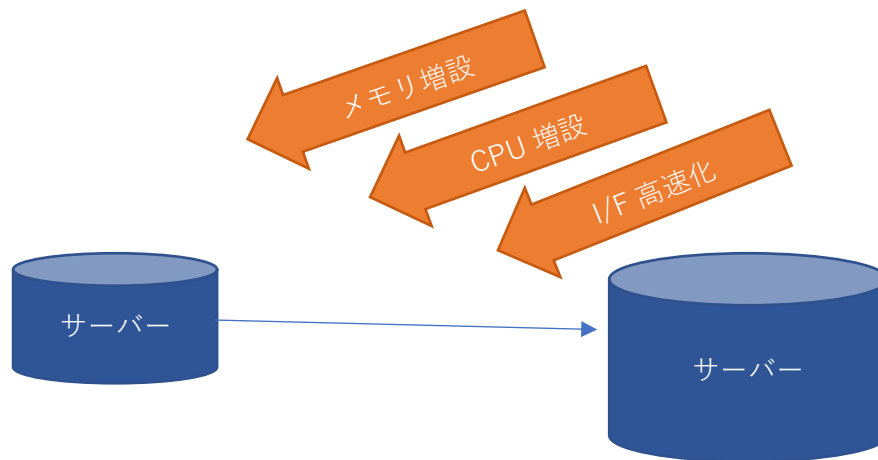
などの対応が考えられます。

3. スケールアップの方式

スケールアップは先に述べたように単一のサーバーのリソースを増強するものです。増強可能なものとしては、

- CPU の個数を増やす、コア数が多いものに換装する
- メモリの容量を増やす、転送速度が高速なものに換装する
- ディスクの増設、大容量のディスクへの換装、転送速度が速いディスクを選択する
- ネットワークインターフェースを高速なタイプに換装する

などが考えられます。



スケールアップのイメージ (各サーバーを增強)

4. スケールアウトの方式

スケールアウトは単一のサーバーではなく、サーバーを複数台構成に（水平方向に）拡張して CPU 負荷や、メモリやディスクなどの消費量を抑えて、サービス停止を防止するための工夫です。

例えば、大学の履修登録や企業のキャンペーンなどが発生すると、Web アクセスが集中することで Web サーバーのメモリを使い果たしてしまい、応答時間が長くなったり Web ページがホワイトアウトしたりしてしまう場合があります。このような時には、Web サーバーを多重化して、各 Web サーバーのメモリ消費量や CPU 消費量を低減させて、安定的に Web ページの配信を行えるようにします。



スケールアウトのイメージ (台数を増やして各ノードの負荷分散)

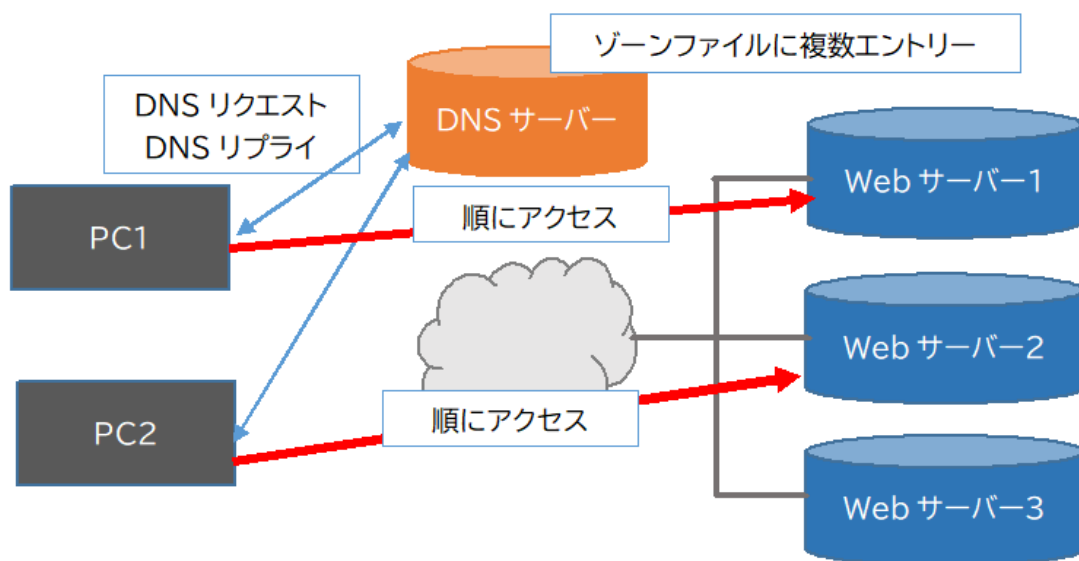
さらに外部から 1 台のサーバーにアクセスしているように見せるために

- DNS ラウンドロビンを用いてランダムにアクセスを振り分ける
- ロードバランサを用いてサーバー負荷の監視を行い、できるだけ負荷の少ないサーバーノードにアクセスを振り分ける

など、ロードバランシングと呼ばれる工夫をします。

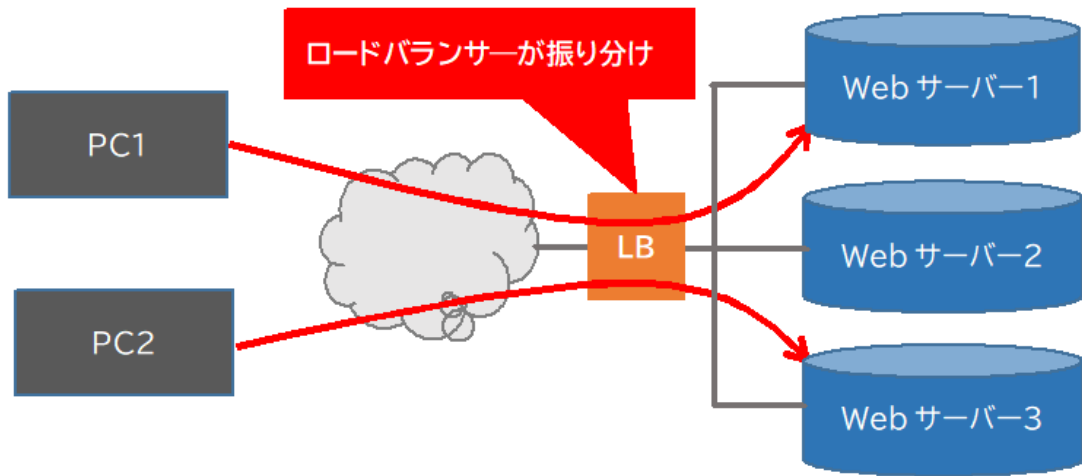
DNS ラウンドロビンを用いると特定のホスト名に送信されてきたリクエストを、複数の異なる IP アドレスが付与された Web サーバーにランダムに割り振って、負荷の分散を図ります。

例えば、DNS サーバーのゾーンファイル上で同じホスト名に対して複数の IP アドレスを割り当てます。すると同じ URL にアクセスしようとした場合に、クライアントのウェブブラウザには登録されている IP アドレスが順番に（ラウンドロビンに）返されます。それによって、アクセスを登録した IP アドレスの個数分のサーバーにアクセスを分散させることができます。



DNS ラウンドロビンによる負荷分散のイメージ
(DNS サーバーが異なる IP アドレスを返す)

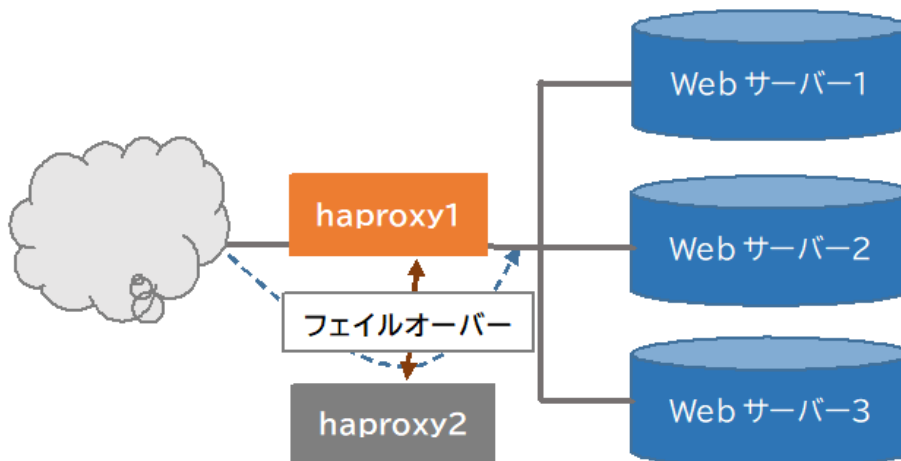
ロードバランサソフトウェアや、専用装置（アプライアンス）を用いると、各ノードの負荷を監視して、より負荷の低いマシンにリクエストを振り分けたり、同じセッションのリクエストを同一のノードで処理したりするように、クライアント（Web ブラウザーなど）からのリクエストや通信を振り分けます。



ロードバランサーの動作イメージ

ロードバランサーはネットワーク機器ベンダーが提供する専用アプライアンス、ソフトウェアベースのロードバランサー、クラウドサービスにおける仮想ネットワークデバイスなどの選択肢があります。

例えば、オープンソースのロードバランサーには、haproxy などがあります。また、SPoF (単一障害点) を減らすために、ロードバランサーを複数台並べて、死活監視を行い、ロードバランサー間でフェイルオーバーしたり、障害が発生したノードを停止したりすることもあります。haproxy を使用する場合には、サーバーを冗長化する keepalived でロードバランサーを切り替え、システム停止を回避することができます。



haproxy を冗長化したイメージ

◎ スケールアウトに対応できるアプリケーション構成

アプリケーションの構成でスケールアウトに対応できるようにするためには、

- Web サーバー、ミドルウェアを動作させるサーバー、データベースサーバーを分離する
まずは、HTTP アクセスの処理、アプリケーションの実行、データベースアクセスの負荷を分散するために、Web サーバー (Nginx や Apache)、ミドルウェア (PHP や Java EE サーバーなどのアプリケーションサーバー)、データベースサーバーのそれぞれを 1 台で構成します。
- 続いて、それぞれのノードの負荷が増大した場合
 - Web サーバーを複数台構成にしてロードバランシングする (先頭にロードバランサを配置する)
 - データベースサーバーを複数台構成にしてデータを複数台で保持するデータベースクラスタリングを行う

などの対策を考えます。

- アプリケーションをステートレスな構成にする。
 - セッションが複数サーバーで利用できるよう、共有領域に保持するようにアプリケーションを開発したり、設定ファイルで指定したりできるようにする。
 - ロードバランサによっては同一セッションを同じマシンに処理させるセッション親和性 (セッションアフィニティ) 機能を搭載しているデバイスもあります。
 - アプリケーションが用いるデータを共有ディスクに保持する。これにより、データの整合性が保持されます。
 - データベース設計を疎結合に対応できるように工夫する。リレーションを厳しくし過ぎないでデータベースを複数に分割できるように構成しておく。参照系と更新・挿入系のデータベースを分ける

