LinuC

Linux研修オンライン【例題で学ぼう】

コンテナ/Dockerの仕組みと操作

株式会社ゼウス・エンタープライズ
（LPI-Japanプラチナスポンサー・アカデミック認定校）
鯨井貴博

ZEUS enterprise

LPI-JAPAN

講師紹介

鯨井貴博(くじらいたかひろ)

・LinuCエヴァンジェリスト
・OSSJ(Open Source Summit Japan)運営ボランティアリーダー
・KubeCon/CloudNativeCon Japan運営ボランティアリーダー

2000年 大学卒業後、建設業界に就職。
2007年 IT業界に転職し、Linuxを学ぶ。
2008年 OSS界隈で勉強会などのイベントに参加し始める。
LinuC(当時はLPIC)のベータ試験に参加。
2012年 社内講師として活動を開始。
個人ブログ、メルマガ執筆、LinuCセミナー、専門学校での授業など行う。
2015年 ACCEL(Apache Cloudstack Certification Exam by LPI-japan)認定第1号。
2022年 OSSJ運営ボランティアリーダー
2025年 KubeCon/CloudNativeCon Japan運営ボランティアリーダー

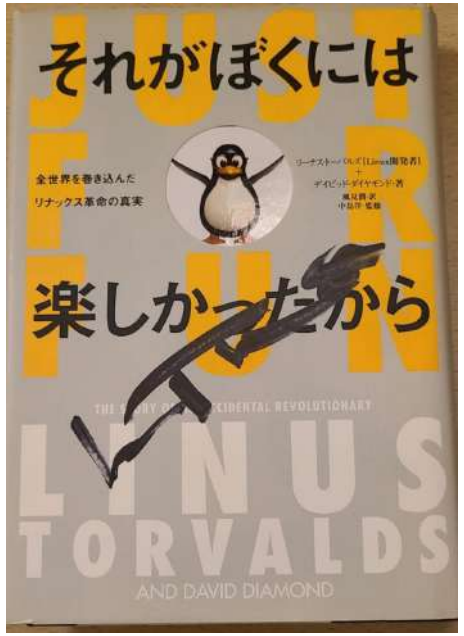
Opensourcetehブログ
https://www.opensourcetech.tokyo/

講師紹介

技術を学習する上で大事にしていること

・その技術が生まれた背景や開発者の思いの理解
・実際に使ってみる

Linux開発者 Linus Torvalds



The Linux Kernel Archives
https://www.kernel.org/category/releases.html

Linuxカーネルメンテナー Greg Kroah-Hartman

Longterm release kernels

Version	Maintainer	Released	Projected EOL
6.12	Greg Kroah-Hartman & Sasha Levin	2024-11-17	Dec, 2026
6.6	Greg Kroah-Hartman & Sasha Levin	2023-10-29	Dec, 2026
6.1	Greg Kroah-Hartman & Sasha Levin	2022-12-11	Dec, 2027
5.15	Greg Kroah-Hartman & Sasha Levin	2021-10-31	Dec, 2026
5.10	Greg Kroah-Hartman & Sasha Levin	2020-12-13	Dec, 2026
5.4	Greg Kroah-Hartman & Sasha Levin	2019-11-24	Dec, 2025

Open Source Summit Japan

2013年 初参加(Attendee)
2014年 ボランティアメンバーとして運営を手伝う
2022年 ボランティアリーダーとして運営を行う

2025/12/8-10(虎ノ門)のボランティアメンバーは、10月頃募集開始予定♪
※気になる方は、以下の QRコードから要フォロー

Open Source Summit Japan
https://events.linuxfoundation.org/open-source-summit-japan/

世界と繋がる、グローバルカンファレンスの舞台裏！　Open Source Summit Japan2024運営ボランティアの体験レポート Part1/Part2
https://thinkit.co.jp/article/37762
https://thinkit.co.jp/article/37777

講師紹介

KubeCon/CloudNativeCon Japan

2025年 日本初開催

※2026年は7月29/30日 横浜開催

KubeCon/CloudNativeCon Japan2025
https://www.cncf.io/reports/kubecon-cloudnativecon-japan-2025/

株式会社ゼウス・エンタープライズ

教育案件や人材サービスを探している方、
ご連絡お待ちしてます！

https://www.zeus-enterprise.co.jp/solution/service/
https://www.it-training.tokyo/