など複数台で運用することを想定したアプリケーション構成にしておくことも有効な場合があります。

◎ 構成管理ツールや仮想マシンイメージを使ったスケールアウト

手動で仮想マシンを追加すると、Apache や Nginx などの Web サーバーソフトウェアをインストールして設定したり、MySQL などのデータベースサーバーソフトウェアをインストールして初期化する、などの初期設定作業が必要になります。こうした初期設定作業はプロビジョニングと呼ばれています。しかし、台数が増えてくると作業者の負担が大きくなり、設定ミスが発生したりするリスクが発生します。

そこで、構成管理ツールを使用してあらかじめサーバーに組み込むソフトウェアや設定などの情報をプロビジョニング設定ファイルに記述しておき、コマンドから読み込むことで一連のプロビジョニング作業を自動的に行うと、前述のリスクを低減させることができます。

構成管理ツールには、2.04.6 で解説したような Ansible などが使用されています。

2.13.3 クラウドサービス上のシステム構成

クラウドはインターネットに接続して利用することを前提とした各種サービスのことを指します。特徴としては、負荷の増減に応じて柔軟にリソースを追加したり、構成を変更しなおしたりできます。

ただし、クラウドサービスには、SaaS、PaaS、IaaS など、さまざまな利用形態があります。

◎ SaaS (Software as a Service⁹)

SaaS は、従来パッケージ製品として提供されていたソフトウェアをインターネット経由でサービスとして提供し、利用する形態のことです。Salesforce や G Suite などが広く使用されています。

◎ PaaS (Platform as a Service¹⁰)

PaaS は、ソフトウェアが動作するためのハードウェアや OS などの一式をインターネット上で提供する形態のことです。各サービスの仕様に沿ってソフトウェアを開発することで、OS やミドルウェアなどのインストールや設定をしなくても、アプリケーションの開発、提供ができるのが特徴です。Google App Engine や Heroku などが広く使用されています。

◎ IaaS (Infrastructure as a Service¹¹)

IaaS は、仮想サーバーやストレージ、ネットワークデバイスなどのインフラを、インターネット上のサービスとして提供する利用形態のことです。ユーザは必要に応じて、自由にシステムを構成するためのサーバーやデバイスのスペックや容量を選択できるのが特徴です。Google Compute Engine や Amazon Elastic Cloud(EC2)など、パブリッククラウドと呼ばれるサービスが IaaS に分類されます。

このセクションでは、IaaS を中心としたシステムを構成する要素について説明していきます。

1. クラウドストレージの種別

⁹ サース

¹⁰ パース

¹¹ アイアース・イアース

クラウドサーバーで使用できるストレージには、以下のようなものがあります。

◎ エフェメラルストレージ

インスタンスの起動時だけ使用可能なストレージです。AWS などではインスタンスストレージと呼ばれます。

仮想マシンのインスタンスを停止するとデータが消えて（蒸発する、揮発する、とも言います）しまうという特徴があります。

しかし、高速・広帯域（ディスク I/O が高速）で、追加料金がかからない（AWS EC2 の場合）ので、

- バッチ処理やバックアップ作業などの一時的な作業領域
- キャッシュ
- テンポラリな（一時的な）ログの保持
- データダンプを一時的に保持する

などの用途に使います。

◎ 永続化ストレージ

高速・広帯域な揮発性のインスタンスストレージに対して、インスタンスを停止、起動してもデータが消えない（蒸発しない、揮発しない）タイプのストレージは、永続化（パーシステントな）ストレージと呼ばれます。

物理的なハードディスクなどブロックデバイスのように、初期化してファイルシステムを構築し、仮想マシンにマウントすることができます。

同じネットワーク上の複数の仮想マシンにマウントしたり、逆に一つの仮想マシンに複数ボリュームをマウントしたりすることができます。

2. クラウドのネットワークの種別

クラウド上で仮想マシンや（仮想）ネットワークデバイスに IP アドレスを割り当てる場合には、固定 IP アドレスを割り当てる方法と、複数の仮想マシンのインスタンスで IP アドレスを共有する方法があります。

◎ 固定 IP アドレス

固定で IP アドレスを割り当てます。パブリック IP アドレスを設定すれば外部からアクセスできます。また、ファイアウォールの内側でプライベートな IP アドレスを割り当てることも可能です。

◎ フローティング IP アドレス

高可用性クラスターで冗長化したシステムを構成する場合などに使います。まず、アクティブなノードにある IP アドレスを割り当てます。もしもアクティブなノードで障害が発生したら、クラスター制御ソフトなどを使い、その IP アドレスをスタンバイしているノードに割り当て直してからアクティブにします。その際、元のアクティブノードの IP アドレスは解放します。

このようにクラスターを構成するノードで IP アドレスをシェアし、動的に割り当てを変更するタイプの IP アドレスを「フローティング IP アドレス」と呼びます。フローティング IP アドレスはクラウド環境だけでなく、ベアメタル環境でも使用することがあります。

3. クラウドのネットワークセキュリティ

◎ テナントネットワーク

多くのクラウドサービスではユーザーは他のネットワークから独立した仮想ネットワークを構築します。この独立したネットワークは「テナントネットワーク」と呼ばれます。また、仮想のエッジデバイス（外部と通信するノード）でルーティングを設定することで異なるテナントネットワーク間で通信をすることも可能です。

◎ ファイアウォール（セキュリティグループ）

セキュリティグループとは、クラウドサービスで設定できるファイアウォール機能のことです。セキュリティグループを構成すると、仮想マシンへのアクセスや外部からサーバープロセスへの通信などを制御することができます。

例えば、

- 特定の IP アドレスからの SSH 接続のリクエストを許可して、管理者権限ユーザーが sshd にアクセスし、ログイン処理を行えるようにする
- 任意のクライアントから Web サーバー（http d）への 80 番、443 番、指定したポート番号を用いたアクセスを許可する

などの設定が可能です。

パブリッククラウドサービスは、オンプレミスのサーバーや社内システムとは違い、常に第三者からのポートスキャンなどの脅威にさらされます。

不用意にインバウンド（外から内）のアクセスを許可していると、不用意にログインされてしまったり、連携している社内システムに侵入されたりするリスクが発生します。そこで、必要以上にアクセスを許可しないなど、セキュリティグループの設定には細心の注意が必要です。

◎ 認証の強化

先に説明したようにパブリッククラウド環境では第三者がポートスキャンをして、接続を試みるリスクがあります。そこで、オンプレミスでは ID・パスワード認証をしていますが、クラウドサーバーについては公開鍵認証方式を用いる方がセキュリティ強度を高めることができます。ユーザーがキーを作成し、登録する手間はかかりますが、サイバー攻撃からシステムを防御するためには非常に有効な手段となりますので、できる限り公開鍵認証方式が使えるように設定をしましょう。

◎ マシンイメージのセキュリティ

クラウドサービスでは、既存のサーバーイメージを元に仮想マシンのインスタンスを生成して、手動や構成管理ツールを用いて、短時間に新しいサーバーの増設ができる利便性があります。

しかし、マスターとなるサーバーイメージにセキュリティホールが存在した場合には、そのイメージから新規に追加したインスタンスにも脆弱性が引き継がれてしまいます。

そこで、既存イメージをそのまま使いまわすのではなく、セキュリティパッチなどを適用して安全なイメージを用いて仮想マシンインスタンスの追加ができるように継続的にメンテナンスする必要があります。

4. クラウドを支える主要な技術やサービス

以下では、クラウドを支える主要な技術やサービスについて説明します。

◎ オブジェクトストレージ

データの保存に特化したサービスはオブジェクトストレージと呼ばれています。

例えば、Amazon S3 では「バケット」という単位でデータオブジェクトを格納する領域を設定し、各種データの保存や取り出しを行えます。例えば、OS のイメージ、バックアップデータから、個別のテキストファイル、画像データなど、さまざまなサイズ、フォーマットのデータを格納することができます。

各データには URL でアクセスすることが可能で、

- バケット名が aws-s3-bucket1
- データのキーが /photos/image1.jpg

である場合には、

<https://aws-s3-bucket1.s3.amazonaws.com/photos/image1.jpg>

のように URL を用いて参照し、アプリケーションから呼び出して使用することが可能となります。

◎ メッセージングシステム（キュー）

メッセージングシステムは、異なるシステムやアプリケーション間で非同期に処理を連携する際に使われる仕組みです。

各アプリケーションはメッセージングプロバイダと呼ばれるサービスにメッセージを送信します。メッセージングプロバイダは、コンシューマーと呼ばれるメッセージを受信するシステムやアプリケーションに対して、受け取ったメッセージを送信します。

メッセージにはアプリケーション固有のデータは含めません。例えば、在庫商品の情報、人事情報などアプリケーションに依存しないデータがやりとりされます。

メッセージングシステムを使用することで、異なるアプリケーション間での処理を非同期に連携することが可能になります。

◎ オートスケーラー

パブリッククラウドサービスでは、システムの負荷の増減状況に基づいて、自動的に仮想マシンインスタンスの増設や削除を行う「自動（オート）スケーリング」機能が提供されています。

オートスケールを行うインスタンス群は、事前に設定したスケーリングポリシー（方針）に基づいて、自動的にインスタンスの追加・削除を実行します。

負荷が大きくなった場合にインスタンスを追加してスケールアウト（台数を増やす）処理は「アップスケーリング」、負荷が少なくなった場合にインスタンスを削除する処理は「ダウンスケーリング」と呼ばれています。

オートスケーリングのポリシーには、以下のようなものがあります。

- 平均 CPU 使用率
例えば、インスタンスあたりの CPU 平均使用率が 70%以下になるようにインスタンス数を増減させて自動調整します。
- 1秒あたりの HTTP/HTTPS リクエスト数
例えば、1秒あたり 1000アクセスを処理できるようにインスタンス数の調整を行います。
- グループ全体での指標
各インスタンスではなく、インスタンスグループ全体でのリソース消費や CPU 負荷、処理能力などの指標を設定します。

2.13.4 典型的なシステムアーキテクチャ

このセクションでは高可用性やスケーラビリティを確保するためのシステム構成パターンについて学びます。

1. LAPP, LAMP 構成

オープンソースの Apache/Nginx Web サーバーと PHP モジュール、PostgreSQL または MySQL などのリレーショナルデータベースから、構成される Web アプリケーションは世界中で広く使用されています。

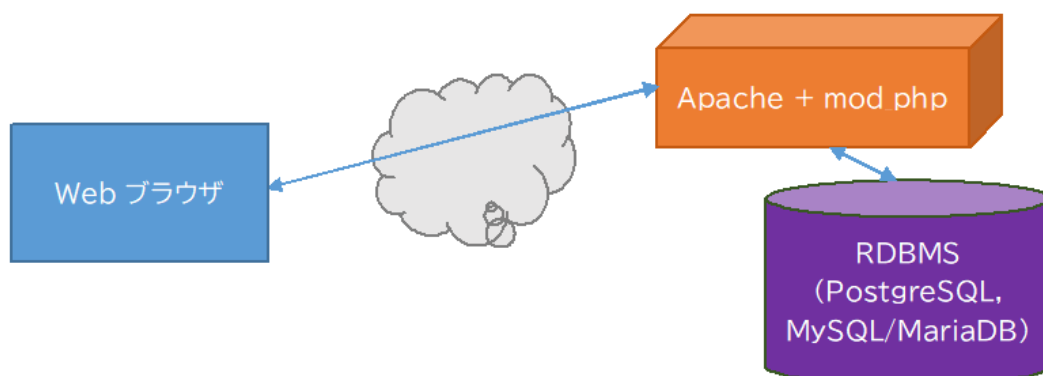
PHP 言語は、

- HTML ファイルに PHP 言語で記述したプログラムを埋め込むことができる
- シンプルな文法で比較的短時間に習得することができる
- 安価なホスティングサービスが PHP プログラムの動作に対応している

などの特徴から、Web アプリケーション開発に広く採用されています。

また、WordPress、Moodle などのコンテンツマネジメントシステム (CMS、Web サイトやブログをプログラミング知識なしに構築・運用できるソフトウェア) やオンライン学習支援システムが動作することもあり、Ruby, Python, Java など書かれたソフトウェアよりもより多くのサイト運用に使用されています。

LAPP は、Linux + Apache + PHP + PostgreSQL、LAMP は Linux + Apache + PHP + MySQL から構成された Web アプリケーションの動作環境です。PostgreSQL はトランザクション制御などに優れ、安定性が注目されています。MySQL は PostgreSQL や他のデータベースソフトウェアに比較すると、より高速に動作することを優先していて、開発者の間で人気があります。ただし、Oracle 社が MySQL の知的財産を取得してから、元の開発者たちはコードをフォーク (分岐) し、MariaDB というプロジェクトで開発を進めています。



LAPP・LAMP 構成

LAMP, LAPP によるシステムは 1 台でも動作しますが、先に解説したように複数台で構成することも可能です。

- Web サーバー (PHP モジュール) とデータベースサーバーを分離して 2 台構成にする
- ロードバランサーを入れて Web サーバーを複数台構成に冗長化する
- データベースサーバーをクラスタリングして複数台構成にする

などシステムの負荷に応じて、冗長構成を取り、より大きな負荷に耐えられるようシステム構成を柔軟に変更することができます。

また近年ではリソース消費を節約できるため、Apache の代わりに Nginx (エンジンエックス) という Web サーバーソフトが使用されるケースも増加しています。

Apache や Nginx は PHP を別のプロセスで動作させる PHP-FPM モジュール¹²を用いて、FastCGI という仕組みで PHP プログラムを処理できます。Web サーバーとは別のプロセスでサーバーサイドの PHP プログラムを動作させる仕組みです。

これにより

- Web サーバーの負荷を軽減できる
- CGI だとプログラムの実行のたびにプロセスを起動・停止するが、FastCGI は常駐するためプロセスの起動・停止のリソース消費を節約できるなどの利点があります。

¹² PHP-FPM、FastCGI Process Manager は Apache にも Nginx にも対応しています。

2. Web3 層モデル

1 の PHP は Web サーバーのプロセス上で動作しましたが、アプリケーションの処理を Web サーバーと切り離すことでそれぞれの負荷を軽減したり、冗長構成にしたりして負荷分散を図ることが可能です。その場合は、

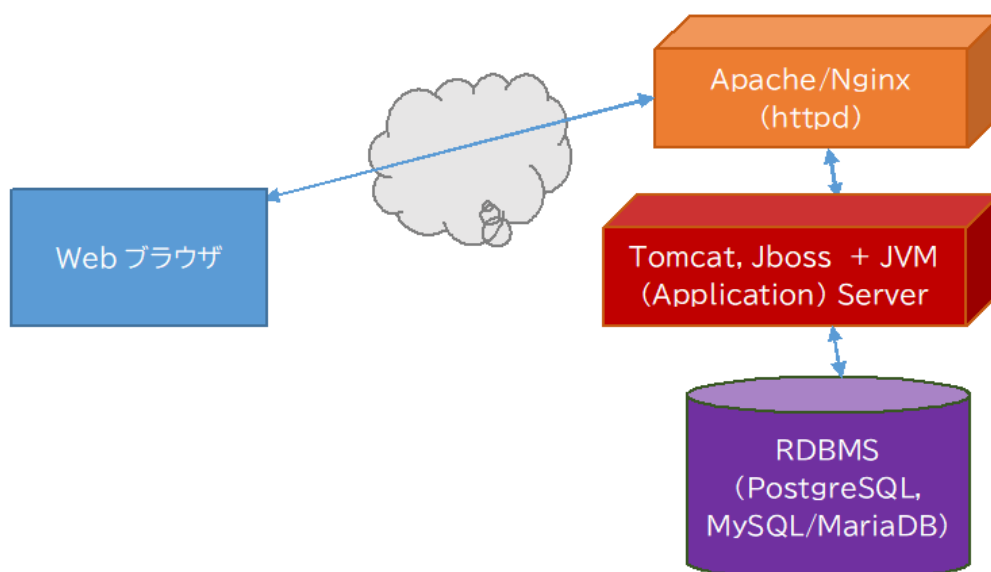
Web サーバー + アプリケーションサーバー + データベースサーバー

という 3 階層の構成を取ります。

またそれぞれのレイヤーで冗長化構成を取ることも可能です。

アプリケーションサーバーとしては、Java に対応した Tomcat, JBoss などが有名です。Tomcat は Java で書かれた Web アプリケーション (Java サーブレットや JSP [Java Server pages]) を動作させることができ、Java を使用した Web アプリケーション開発で人気があります。また、簡易な Web サーバーとして機能することが可能です。

ただし、Tomcat など Java ベースのアプリケーションサーバーは Java VM (Java の実行環境、JVM と呼ばれる。OpenJDK や Oracle JDK などに含まれる) や JRE (Java Runtime Engine) の上で動作するため、メモリ消費量が増大したり、Java 実行環境へのメモリ割当て量を増加させたりして実行環境がクラッシュしないようにするなどの工夫が必要で、一定のチューニング知識を要します。



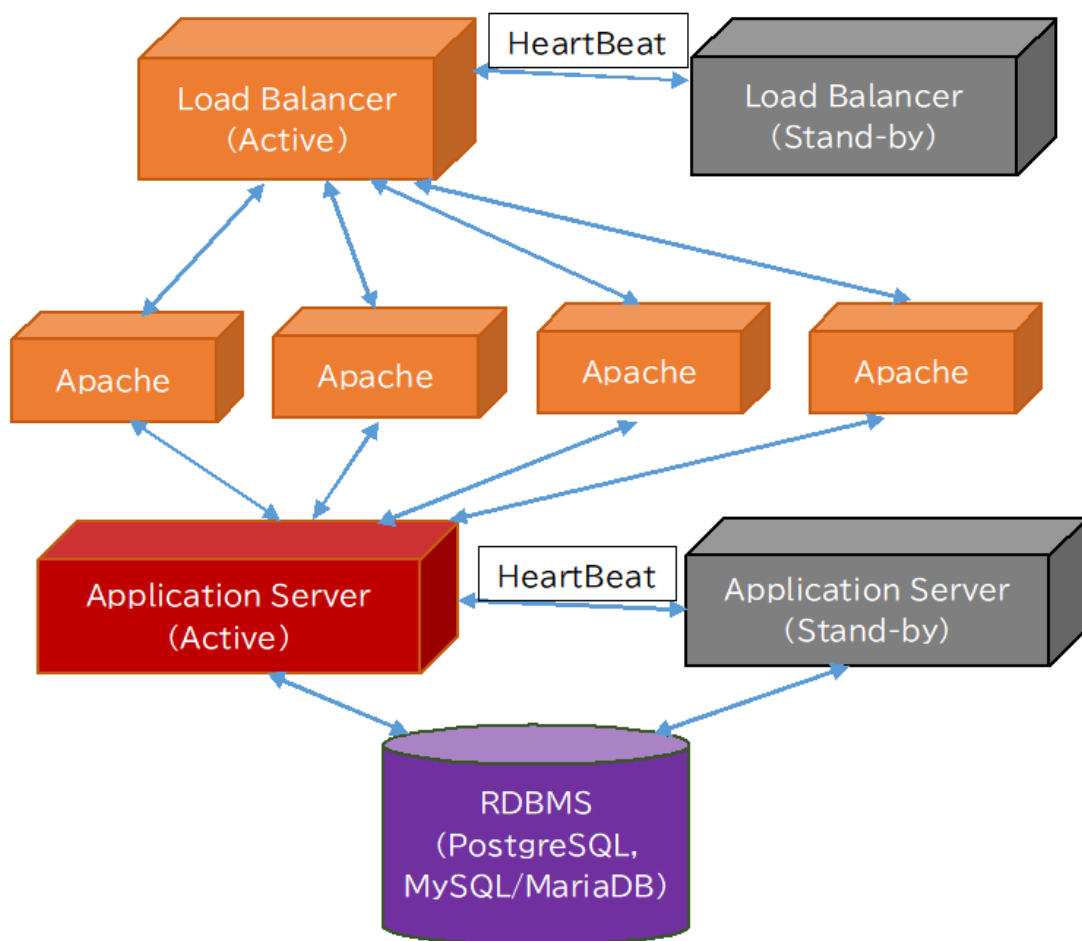
Web3 層モデルのイメージ

3. 冗長構成の Web3 層モデル

Web3 層モデルのそれぞれの層は多重化したり、冗長構成を取ることが可能です。

- 多重化はロードバランサーを入れて処理を複数台に分散する仕組み
- 冗長化は、HA クラスタリングソフトウェアでサービスの死活監視を行い、アクティブなサーバーをスタンバイ系のサーバーと切り替えて、システム停止を回避する仕組み