アジェンダ
- LinuC、およびその学習方法
- コンテナ/Dockerについて
- 例題と解説
- 振り返り

本日のゴール
- コンテナの概念の理解
- Docker環境の操作方法の理解
- LinuCレベル2資格対策の理解

LinuC、およびその学習方法

Linux技術者認定「LinuC（リナック）」とは

■LinuCとは

クラウド／DX時代のITエンジニアに求められるシステム構築から運用管理に
必要なスキルを証明する技術者認定です。

✔ クラウド活用に役立つスキルの習得

- オンプレミス／仮想化・コンテナを問わず様々な環境下でのサーバー構築
- 他社とのコラボレーションの前提となるオープンソースへの理解

✔ 習得できるスキルが実践的

問題作成にはトップエンジニアも参加するコミュニティ内の意見を
取り込むことで、本当に必要な内容を網羅的に盛り込んでいます。

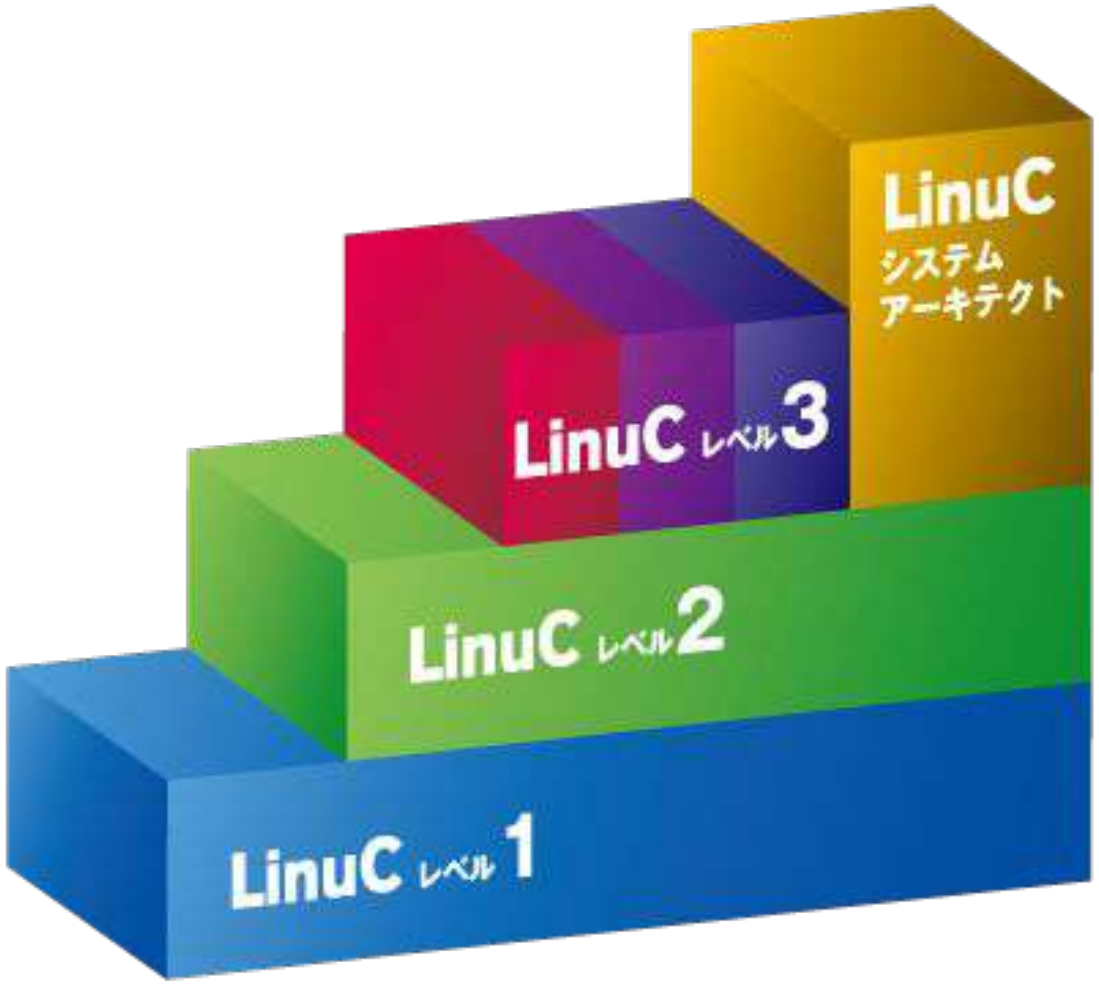
✔ 上流工程を担うアーキテクトの領域までカバー

システムの運用管理からアーキテクチャ設計までの4つのレベルを
ひとつずつ習得していくことで、活躍できるエンジニアとして必要な
スキルを網羅的に身につけていくことができます

LinuC
Linux技術者認定

LinuCの4つのレベル

LinuCは、サーバーの運用管理からアーキテクト設計まで、システム開発・運用に必要な知識とスキルを体系立てて習得することができます。



LinuC システムアーキテクト
ITプロジェクトを成功に導く上級エンジニア
SA01試験 | SA02試験

LinuC レベル3
高度な技術力を備えた特定分野のスペシャリスト
304試験（仮想化&高可用性）
300試験（混在環境） | 303試験（セキュリティ）

LinuC レベル2
仮想マシン・コンテナを含むLinuxシステム、ネットワークの設計・構築
201試験 | 202試験

LinuC レベル1
物理／仮想Linuxサーバーの操作・運用
101試験 | 102試験

LinuCの学習方法

学習の具体的な進め方(2~3か月程度)

試験範囲の確認(LinuC HP)

10月頃、受験しようという目標を立てる

↓

LinuC認定教材の購入・1週目読込

https://lpi.or.jp/linuc1/book.shtml
https://lpi.or.jp/linuc2/book.shtml

↓

LinuC認定教材・LinuC技術解説動画を参考に、実機操作(サーバ構築やコマンド操作)を試す
※操作やトラブルシュートで力が身に付く！

https://linuc.org/measures/movie/

↓

LinuC認定教材 2週目読込

10月26日頃、受験しようと決める

↓

問題集やメルマガサンプル問題で理解力確認
※理解不足箇所の洗い出し

https://linuc.org/study/samples/

↓

LinuC認定教材 3週目
※弱点の補強

↓

受験申込

受験日の変更も可能なので安心

↓

問題を8~9割以上、正解となるまで繰り返し解く
苦手な部分を重点的に復習

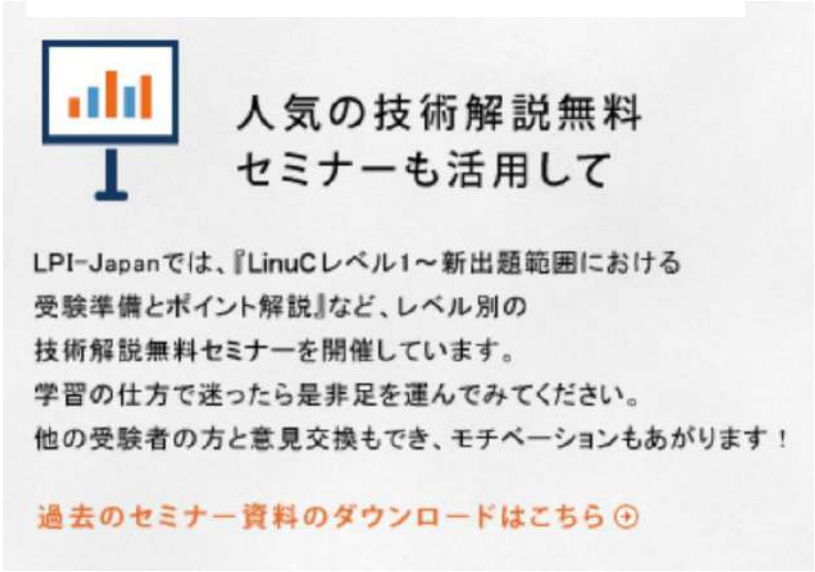
・受験まで継続して学習すること
・繰り返し学習し、理解度 /問題正解率を高めた状態で受験すること

↓

受験

LinuCの学習方法

①出題範囲の内容について調べてみる
　公式ドキュメント・技術書など
②実際に操作してみる
　これが大事！
③学習の補助教材などを利用する
・メールマガジン
・標準教科書
・過去のセミナー資料
　詳細は、https://lpi.or.jp/learning/



LinuCの学習方法

④過去セミナーの動画
https://linuc.org/measures/movie/

LinuC-2

201試験

主題	セミナー動画	講師
201全般	LinuCレベル2 201試験概要解説 サーバー構築のできるLinuxエキスパートへの道	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.01：システムの起動とLinuxカーネル	システムの起動とLinuxカーネル	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.02：ファイルシステムとストレージ管理	ファイルシステムとストレージ管理	末永 貴一 氏（エスディーテック株式会社）
2.03：ネットワーク構成	ネットワーク構成	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.04：システムの保守と運用管理	リソースの使用状況の把握／死活監視と運用監視ツール	福田 浩之 氏
	「Ansible」による環境構築の自動化	植草 克友 氏（株式会社カサレアル）
2.05：仮想化サーバー	仮想化サーバー	末永 貴一 氏（エスディーテック株式会社）
2.06：コンテナ	コンテナ技術・Dockerの解説（Linux学習）	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
	コンテナの仕組み/Dockerコンテナとコンテナイメージの管理	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
	コンテナ技術について	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）

202試験

主題	セミナー動画	講師
202全般	LinuCレベル2 202試験概要解説 サーバー構築のできるLinuxエキスパートへの道	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.07：ネットワーククライアントの管理	ネットワーククライアントの管理	末永 貴一 氏（エスディーテック株式会社）
2.08：ドメインネームサーバー	DNSの役割を理解しよう！（BIND、ゾーン情報、他）	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.09：HTTPサーバーとプロキシサーバー	サーバ構築ハンズオン Web編(nginx/apache/Squid)	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.10：電子メールサービス	電子メールサービスの仕組み（Postfix/Dovecotの設定方法を理解する）	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.11：ファイル共有サービス	NFSサーバーの設定と管理	末永 貴一 氏（エスディーテック株式会社）
	Sambaの設定と管理	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
	ファイル共有サービス、システムのセキュリティ	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.12：システムのセキュリティ	OpenSSHサーバーの設定と管理	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
	ファイル共有サービス、システムのセキュリティ（一部解説）	鯨井 貴博 氏（株式会社ゼウス・エンタープライズ）
2.13：システムアーキテクチャ	システムアーキテクチャ	濱野 賢一朗 氏

LinuCの学習方法

⑤例題&解説　※レベル1・3・SAもあり
https://linuc.org/study/samples/

LinuC1/2におけるコンテナが含まれる試験範囲

LinuCレベル1(101)