です。

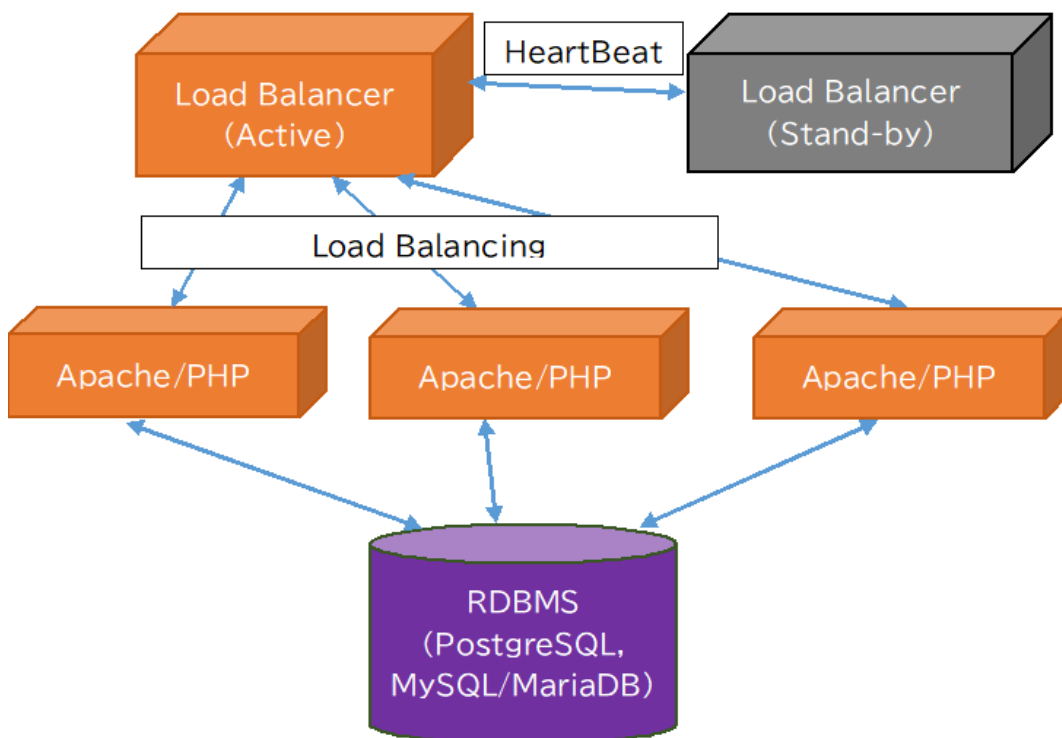


Web サーバー多重化 + アプリケーションサーバーの HA クラスタ + DB サーバー構成

4. LB/DNS ラウンドロビン + 複数 Web サーバー(スケールアウト)によるスケーラブルなシステム

3で取り上げたフロントエンドのWebサーバーの多重化、スケールアウトによるロードバランシング（負荷分散）は、バックエンドのデータベースやアプリケーションサーバーがない環境でも機能します。

例えば、WordPressサイトを以下の様に構成して大規模アクセスに耐えるように構成できます。



ロードバランサを使ってWebサーバーを多重化したイメージ

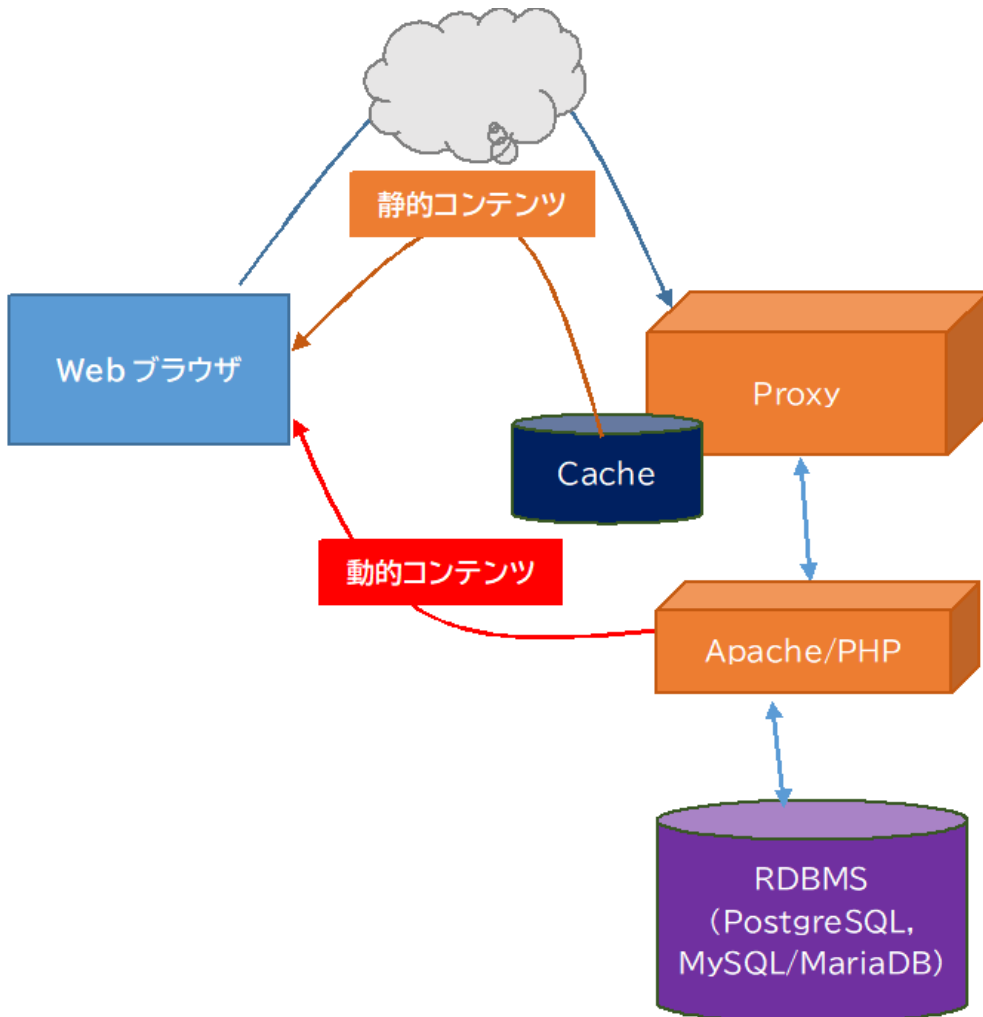
5. プロキシサーバー(キャッシュサーバー)やCDN(Content Delivery Network)を活用したWebシステム

◎ プロキシサーバー

ApacheやNginxなどのプロキシサーバー（リバースプロキシ機能）を用いると、クライアントからのリクエストのうち、

- 静的なコンテンツはキャッシュから配信する
- 動的なコンテンツ（例えばデータベースを参照して結果を動的に返す）については、

アプリケーションサーバーで処理してレスポンスを返す
というように、アプリケーションサーバーの負荷やテナントネットワーク内のトラフィックを軽減することが可能になります。



プロキシサーバーを用いて、負荷分散をした構成例

◎ CDN (Content Delivery Network) を活用したシステム

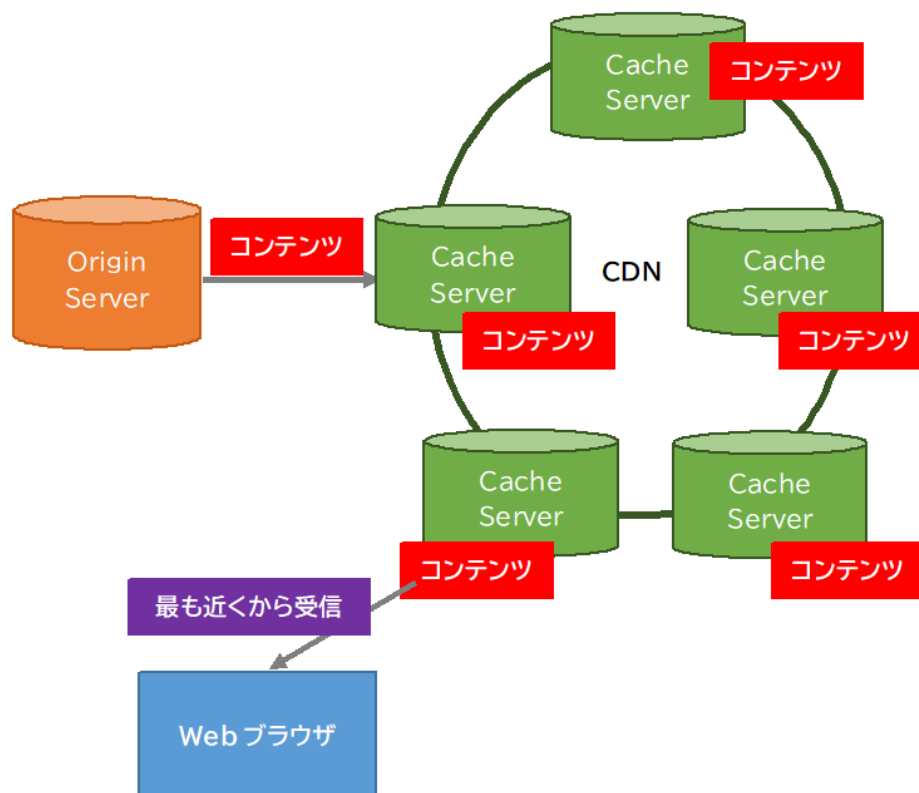
大規模の動画コンテンツ配信などを行う際に、一拠点にあるサーバーからコンテンツ配信を行うと、サーバーだけでなく、インターネット回線がいっぱいになってしまったり、動画の再生が途切れたり、再生までの時間が長くなってしまったり、などの問題が発生します。そこで、コンテンツを複数拠点のサーバーに分散させておき、地理的に最も近いサーバーからコンテンツ配信を行うための仕組みが用いられます。Content Delivery Network、コンテンツデリバリーネットワーク、または略してCDNと呼ばれます。

例えば、Akamai 社のサービスなどが有名です。また動画配信に限れば、YouTube, Vimeo, Brightcove などの動画配信および配信用の変換サービスを提供する事業者もいます。

CDN を使う場合には、オリジナルのコンテンツを CDN サービスの管理画面などからオリジンサーバーと呼ばれるサーバーにアップロードします。

オリジンサーバーは配信用のデータ変換（動画の場合にはトランスコード、複数の回線速度で切り替えて配信できるように、複数のビットレートに変換する）を行った後、各拠点にあるキャッシュサーバー（エッジノード）にデータを複製していきます。

こうした仕組みを用いることで、できるだけ配信サーバーの負荷や回線消費量を抑え、配信サービスの遅延や再生できないトラブルを減らすことが可能になります。



CDN の動作イメージ

6. メッセージングキューを活用した非同期データ処理システム

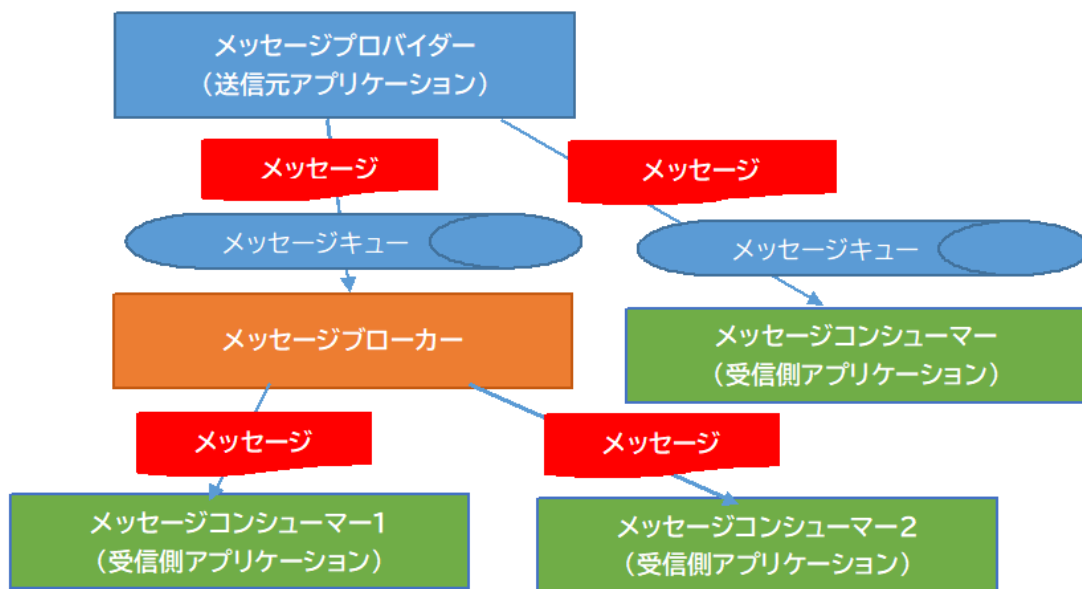
メッセージングシステムは、異なるサーバー間で処理を連携するための仕組みです。サーバ