1.01：Linuxのインストールと仮想マシン・コンテナの利用
- 1.01.1Linuxのインストール、起動、接続、切断と停止
- 1.01.2仮想マシン・コンテナの概念と利用
- 1.01.3ブートプロセスとsystemd
- 1.01.4プロセスの生成、監視、終了
- 1.01.5デスクトップ環境の利用

1.02：ファイル・ディレクトリの操作と管理
- 1.02.1ファイルの所有者とパーミッション
- 1.02.2基本的なファイル管理の実行
- 1.02.3ハードリンクとシンボリックリンク
- 1.02.4ファイルの配置と検索

1.03：GNUとUnixのコマンド
- 1.03.1コマンドラインの操作
- 1.03.2フィルタを使ったテキストストリームの処理
- 1.03.3ストリーム、パイプ、リダイレクトの使用
- 1.03.4正規表現を使用したテキストファイルの検索
- 1.03.5エディタを使った基本的なファイル編集の実行

1.04：リポジトリとパッケージ管理
- 1.04.1apt コマンドによるパッケージ管理
- 1.04.2Debianパッケージ管理
- 1.04.3yumコマンドによるパッケージ管理
- 1.04.4RPMパッケージ管理

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
- 1.05.1ハードウェアの基礎知識と設定
- 1.05.2ハードディスクのレイアウトとパーティション
- 1.05.3ファイルシステムの作成と管理、マウント

https://linuc.org/linuc1/range/101.html

LinuCレベル2(201)

2.01：システムの起動とLinuxカーネル
- 2.01.1 ブートプロセスとGRUB
- 2.01.2 システム起動のカスタマイズ
- 2.01.3 Linux カーネルの構成要素
- 2.01.4 Linuxカーネルのコンパイル
- 2.01.5 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

2.02：ファイルシステムとストレージ管理
- 2.02.1 ファイルシステムの設定とマウント
- 2.02.2 ファイルシステムの管理
- 2.02.3 論理ボリュームマネージャの設定と管理

2.03：ネットワーク構成
- 2.03.1 基本的なネットワーク構成
- 2.03.2 高度なネットワーク構成
- 2.03.3 ネットワークの問題解決

2.04：システムの保守と運用管理
- 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
- 2.04.2 バックアップとリストア
- 2.04.3 ユーザへの通知
- 2.04.4 リソース使用状況の把握
- 2.04.5 死活監視、リソース監視、運用監視ツール
- 2.04.6 システム構成ツール

2.05：仮想化サーバー
- 2.05.1 仮想マシンの仕組みとKVM
- 2.05.2 仮想マシンの作成と管理

2.06：コンテナ
- 2.06.1 コンテナの仕組み
- 2.06.2 Dockerコンテナとコンテナイメージの管理

https://linuc.org/linuc2/range/201.html

本日のセミナー対象範囲

主題2.06:コンテナ

2.06.1 コンテナの仕組み

重要度　2

概要　基本的なコンテナの仕組みについて理解している。

詳細

物理マシン、仮想マシン、コンテナの特徴と違いを理解している。

コンテナのファイルシステムとイメージの関係を知っている。

コンテナを実現する技術の概念を知っている。

名前空間, cgroups

2.06.2 Dockerコンテナとコンテナイメージの管理

重要度　3

概要

Dockerを導入してコンテナ実行環境を構築できる。

Dockerコンテナを実行できる。

コンテナイメージを管理できる。

詳細

Dockerを導入して、ネットワークを構成する。

ポート変換, フラットL2ネットワーク

Dockerコンテナを実行して、停止する。

docker ps/stats, docker run/create/restart, docker pause/unpause, docker stop/kill, docker rm

Dockerコンテナに接続してプロセスを実行する。

docker attach, docker exec

コンテナイメージを管理する。

Dockerレジストリ: docker images, docker pull, docker rmi, docker import

Dockerfile: docker build, docker commit

コンテナ/Dockerについて

物理マシン・仮想マシン・コンテナの特徴

物理マシン (Physical Machine)

特徴:
普段利用するPCやサーバーそのもの。CPU、メモリ、ストレージなどの**ハードウェアリソースを専有**して利用します。OSを直接インストールしてアプリケーションを動かすため、最もハードウェア性能を最大限に引き出せます。

メリット:
最高のパフォーマンス: ハードウェアリソースを直接利用するため、最も**高いパフォーマンス**が得られます。
シンプルな構成: 仮想化レイヤーがないため、**構成がシンプル**でトラブルシューティングが比較的容易です。
独立性: 他の環境の影響を一切受けません。

デメリット:
リソースの無駄: 1つのOSしか動かせないため、ハードウェアリソースを十分に使いきれない場合があります。
高いコスト: **ハードウェア購入費用**やメンテナンス費用がかかります。
柔軟性の低さ: サーバーの増減や構成変更に**時間がかかり**、柔軟性に欠けます。
運用負荷: 物理的な設置場所、電源、冷却などの管理が必要です。

物理マシン・仮想マシン・コンテナの特徴

仮想マシン (Virtual Machine – VM)

特徴: ハイパーバイザーと呼ばれるソフトウェアによって、物理マシン上に仮想的なコンピュータ環境を構築する技術です**1台の物理マシン上で複数のゲストOSを同時に実行** できます。それぞれのVMは、あたかも独立した物理マシンであるかのように振る舞います。

メリット:
リソースの有効活用: 1台の物理マシンで複数のVMを動かすことで、**ハードウェアリソースを有効活用** できます。
迅速な展開: **新しいVMの作成や複製が容易** で、環境構築にかかる時間を短縮できます。
高い柔軟性: 必要に応じて**VMの増減やリソースの割り当て変更が容易** です。
環境の分離: 各VMは独立しているため、**あるVMでの問題が他のVMに影響を与えることはありません** 。
災害対策: スナップショット機能やVMの移行機能により、**バックアップや障害回復が容易** になります。

デメリット:
パフォーマンスのオーバーヘッド: ハイパーバイザーを介するため、**物理マシンに比べて若干のパフォーマンスロスが発生** します。
リソースの浪費: 各VMには独自のOSが必要なため、その分の**ディスク容量やメモリが消費** されます。
管理の複雑さ: ハイパーバイザーや各VMの管理が必要になり、物理マシンのみの場合より**管理が複雑**になることがあります。

物理マシン・仮想マシン・コンテナの特徴

コンテナ（Container）

特徴: 仮想マシンよりも**さらに軽量な**仮想化技術です。**OSを共有**し、アプリケーションとその実行に必要なライブラリや依存関係を**コンテナイメージという形でパッケージ化**し、分離された環境で実行します。Docker/Kubernetesなど。

メリット:
非常に軽量: OSを共有するため、VMに比べて**起動が速く、ディスク容量やメモリ消費量が少ない**です。
高いポータビリティ: コンテナイメージを使い、開発環境からテスト環境、本番環境への**移行が非常にスムーズ**です。
一貫性: 開発環境と本番環境で**同じコンテナイメージを利用可能**。
リソース効率の向上: 1つのOS上で多数のコンテナを実行できるため、**リソースを最大限に活用**できます。
迅速なスケーリング: 必要に応じて、**瞬時にコンテナを起動・停止**できます。

デメリット:
OSの共有: **セキュリティ上の懸念**が皆無ではありません。
永続データの扱いの複雑さ: 使い捨ての設計思想なので、**永続的データを扱う為にストレージを考慮する必要**があります。
学習コスト: VMや物理マシンに比べて新しい概念が多く、**学習に時間がかかる**場合があります。