一間のメッセージは所定のフォーマットのXMLファイルなどで記述され、アプリケーションにできるだけ依存しない形でデータの受け渡しを行い、柔軟に連携を図ることができます。こうしたアプリケーションやシステムに依存しない連携は「疎結合」と呼ばれています。

メッセージングシステムを構成する要素としては、以下のようなものがあります。

- メッセージプロデューサー： メッセージを送信するアプリケーション
- メッセージキュー： メッセージを格納するところ。キューは列という意味。
- メッセージブローカー： メッセージを中継するノード
- メッセージコンシューマー： メッセージを受け取るシステム

メッセージングブローカーには、オープンソース（Apache ライセンス）で開発されているActiveMQなどがあります。



メッセージングシステムの動作イメージ

Appendix

A.1 Zabbix の概要

Zabbix とは、システム運用時に、サーバーやネットワークの状態を監視するための運用監視ツールです。システムの状態を監視し、必要に応じてアラートをあげるために、

- ネットワーク経由でサーバーやデバイスなどの死活監視やリソース消費状況を監視する
- 指定したしきい値を超えたりエラーが検出されたりしたらメールなどで通知をする
- 指定の条件を満たしたら、コマンドを実行する

などを行う機能を有しています。

1. Zabbix でできること

Zabbix は以下のような機能を有しています。

1.1 さまざまなデバイスやアプリケーションの監視

監視対象には以下のようなものが挙げられます。

- Zabbix エージェントをインストールしたサーバーや仮想マシン
- SNMP で監視可能なネットワークデバイス
- エンドユーザーの Web ブラウザー
- Web アプリケーション
- データベース

1.2 障害検知

1 で挙げたような監視対象から収集した稼働状況やパフォーマンスについての情報から、障害ステータスを自動的に検知します。Zabbix は障害検知に関連して以下のような機能を提供します。

- 柔軟な定義設定
- 障害の切り分けと障害復旧状況
- 障害の緊急度・深刻度 (Severity) レベル
- 原因の解析・推定
- 異常検知 (障害発生はしていないが、リソースがひっ迫しているなど)
- 傾向予測

1.3 可視化

Zabbix に内蔵された Web インターフェースを用いると、以下のような情報にアクセスできます。

- ウィジェット（機能ごと）ベースのダッシュボード（一覧表示画面）
- グラフ
- ネットワーク構成図
- スライドショー
- ドリルダウンレポート（詳細分析機能）

1.4 アラート通知と復旧

Zabbix を用いると、システム管理者にさまざまな経路で障害の発生を通知したり、復旧用スクリプトの実行などが可能になります。

- メールによる通知
- あらかじめ設定した復旧用スクリプトの実行
- 問題の深刻度によるレポート機能（エスカレーション）のカスタマイズ
- メッセージのカスタマイズ

1.5 セキュリティと認証

Zabbix では、エージェントやノード間の通信においてデータ保護を実現するためのセキュリティ機能を提供します。

- Zabbix コンポーネント（サーバー・エージェント・WebUI）間での強力な暗号化
- 複数の認証方法への対応
 - Open LDAP, Active Directory(Microsoft の認証プロダクト)
- ユーザー権限の設定

1.6 簡単な導入

Zabbix は導入・運用のハードルを下げるために以下のような機能が提供されます。

- 主要ディストリビューション向けの公式インストールパッケージ
- すぐに使えるテンプレート機能
- テンプレートのカスタマイズ機能
- ユーザコミュニティで公開されている多数のテンプレート

- 有償のテンプレート作成サービス

1.7 オートディスカバリー

Zabbix を用いるとコンポーネントの追加・削除・変更を自動で行えます。

- ネットワークディスカバリー
定期的にネットワークをチェックし、デバイスタイプ、IP アドレス、ステータス、アップタイム・ダウンタイムなどを発見し、問題を発見した際には事前に定義したアクションを実行します。
- ローレベルディスカバリー
あらかじめ定義したターゲットデバイスに対して、監視対象となるファイルシステム項目、トリガー（アクションを実行するきっかけやしきい値）、グラフを自動で生成します。
- アクティブエージェントの自動登録
Zabbix エージェントがインストールされている機器が接続されると自動的に監視をスタートします。

1.8 分散監視

Zabbix は分散したデバイスやアプリケーションの集中管理を実現します。例えば、以下のような分散監視が可能になります。

- 数千の監視ターゲットからのデータ収集・可視化
- ファイアウォールや DMZ（ネットワーク上の非武装領域、ファイアウォールと同じゾーンまたは外にある領域）越しの監視
- ネットワーク障害時にも（可能な限り）データを収集
- 監視ターゲット上で、リモート環境からカスタムスクリプトを実行

1.9 Zabbix API

Zabbix は API（アプリケーション・プログラミング・インターフェース）を提供し、他のアプリケーションから、Zabbix の機能にアクセスしたり連携したりすることができます。例えば、以下のような連携が可能になります。

- API 経由での Zabbix 管理の自動化
- 200 以上のメソッド（関数）を用いた Zabbix の機能へのアクセス
- Zabbix と連携するアプリケーション開発用の API の提供
- 外部アプリケーションとの連携

- 設定管理、チケットシステム（障害トラッカーへの自動登録・チケット発行）
- 設定やログの取り出し、管理

2. Zabbix を構成するコンポーネント

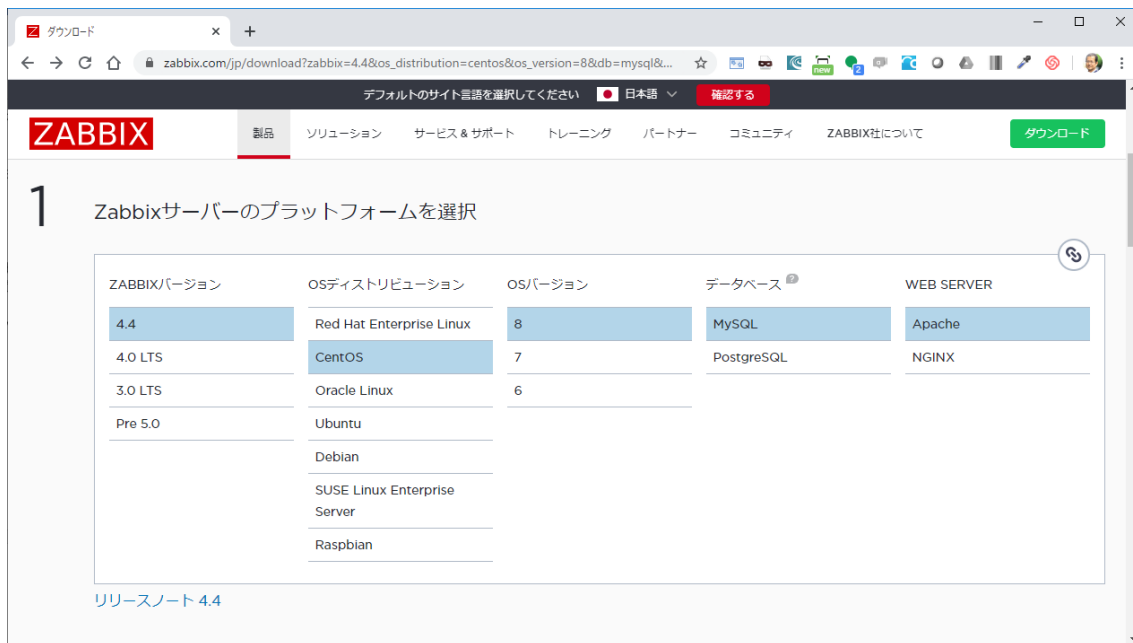
Zabbix は三つのコンポーネントから構成されています。

- Zabbix サーバー
監視ターゲットからデータを取得してデータベースに保存したり、特定の条件を満たした場合に通知を送信したりする
- Zabbix Web UI
収集したデータやリソース消費状況などを表示したり、監視内容の設定を Web ベースで実行したりする
- Zabbix エージェント
監視ターゲットとなるマシンにインストールして常時起動し、サーバーから監視を実行できるようにするクライアント

3. Zabbix のインストール (2020 年 4 月現在)

Zabbix の最新安定版のバージョンは 4.4, 長期サポート対応版は 4.0 となっています。

Zabbix の公式サイトでは、バージョンやディストリビューション、データベースソフトウェアの種類などを選択すると、その組み合わせに対応したインストール手順が表示されるようになっています。



それでは、CentOS 8.0 と MySQL、Apache の組み合わせでインストールを実行してみましょう。

(1) Zabbix のリポジトリの追加とテンポラリファイルのクリーンアップ (削除)

```
# rpm -Uvh https://repo.zabbix.com/zabbix/4.4/rhel/8/x86_64/zabbix-release-4.4-1.el8.noarch.rpm
# yum clean all
```

(2) Zabbix サーバー、Web インターフェース、Apache 設定ファイル、Zabbix エージェント

```
# yum install zabbix-server-mysql zabbix-web-mysql zabbix-apache-conf zabbix-agent
```

ントのインストール

(3) MySQL データベース上に Zabbix の収集したデータを保存するデータベースインスタンスを追加して zabbix ユーザーにアクセス権を追加する。

```
# mysql -uroot -p
<password>
mysql> create database zabbix character set utf8 collate utf8_bin;
mysql> grant all privileges on zabbix.* to zabbix@localhost identified
by '<zabbix ユーザのパスワード>';
mysql> quit;
```

Zabbix サーバー上で、データベースのスキーマ (テーブル構成) や初期データのインポートを行います。

```
# zcat /usr/share/doc/zabbix-server-mysql*/create.sql.gz | mysql -
uzabbix -p zabbix
```

(4) Zabbix の設定ファイル (/etc/php-fpm.d/zabbix.conf) にデータベース設定 (zabbix ユーザの DB パスワード) を追記する。

```
DBPassword=<zabbix ユーザーのパスワード>
```

(5) タイムゾーンの設定を PHP の設定ファイル (/etc/php-fpm.d/zabbix.conf) に追加する。

```
; php_value[date.timezone] = Europe/Riga
⇒
php_value[date.timezone] = Asia/Tokyo
```

(6) Zabbix サーバーと Apache, PHP(FastCGI モードの PHP プロセス)を起動する。また OS 再起動時に自動起動するための enable 設定を行う。

```
# systemctl restart zabbix-server zabbix-agent httpd php-fpm
# systemctl enable zabbix-server zabbix-agent httpd php-fpm
```


(7) Web インターフェースにアクセスしてフロントエンドの設定を行う。

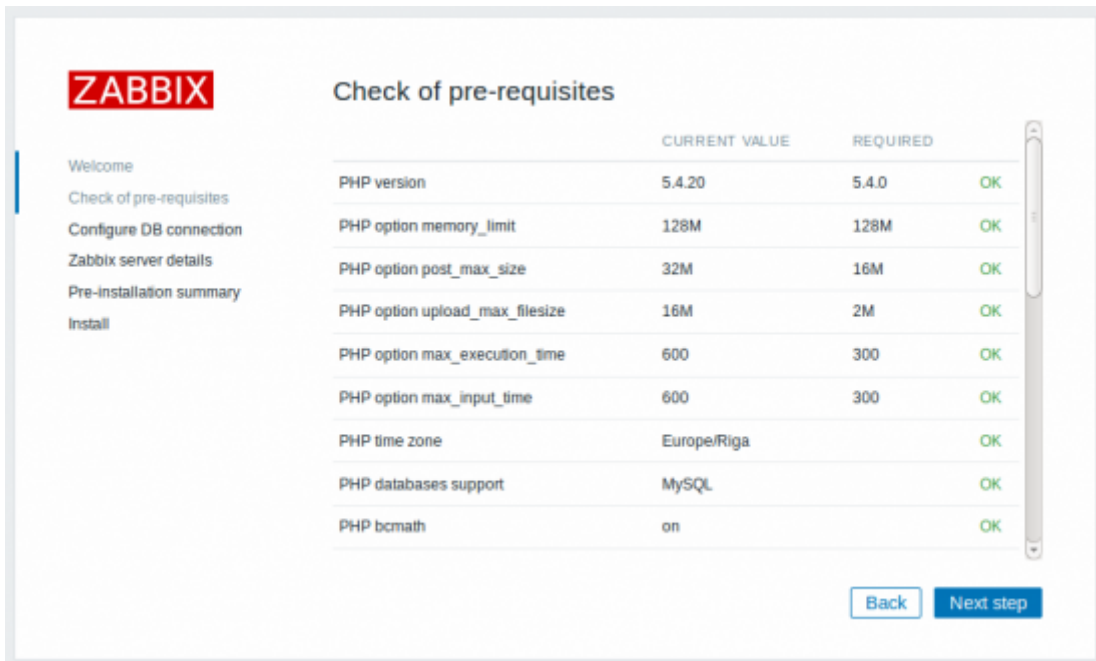
`http://Zabbix サーバーをインストールしたサーバーのホスト名・IP/zabbix`

にアクセスします。

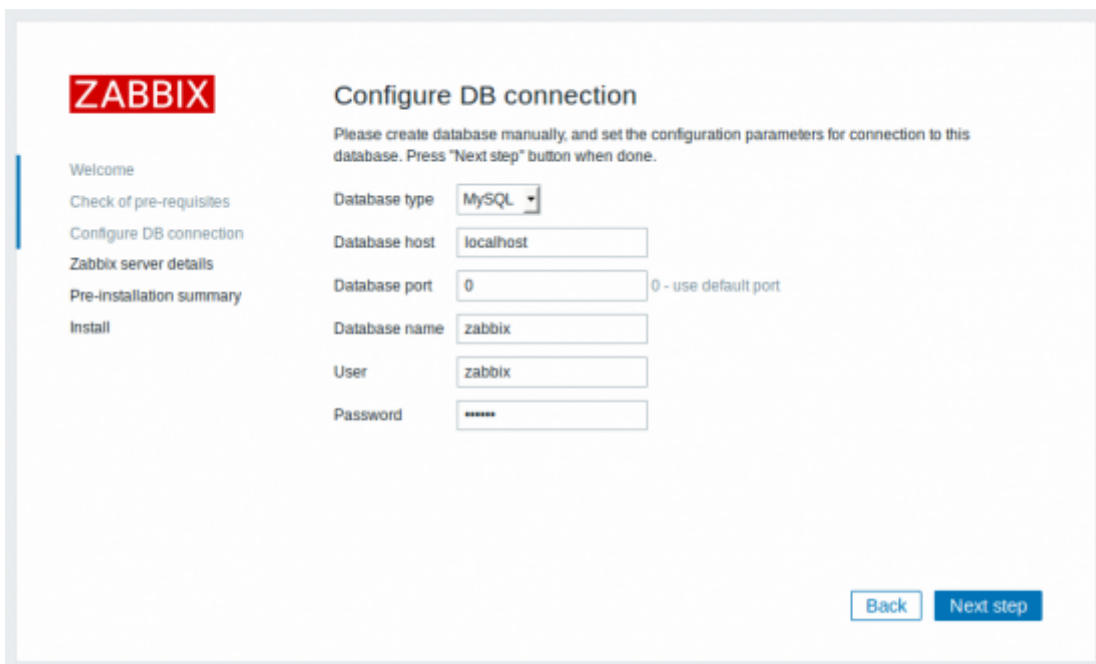
◎フロントエンドの設定 URL を開いたところ



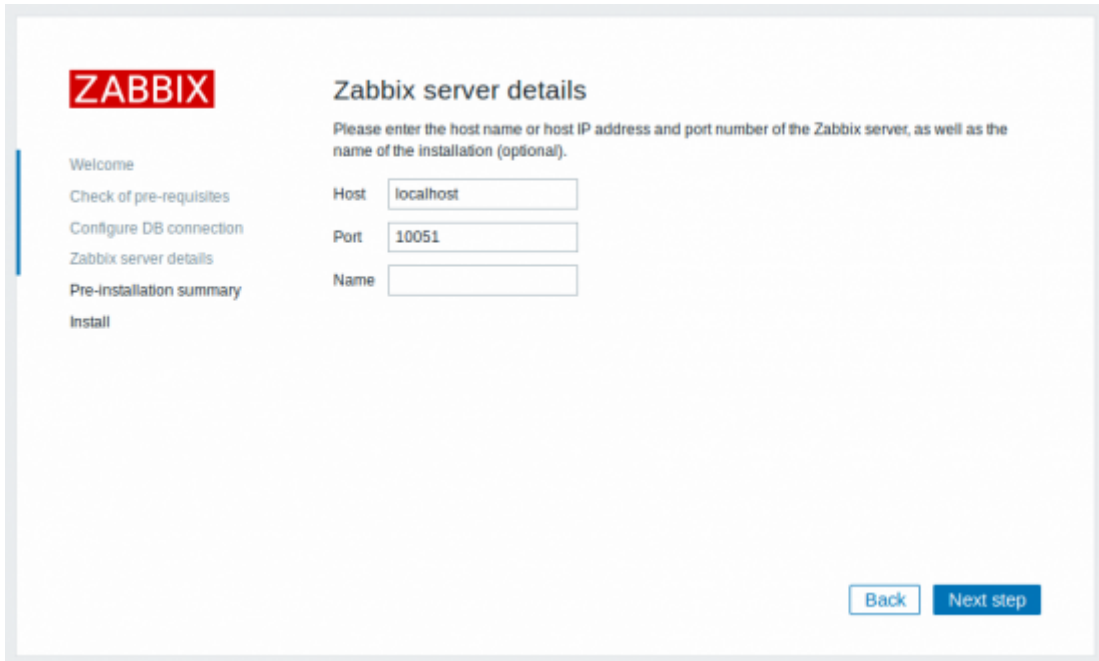
インストール条件の確認 (問題があれば PHP 設定などを変更)



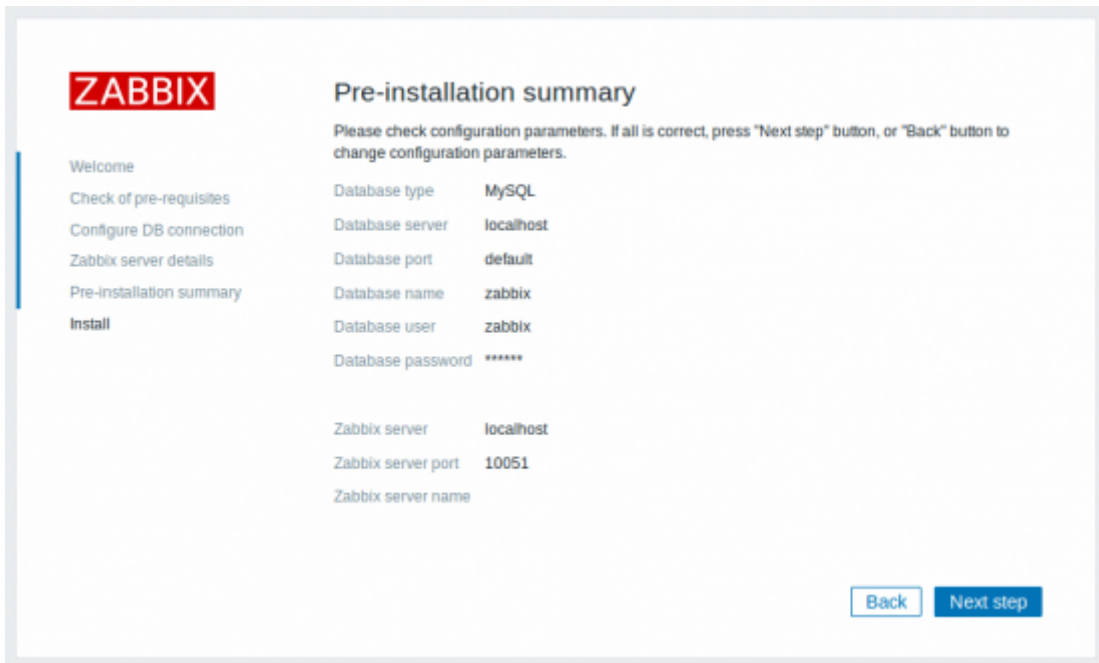
データベースの接続設定を入力して「Next step」をクリックします。



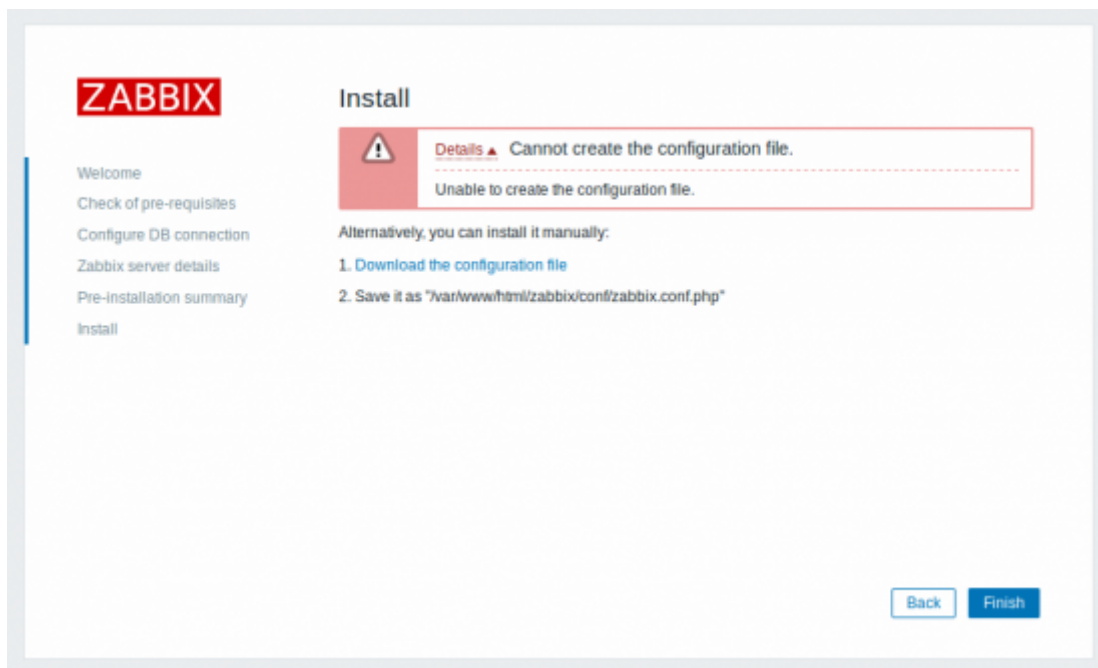
Zabbix サーバーの設定を行います。Name はオプションですが、入れておくと管理画面に表示され、他のサーバーと識別しやすくなります。