物理マシン・仮想マシン・コンテナの特徴

	物理マシン	仮想マシン (VM)	コンテナ
独立性	最高 (ハードウェア専有)	高 (ゲストOSが独立)	中 (ホストOSを共有)
リソース効率	低 (余剰リソース発生しやすい)	中 (OS分のオーバーヘッド)	高 (OS共有で効率的)
起動速度	遅い (OS起動)	中 (ゲストOS起動)	速い (プロセス起動)
ポータビリティ	低 (ハードウェア依存)	中 (VMファイルとして移行可能)	高 (どこでも同じように動作)
管理の複雑さ	低 (シンプルだが物理管理が必要)	中 (ハイパーバイザーとVMの管理)	高 (コンテナオーケストレーションなど)
用途	高性能が必要なシステム、大規模 DB	サーバー統合、開発・検証環境、柔軟な環境	マイクロサービス、CI/CD、ポータブルな環境
コスト	高	中	低

コンテナを支える技術 (名前空間、namespaces)

Linuxカーネルの機能の一つ で、プロセス、ネットワーク、ファイルシステムなどの**システムリソースを論理的に隔離（分離）** するための技術です。これにより、**各コンテナが他のコンテナやホストOSのプロセスから独立** しているかのように振る舞うことができます。コンテナが軽量な仮想化を実現できる核となる技術の一つです。

名前空間がない場合、全コンテナがホストOSの同じリソースを共有することになり、
以下のような問題が発生します。

- 競合: あるコンテナが使用しているポート番号を、別のコンテナも使用しようとする
- セキュリティリスク: あるコンテナがホストOSのファイルシステムに自由にアクセスできたり、他のコンテナのプロセスを停止させたりできる。
- 再現性の欠如: 環境によってアプリケーションの動作が変わってしまう。

コンテナを支える技術 (名前空間、namespaces)

1. PID 名前空間 (PID namespace)

役割: **プロセス**ID (PID) を隔離します。

説明: 各コンテナは独自のPID空間を持ち、コンテナ内で実行されるプロセスは、そのコンテナ内のPID空間で1から番号が振られます。これにより、コンテナ内のプロセスはホストOSや他のコンテナのプロセスを直接認識したり、影響を与えたりすることができません。コンテナ内で $ ps aux コマンドを実行しても、そのコンテナ内のプロセスしか表示されないのはこのためです。

2. ネットワーク名前空間 (Network namespace)

役割: **ネットワークインターフェース、IPアドレス、ルーティングテーブル、ポート番号** などを隔離します。

説明: 各コンテナは独自のネットワークスタックを持つことができます。これにより、それぞれのコンテナが独自のIPアドレスを持ち、他のコンテナとポートの競合を起こさずにサービスを提供できます。

3. マウント名前空間 (Mount namespace)

役割: **ファイルシステム** のマウントポイントを隔離します。

説明: 各コンテナは独自のファイルシステムビューを持ちます。これにより、あるコンテナがファイルシステムを変更しても、ホストOSや他のコンテナのファイルシステムには影響を与えません。コンテナ内のルートディレクトリ (/) は、そのコンテナ専用の独立したファイルシステムとなります。

コンテナを支える技術 (名前空間、namespaces)

4. UTS 名前空間 (UTS namespace)
役割: **ホスト名とドメイン名** を隔離します。
説明: 各コンテナは独自のホスト名を持つことができます。コンテナ内で $ hostname コマンドを実行すると、そのコンテナに割り当てられたホスト名が表示されます。

5. IPC 名前空間 (IPC namespace)
役割: **プロセス間通信 (Inter-Process Communication - IPC) リソース(セマフォ、メッセージキュー、共有メモリなど)** を隔離します。
説明: コンテナ内のプロセスは、そのコンテナ内のIPCリソースのみを利用でき、ホストOSや他のコンテナのIPCリソースとは分離されます。

6. ユーザー名前空間 (User namespace)
役割: **ユーザーID (UID) とグループID (GID)** を隔離します。
説明: コンテナ内で root ユーザーとして実行されているプロセスが、ホストOS上では権限の低いユーザーとしてマッピングされるように設定できます。これにより、コンテナ内での root 権限の悪用によるホストOSへの影響を限定し、セキュリティを向上させます。

コンテナを支える技術 (cgroups)

Linuxカーネルの機能の一つで、**プロセスやプロセスのグループが利用できるCPU、メモリ、ディスクI/O、ネットワークなどのシステムリソースを制限、優先順位付け、監視**　、管理するためのメカニズムです。名前空間 (namespaces) がリソースの隔離を提供するのに対し、cgroupsはリソースの制限・管理を提供します。

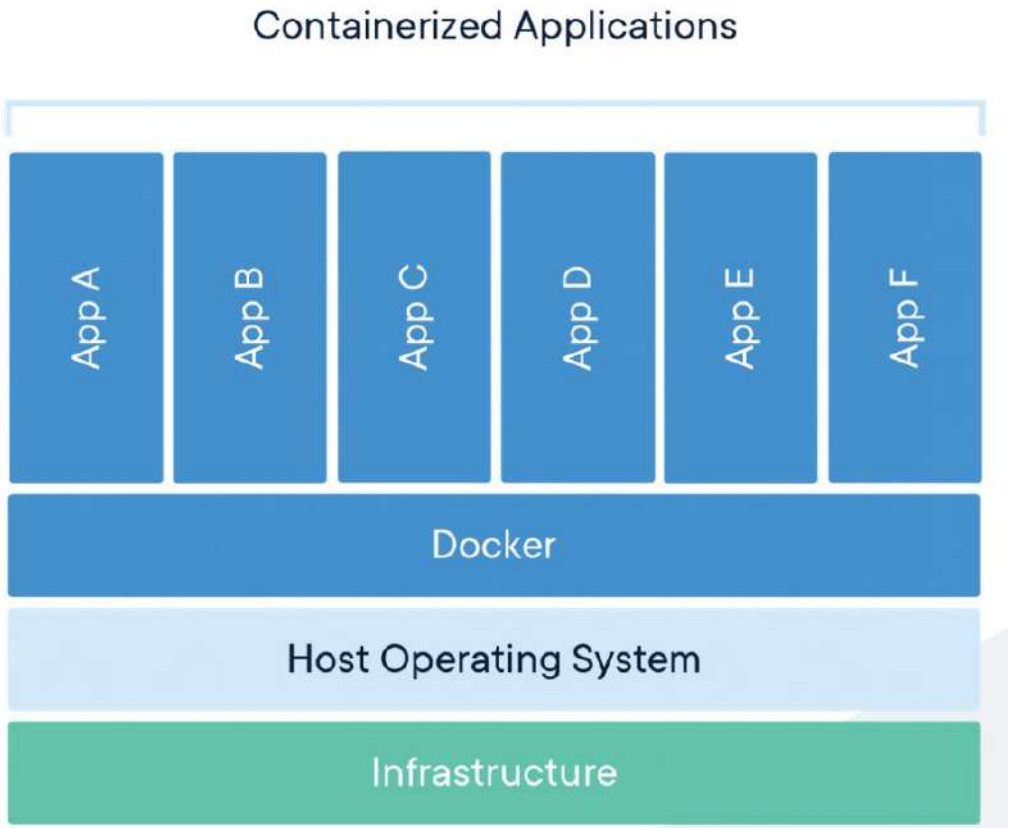
以下の問題への対応の為、必要となる。

リソース枯渇: 特定のコンテナがCPUやメモリを大量に消費し、ホストOSや他のコンテナの動作に悪影響を与える。
パフォーマンスの不安定化: リソースの利用状況がコンテナ間で変動し、安定したサービス提供が難しくなる。
公平性の欠如: 重要なアプリケーションが利用できるリソースが、それほど重要ではないアプリケーションによって奪われる。