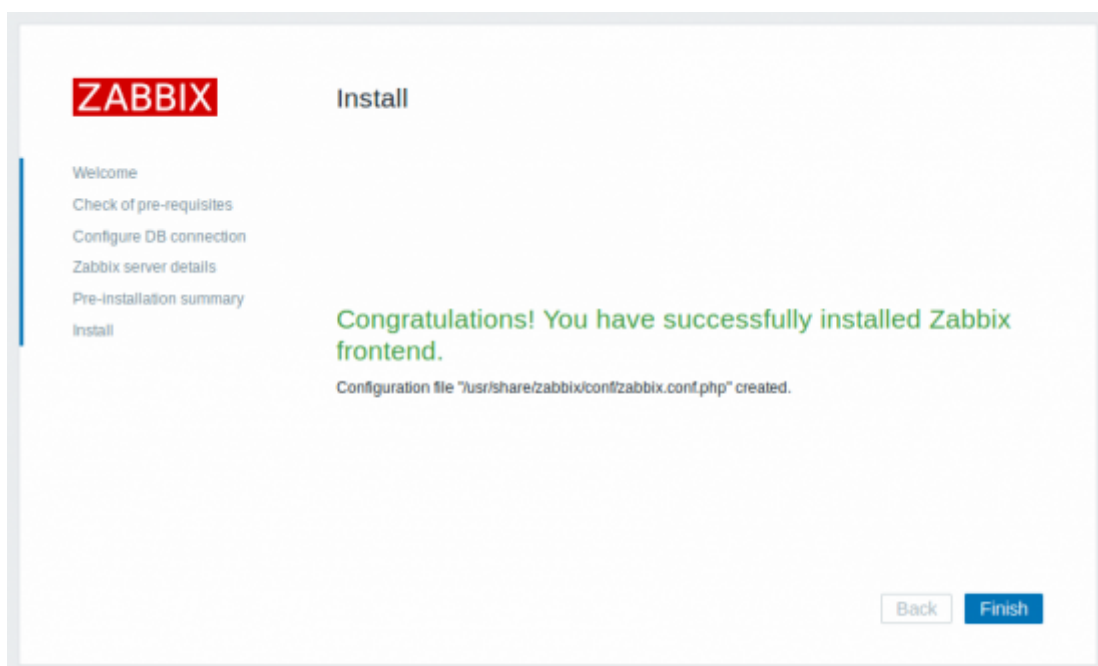
すると設定のサマリー（要約）ページが表示されます。



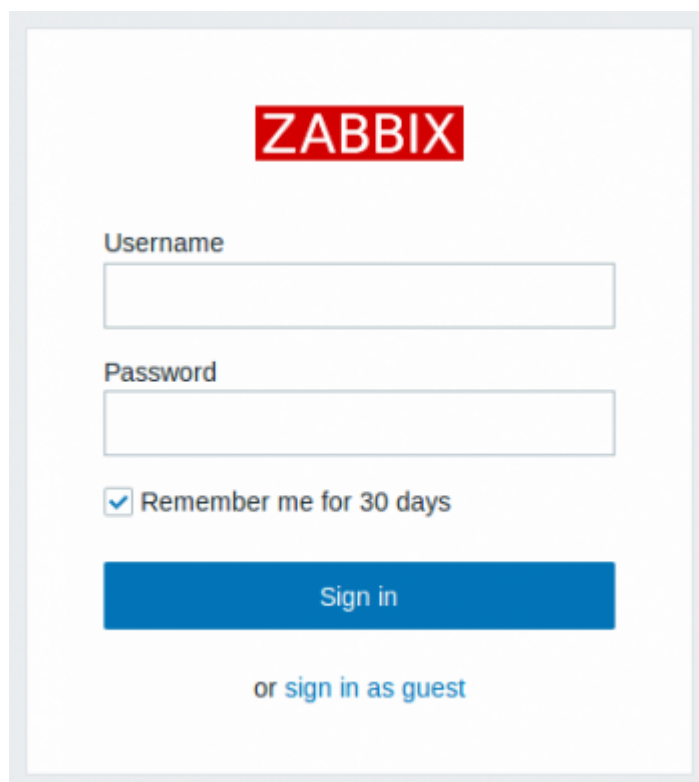
設定ファイルをダウンロードして、zabbix のドキュメントルートの conf ディレクトリに保存します。



インストールを完了します。



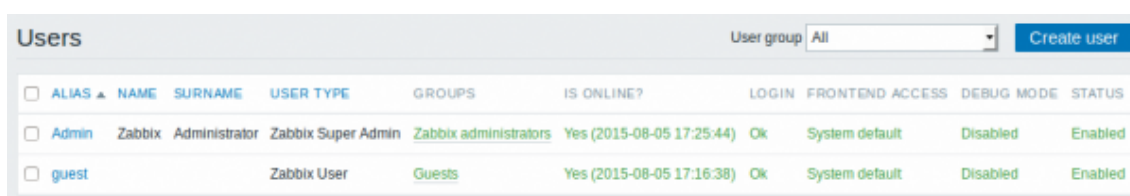
インストールが完了したら、ログイン画面が出るので、初期 ID・パスワード (Admin・zabbix) でログインしましょう。



The image shows the Zabbix login interface. At the top center is the ZABBIX logo in a red box. Below it are two input fields: 'Username' and 'Password'. Under the password field is a checkbox labeled 'Remember me for 30 days' which is checked. Below these is a blue 'Sign in' button. At the bottom, there is a link that says 'or sign in as guest'.

4. Zabbix の初期設定（ユーザの追加・設定）

無事にログインできたら、Administration ⇒ Users というメニューを開いて、ユーザー一覧を表示しましょう。



The image shows the 'Users' management page in Zabbix. It features a table with columns for user details and a 'Create user' button in the top right corner. The table lists two users: 'Admin' and 'guest'.

<input type="checkbox"/>	ALIAS ▲	NAME	SURNAME	USER TYPE	GROUPS	IS ONLINE?	LOGIN	FRONTEND ACCESS	DEBUG MODE	STATUS
<input type="checkbox"/>	Admin	Zabbix	Administrator	Zabbix Super Admin	Zabbix administrators	Yes (2015-08-05 17:25:44)	Ok	System default	Disabled	Enabled
<input type="checkbox"/>	guest			Zabbix User	Guests	Yes (2015-08-05 17:16:38)	Ok	System default	Disabled	Enabled

右上の「Create User」をクリックして、ユーザーの追加を行います。

Users

User Media Permissions

* Alias

Name

Surname

* Groups

* Password

* Password (once again)

Language

Theme

Auto-login

Auto-logout (min 90 seconds)

* Refresh (in seconds)

* Rows per page

URL (after login)

「Add」をクリックしてユーザーが追加されたら、続いては上部の「Media」タブをクリックして通知設定を行います。

Media



Type

* Send to [Remove](#)

[Add](#)

* When active

Use if severity Not classified
 Information
 Warning
 Average
 High
 Disaster

Enabled

- Send to は送信先のメールアドレス
- When active は、通知を送信する時間帯の指定
- Use if severity は、情報や障害の深刻度により送受信の可否を設定できます。例えば、Disaster は障害発生情報、Warning は警告や注意、それ以外はパフォーマンスなどの情報です。
- Enabled では通知の有効化・無効化を切り替えられます。

続いては「Permissions」タブをクリックして、監視対象のデバイスやサーバーの選択をします。

User groups

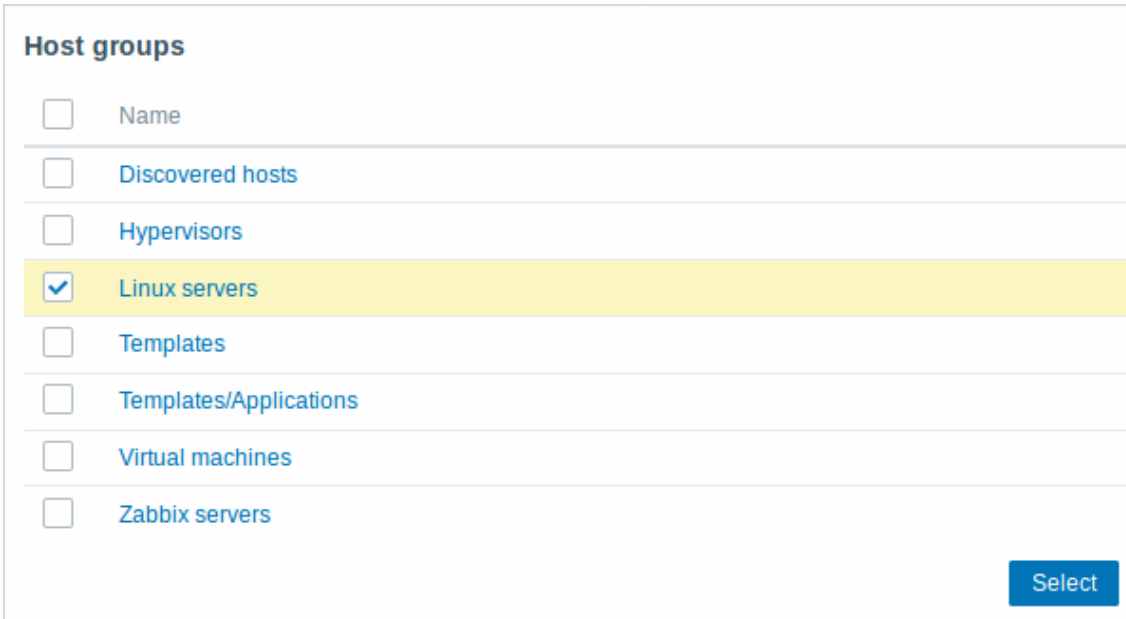
User group Permissions

Host group	Permissions
*	None

[Add](#)

Select をクリックすると選択肢の一覧が表示されます。

例えば、下の図では「Linux Servers」へのアクセスを選択しています。それ以外には、自動検知されたホスト、ハイパーバイザー、テンプレート、仮想マシン、Zabbix サーバーなどが選択可能となっています。



The screenshot shows a list of host groups under the heading "Host groups". The options are:

- Name
- Discovered hosts
- Hypervisors
- Linux servers
- Templates
- Templates/Applications
- Virtual machines
- Zabbix servers

A blue "Select" button is located at the bottom right of the list.

チェックを入れたら「Select」をクリックして、設定を保存しユーザー情報のページに戻ります。

以上の設定で、Web インターフェースから Linux サーバーの情報にアクセスできるようになります。

◎インターフェースの日本語化

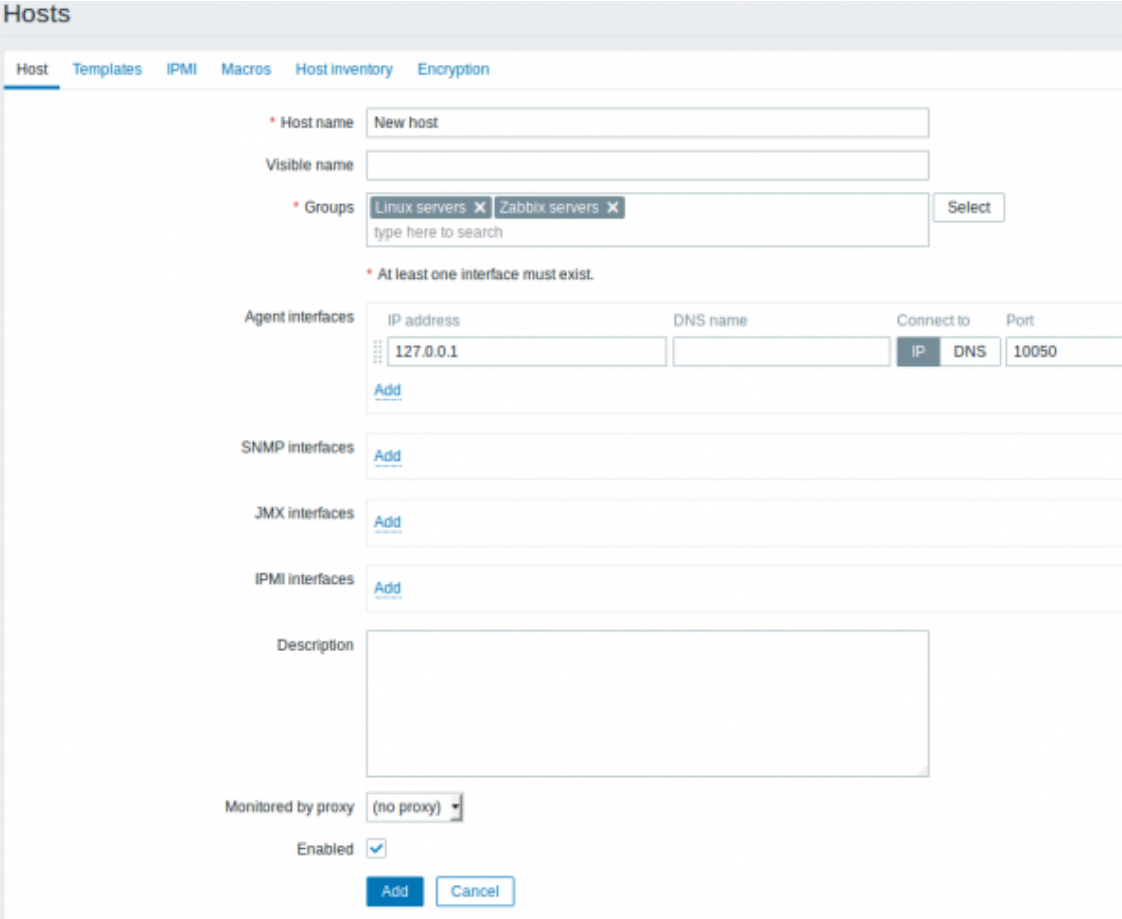
ページ右上のユーザプロフィールを表示するアイコンをクリックすると、言語設定を切り替えることができます。

デフォルトでは英語メニューになっています。もし、日本語メニューで使用したい場合には、「Japanese(ja_JP)」を選択すると日本語メニューに切り替えることができます。

5. 監視対象の追加と表示例

あとは、監視ターゲットの情報を追加して、監視する対象（例えば、CPU 使用率やメモリ使用量）などを指定します。

まずは監視対象のホストを追加します。



The screenshot shows the 'Hosts' configuration page in Zabbix. The 'Host name' field is filled with 'New host'. The 'Groups' field contains 'Linux servers' and 'Zabbix servers'. The 'Agent interfaces' section has a table with one entry: IP address '127.0.0.1', DNS name (empty), Connect to 'IP', DNS, and Port '10050'. There are 'Add' buttons for Agent, SNMP, JMX, and IPMI interfaces. The 'Monitored by proxy' dropdown is set to '(no proxy)'. The 'Enabled' checkbox is checked. 'Add' and 'Cancel' buttons are at the bottom.

Zabbix エージェントのホストを追加する画面¹³

監視対象のホストを追加したら、次は監視するリソースを設定します。

例えば、下の図では CPU 使用率のデータを設定しています。

¹³ <https://www.zabbix.com/documentation/4.0/manual/quickstart/host>

Item Preprocessing

* Name CPU Load

Type Zabbix agent

* Key system.cpu.load Select

* Host interface 127.0.0.1 : 10050

Type of information Numeric (float)

Units

* Update interval 30s

Custom intervals	Type	Interval	Period	Action
	Flexible Scheduling	50s	1-7,00:00-24:00	Remove

[Add](#)

* History storage period 90d

* Trend storage period 365d

Show value As is [show value mappings](#)

New application

Applications

Populates host inventory field -None-

Description

Enabled

Add Cancel

localhost の CPU 使用率を監視アイテムに追加する画面¹⁴

ダッシュボードに戻ると、以下のような CPU 使用率の推移を可視化したグラフを得ることができます。

¹⁴ <https://www.zabbix.com/documentation/4.0/manual/quickstart/item>



Zabbix で CPU 使用率の推移を可視化したグラフの例¹⁵

◎SELinux を有効化した状態で Zabbix を動作させるための設定

SELinux が有効になっている状態では、Zabbix が外部と通信することができません。そこで、Zabbix 用の SELinux のポリシーパッケージをインストールして、OS を再起動しまし

```
# semodule -i my-zabbixserver.pp
```

よう。

¹⁵ <https://www.zabbix.com/documentation/4.0>

A.2 Pacemaker, Corosync のインストールと主な設定

以下では 2 台の CentOS 7 マシンに Pacemaker と Corosync を導入し、簡単なクラスタ構成を構築してみます。

Pacemaker と Corosync は yum コマンドでインストールできます。

```
$ yum install pacemaker-all
```

yum コマンドを使用することで、リソースエージェントのライブラリなども同時にインストールされます。

インストールが完了したら、クラスタ制御部 (Corosync) とリソース制御部 (Pacemaker) それぞれの設定を行います。

まずはクラスタの基本的な動作設定を行います。

/etc/corosync/corosync.conf という設定ファイルにクラスタの定義やハートビート (死活監視) の設定などを記述します。設定ファイルを作成したら、もう 1 台のマシンに SCP などで転送して同じディレクトリに配置します。

```

compatibility: whitetank
aisexec { # 実行ユーザーとグループの設定
    user: root
    group: root
}
service {
    name: pacemaker # 使用するクラスタ
    ver: 0 # バージョン番号
}
totem { # ハートビート（死活監視）の通信方法の指定
    version: 2 # 設定ファイルのバージョン
    secauth: off # メッセージの認証に HMAC/SHA1 認証使用の有無
    threads: 0 # 認証に使用するスレッド数
    rrp_mode: none # 冗長リングのモード
    clear_node_high_bit: yes # nodeid の最高位を 0 にするか
    token: 4000 # タイムアウト値
    consensus: 10000 # 新しい構成のラウンドを開始するまでの待機時間
    rrp_problem_count_timeout: 3000 # 障害カウンタに減算処理をするまでの時間
    interface {
        ringnumber: 0 # 冗長リングの番号
        bindnetaddr: 192.168.10.0 # ネットワークアドレス
        mcastaddr: 226.94.1.1 # マルチキャストアドレス
        mcastport: 5405 # マルチキャストで使用するポート
    }
}
logging{ # ログの設定オプション
    fileline: on # 表示されるファイルと行
    to_syslog: yes # ログの出力先
    syslog_facility: local1 # シスログのファシリティタイプ
    syslog_priority: info # シスログのロギングレベル
    debug: off # デバッグ出力を記録するか
    timestamp: on # 全ログにタイムスタンプをつけるか
}

```

/etc/syslog.conf に先ほどの設定ファイルで指定したロギングの設定を追加します。local1 について、/var/log/messages ではなく、/var/log/ha-log にログを記録するように設定しています。local.none は/var/log/messages にログを記録しない設定です。こちらも最初のマシンで設定を終えたら、もう 1 台のマシンに SCP などファイルを送信しておきます。

```
*.info;mail.none;authpriv.none;cron.none;local1.none /var/log/messages

local1.* /var/log/ha-log
```

上記の設定を終えたら、クラスタを起動します。クラスタに属する各サーバー上で起動します。これでリソースエージェントによるサービスの死活監視、ハートビートによるアクティブ・スタンバイのフェイルオーバーが開始されます。

```
$ systemctl start corosync.service
```

corosync の動作を確認するには、1 台目のマシンの corosync を停止させ、オンラインのマシンが切り替わるか確認します。

まずは、2 台目のマシンで crm_mon を実行します。

```
[root@SRV2] crm_mon -Afro
Online: [ SRV1 SRV2 ]

Full list of resources:
Resource Group: VM-1g
    SRV1-VIP    (ocf::heartbeat:IPaddr2):           Started SRV1
    SRV1-apache (lsb:httpd_vm1):                 Started SRV1
```

すると最初は 2 台ともオンラインと認識されています。

そして、1 台目の corosync を停止します。

```
[root@SRV1]# service corosync stop
Signaling Corosync Cluster Engine (corosync) to terminate: [ OK ]
Waiting for corosync services to unload:.... [ OK ]
```

そして、2 台目の `crm_mon` の表示を確認します。

```
Online: [ SRV2 ]
OFFLINE: [ SRV1 ]
Full list of resources:
Resource Group: VM-1g
  SRV1-VIP (ocf::heartbeat:IPaddr2): Started SRV2
  SRV1-apache (lsb:httpd_vm1): Started SRV2
```

すると `corosync` を停止した 1 台目のホストはオフラインになり、2 台目のマシンで Apache が動作していることがわかります。

LinuC レベル 2 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

Copyright© 2021 LPI-Japan. All Rights Reserved.