コンテナ/Dockerについて

Docker

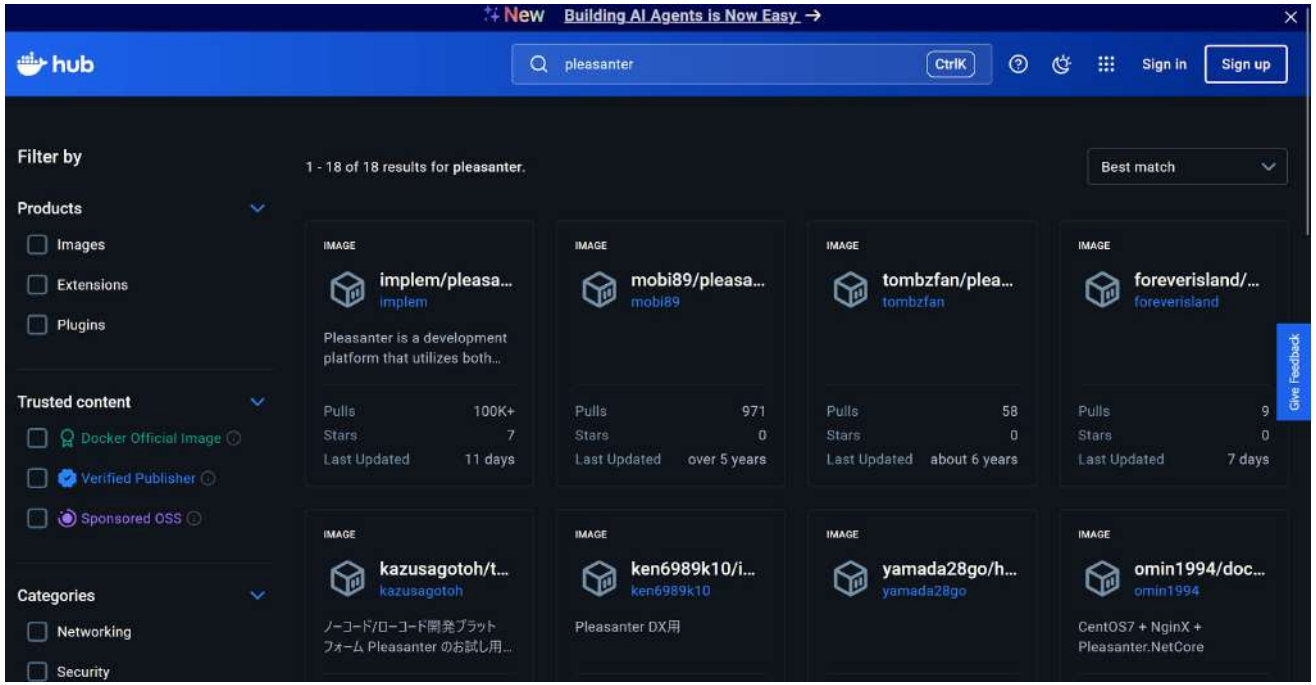
コンテナ管理ソフトウェア(コンテナエンジン)。

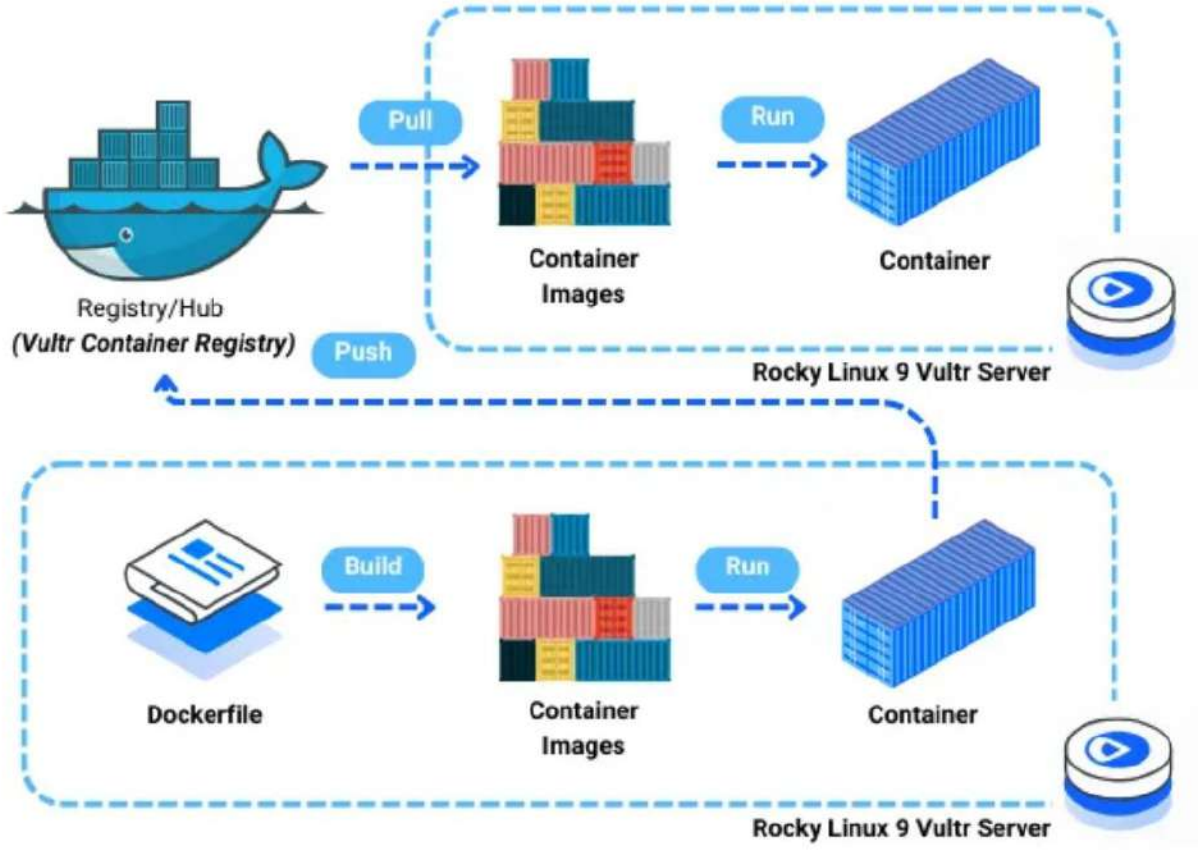


https://www.docker.com/
https://docs.docker.com/
https://www.docker.com/resources/what-container/

コンテナ/Dockerについて

Dockerにおけるコンテナ起動の流れ

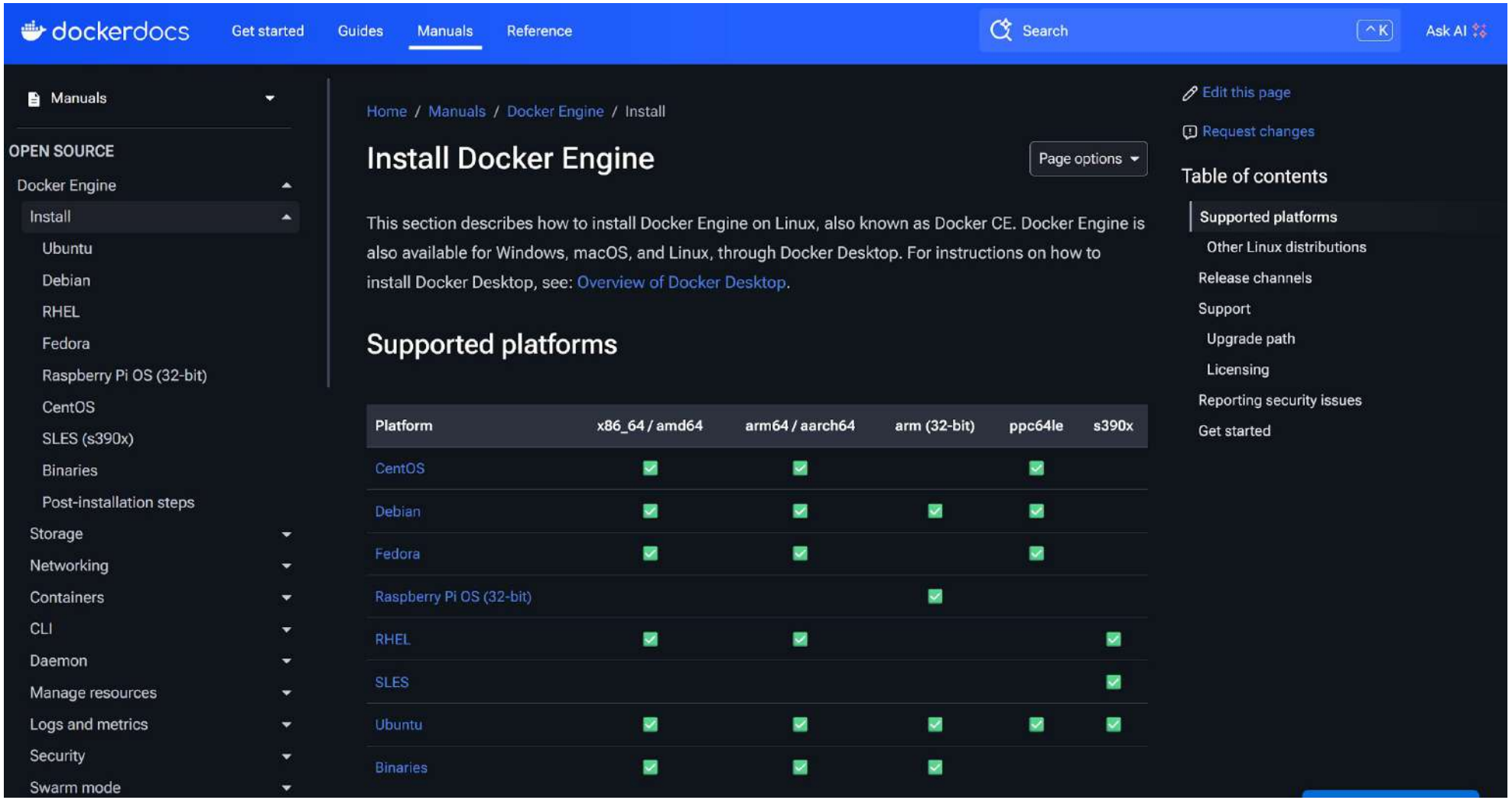




①DockerHub(コンテナイメージ配布場所)からダウンロード
②ローカル環境にコンテナイメージ保存
③コンテナイメージからコンテナ起動

https://hub.docker.com/
https://docs.vultr.com/how-to-install-docker-on-rocky-linux-9

Dockerのインストール (環境の用意)



https://docs.docker.com/engine/install/

Dockerのインストール (環境の用意) ※Ubuntu 24.04の場合

$ for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done

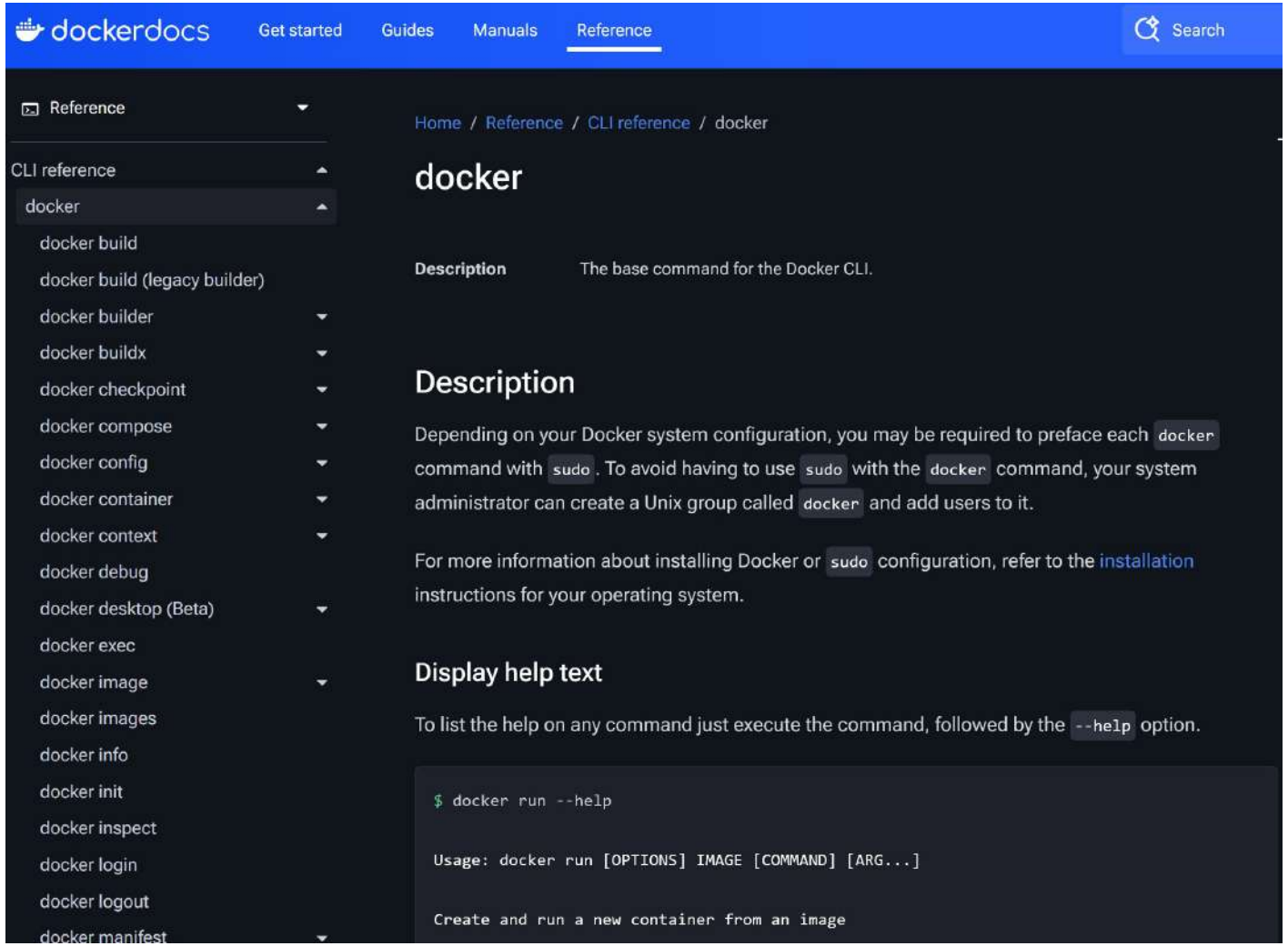
```
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl
$ sudo install -m 0755 -d /etc/apt/keyrings
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
$ sudo chmod a+r /etc/apt/keyrings/docker.asc

$ echo ¥
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu ¥
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | ¥
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

https://docs.docker.com/engine/install/

Dockerで利用するコマンド



https://docs.docker.com/reference/cli/docker/

Dockerで利用する主なコマンド

docker image pull(コンテナイメージの取得)
docker images(ローカル環境に保存されているコンテナイメージの一覧表示)
docker rmi(ローカル環境に保存されているコンテナイメージの削除)
docker run(コンテナの起動)
docker stop(コンテナの停止)
docker rm(コンテナの削除)
docker ps(起動しているコンテナの一覧表示)
docker exec(指定コンテナでのコマンド実施)

https://www.docker.com/
https://docs.docker.com/

docker image pull(コンテナイメージの取得)

```
$ docker image pull debian

Using default tag: latest
latest: Pulling from library/debian
e756f3fdd6a3: Pull complete
Digest: sha256:3f1d6c17773a45c97bd8f158d665c9709d7b29ed7917ac934086ad96f92e4510
Status: Downloaded newer image for debian:latest
docker.io/library/debian:latest
```

https://docs.docker.com/reference/cli/docker/image/pull/

docker images(ローカル環境に保存されているコンテナイメージの一覧表示　)

```
$ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              77af4d6b9913        19 hours ago        1.089 GB
committ             latest              b6fa739cedf5        19 hours ago        1.089 GB
<none>              <none>              78a85c484f71        19 hours ago        1.089 GB
docker              latest              30557a29d5ab        20 hours ago        1.089 GB
<none>              <none>              5ed6274db6ce        24 hours ago        1.089 GB
postgres            9                   746b819f315e        4 days ago          213.4 MB
postgres            9.3                 746b819f315e        4 days ago          213.4 MB
postgres            9.3.5               746b819f315e        4 days ago          213.4 MB
postgres            latest              746b819f315e        4 days ago          213.4 MB
```

https://docs.docker.com/reference/cli/docker/image/ls/

docker rmi(ローカル環境に保存されているコンテナイメージの削除　)

```
$ docker images

REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
test1                         latest              fd484f19954f        23 seconds ago      7 B (vir
test                          latest              fd484f19954f        23 seconds ago      7 B (vir
test2                         latest              fd484f19954f        23 seconds ago      7 B (vir

$ docker rmi fd484f19954f

Error: Conflict, cannot delete image fd484f19954f because it is tagged in multiple repositorie
2013/12/11 05:47:16 Error: failed to remove one or more images
```

https://docs.docker.com/reference/cli/docker/image/rm/

docker run(コンテナの起動)

```
$ docker run --name test -d nginx:alpine
4bed76d3ad428b889c56c1ecc2bf2ed95cb08256db22dc5ef5863e1d03252a19
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
4bed76d3ad42   nginx:alpine   "/docker-entrypoint.…"   1 second ago   Up Less than a second
```

```
$ docker run -t -i --privileged ubuntu bash
root@50e3f57e16e6:/# mount -t tmpfs none /mnt
root@50e3f57e16e6:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
none            1.9G     0  1.9G   0% /mnt
```

https://docs.docker.com/reference/cli/docker/container/run/

docker run(コンテナの起動)

```
$ docker stop my_container
```

docker rm(コンテナの削除)

```
$ docker rm /redis

/redis
```

https://docs.docker.com/reference/cli/docker/container/stop/
https://docs.docker.com/reference/cli/docker/container/rm/

docker ps(起動しているコンテナの一覧表示)

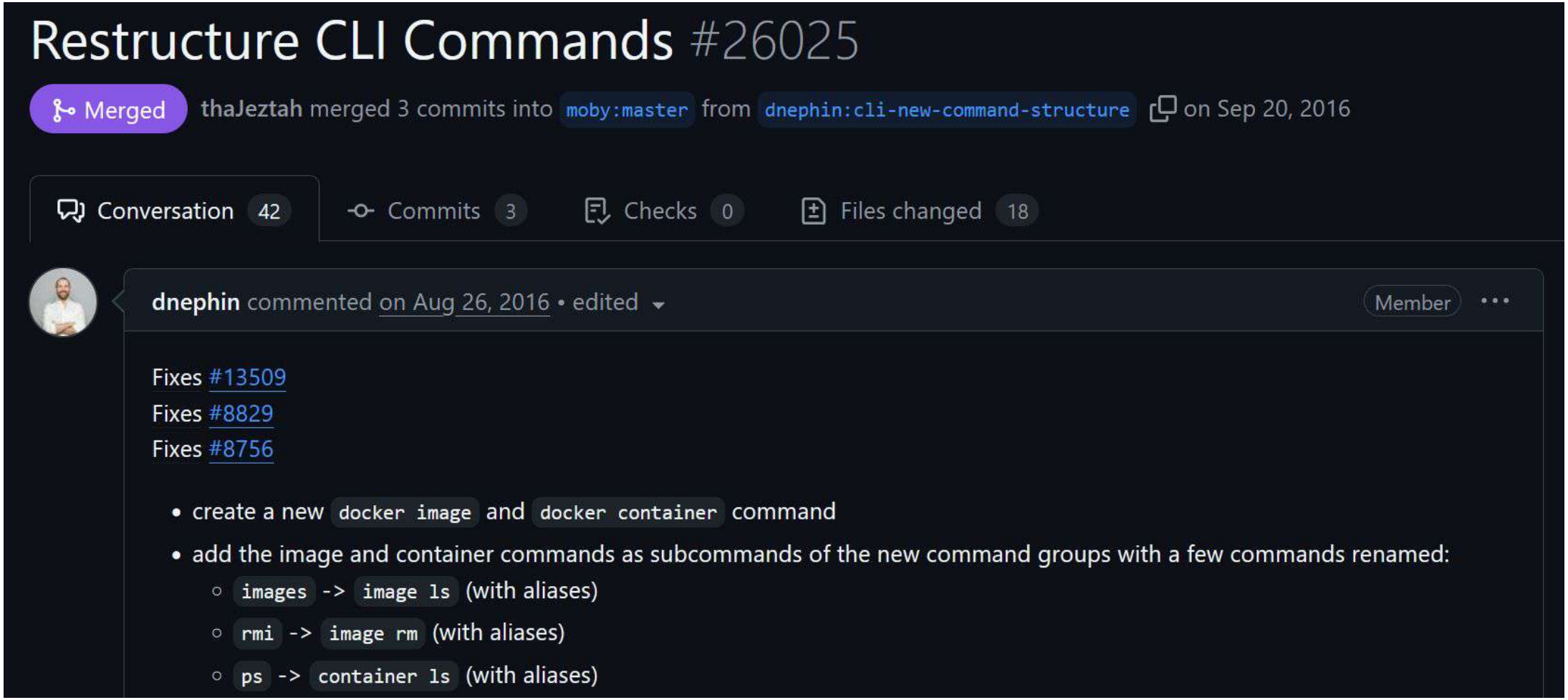
```
$ docker ps -a
```

docker exec(指定コンテナでのコマンド実施)

```
$ docker run --name mycontainer -d -i -t alpine /bin/sh
```

https://docs.docker.com/reference/cli/docker/container/ls/
https://docs.docker.com/reference/cli/docker/container/exec/

docker v1.13からのコマンド体系変更　※ 2017年1月18日リリース



Restructure CLI Commands #26025

Merged thaJeztah merged 3 commits into moby:master from dnephin:cli-new-command-structure on Sep 20, 2016

Conversation 42 Commits 3 Checks 0 Files changed 18

dnephin commented on Aug 26, 2016 • edited Member

Fixes #13509
Fixes #8829
Fixes #8756

- create a new `docker image` and `docker container` command
- add the image and container commands as subcommands of the new command groups with a few commands renamed:
  - `images` -> `image ls` (with aliases)
  - `rmi` -> `image rm` (with aliases)
  - `ps` -> `container ls` (with aliases)

https://docs.docker.com/engine/release-notes/prior-releases/#1130-2017-01-18
https://github.com/moby/moby/pull/26025

Dockerfile

Dockerイメージを自動的に構築するための設定を記載したテキストファイル。
再現性、自動化、バージョン管理、透明性などのメリットがある。

```
# ベースイメージを指定（必須）
FROM ubuntu:latest

# メタデータ（オプション）
LABEL maintainer="your_name <your_email@example.com>"
WORKDIR /app

# 依存関係のインストール
RUN apt-get update && apt-get install -y nginx

# アプリケーションコードのコピー
COPY . /app

# ポートの公開（オプション）
EXPOSE 80

# コンテナ起動時に実行するコマンド
CMD ["nginx", "-g", "daemon off;"]
```

https://docs.docker.com/build/concepts/dockerfile/
https://docs.docker.com/reference/dockerfile/

Dockerfile

Dockerイメージを自動的に構築するための設定を記載したテキストファイル。
再現性、自動化、バージョン管理、透明性などのメリットがある。

```
$ docker build -t test:latest .
```

https://docs.docker.com/build/concepts/dockerfile/
https://docs.docker.com/reference/dockerfile/

例題と解説

例題1

Overlay Filesystemについて、正しいものはどれか。

1.ジャーナルと呼ばれるデータを記録するファイルシステムである
2.プロセスなどのリソースの論理的な分離を行うLinuxカーネルの機能である
3.イメージのレイヤーを重ねて構成されるファイルシステムである
4.物理マシン・仮想マシン・コンテナで共通して利用されるファイルシステムである

解説1

正解：3.イメージのレイヤーを重ねて構成されるファイルシステムである

Overlay Filesystemはコンテナ環境においてコンテナイメージを構成するために使われるファイルシステムです。土台となるベースイメージの上にいくつかのレイヤーを追加し、目的に合わせたイメージを作り上げます。

例えばnginx(Webサーバ)のイメージを作る場合、以下のようになります。

レイヤ4：サーバの起動(systemctl start nginx)
レイヤ3：コンテンツファイルの配置(vi index.html)
レイヤ2：設定ファイルの変更(vi nginx.conf)
レイヤ1：パッケージのインストール(apt install nginx)
ベースレイヤ：ベースイメージ(公開されているUbuntuなど)

ベースイメージはDockerHubなどのコンテナレポジトリから取得し、レイヤ1以上を追加し、自チーム内の共通イメージと利用するなどします。

なお、Overlay Filesystemの詳細については、以下で確認ができます。
https://docs.kernel.org/filesystems/overlayfs.html

https://linuc.org/study/samples/6488/

例題2

以下のうち、「コンテナ」について説明しているものを選択してください。

1.物理的なハードウェアで動作するオペレーティングシステムでアプリケーションを実行します。
2.ハイパーバイザーと呼ばれるソフトウェアを使用して、仮想的な空間でオペレーティングシステムを実行します。
3.オペレーティングシステムレベルでアプリケーションを分離して実行します。
4.仮想的なデスクトップ環境を提供し、リモートデスクトップクライアントから接続します。

解説2

正解：3.オペレーティングシステムレベルでアプリケーションを分離して実行します。

コンテナとは、仮想的な隔離されたユーザ空間を作成し、その中でアプリケーションを実行するための技術です。カーネル空間はオペレーティングシステムと共有しますが、アプリケーションが実行される空間はコンテナ毎に作成されます。

このため、コンテナで動作するアプリケーションを、分離された個別の環境で実行させることができます。

コンテナにはアプリケーションと必要なライブラリだけが含まれており、オペレーティングシステムが含まれていません。オペレーティングシステムを含んでいる仮想マシンと比較すると、イメージファイルのサイズが小さくなります。

また、コンテナ実行時には、ホストOSとカーネル空間を共有しますので、実行サイズが小さくなります。イメージサイズ、実行サイズともに、少ないリソースで動作することができますので、仮想マシンより多くのコンテナを実行することができます。

また、ソフトウェアのインストールや設定が完了している環境をコンテナイメージとして提供することができます。このため、システム構築の工程を短縮して、素早くアプリケーションを起動することができます。

この特徴を活かして、開発環境、テスト環境の構築やアプリケーションのリリース等にコンテナを利用することができます。

https://linuc.org/study/samples/4119/

解説2

正解：3.オペレーティングシステムレベルでアプリケーションを分離して実行します。

1．物理的なハードウェアで動作するオペレーティングシステムでアプリケーションを実行します。
これは、物理マシンの説明です。

2．ハイパーバイザーと呼ばれるソフトウェアを使用して、仮想的な空間でオペレーティングシステムを実行します。
これは、仮想マシンの説明です。

4．仮想的なデスクトップ環境を提供し、リモートデスクトップクライアントから接続します。
これは、VDI(仮想デスクトップ環境)の説明です。

https://linuc.org/study/samples/4119/

例題3

Dockerのコンテナイメージの管理を行っている。不要となったコンテナイメージの削除に使用するコマンドはどれか。

1.docker images

2.docker rmi

3.docker run

4.docker rm

解説3

正解：2.docker rmi

Dockerは、Docker Hubなどのコンテナレポジトリからコンテナイメージを取得し、それをベースにコンテナを作成します。
選択肢の各コマンドは、以下の意味を持ちます。

docker images：コンテナイメージを一覧表示する
docker rmi：コンテナイメージを削除する
docker run：指定されたコンテナイメージ上に書き込み可能なコンテナ・レイヤを作成する
docker rm：コンテナを削除する

以下、コンテナイメージ削除の操作例です。
※コンテナイメージの一覧表示をし、その後指定したコンテナイメージを削除している

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
test                latest              fd484f19954f        23 seconds ago      7 B (virtual 4.964 MB)
$ docker rmi test
Untagged: test:latest
Deleted: fd484f19954f4920da7ff372b5067f5b7ddb2fd3830cecd17b96ea9e286ba5b8
```

https://linuc.org/study/samples/6659/

例題4

以下のようにコンテナが動作しているとき、コンテナの停止が出来ないものはどれか。

```
[root@c6bae784f333 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
7bff0801444c        nginx               "/docker-entrypoint.…"   4 seconds ago       Up 3 seconds        0.0.0.0:8080->80/tcp     nginx001
```

1.docker stop 7bff0801444c

2.docker stop 7b

3.docker stop nginx001

4.docker stop nginx

解説4

正解：4.docker stop nginx

その他選択肢は、コンテナの停止が可能です。

具体的な操作例で確認してみましょう。

例題5

Dockerコマンドの説明のうち、正しいものを2つ選択せよ。

1.docker images は、コンテナイメージを取得する
2.docker pull は、コンテナイメージを取得する
3.docker rmi は、コンテナを削除する
4.docker exec は、稼働中のコンテナでコマンドを実行する

解説5

正解：2.docker pull は、コンテナイメージを取得する
　　4.docker exec は、稼働中のコンテナでコマンドを実行する

不正解の選択肢（1と3）は、正しくは以下となります。
・docker images は、取得したコンテナイメージの一覧表示を行う
・docker rmi は、コンテナイメージを削除する

Dockerコンテナは、基本的には以下のステップで利用します。
①コンテナイメージの取得
②コンテナの起動
③コンテナでのコマンドやプロセス実行
④コンテナの終了
⑤コンテナの削除 ※必要に応じて
⑥コンテナイメージの削除　※必要に応じて

https://linuc.org/study/samples/1309/

例題6

ダウンロードしたDockerコンテナイメージを削除する方法として正しいものをふたつ選択してください。

1.docker stop 64c383ce87ae

2.docker rmi −f 64c383ce87ae

3.docker rm −f 64c383ce87ae

4.docker image rm −f 64c383ce87ae

64c383ce87aeはIDを示しています。

解説6

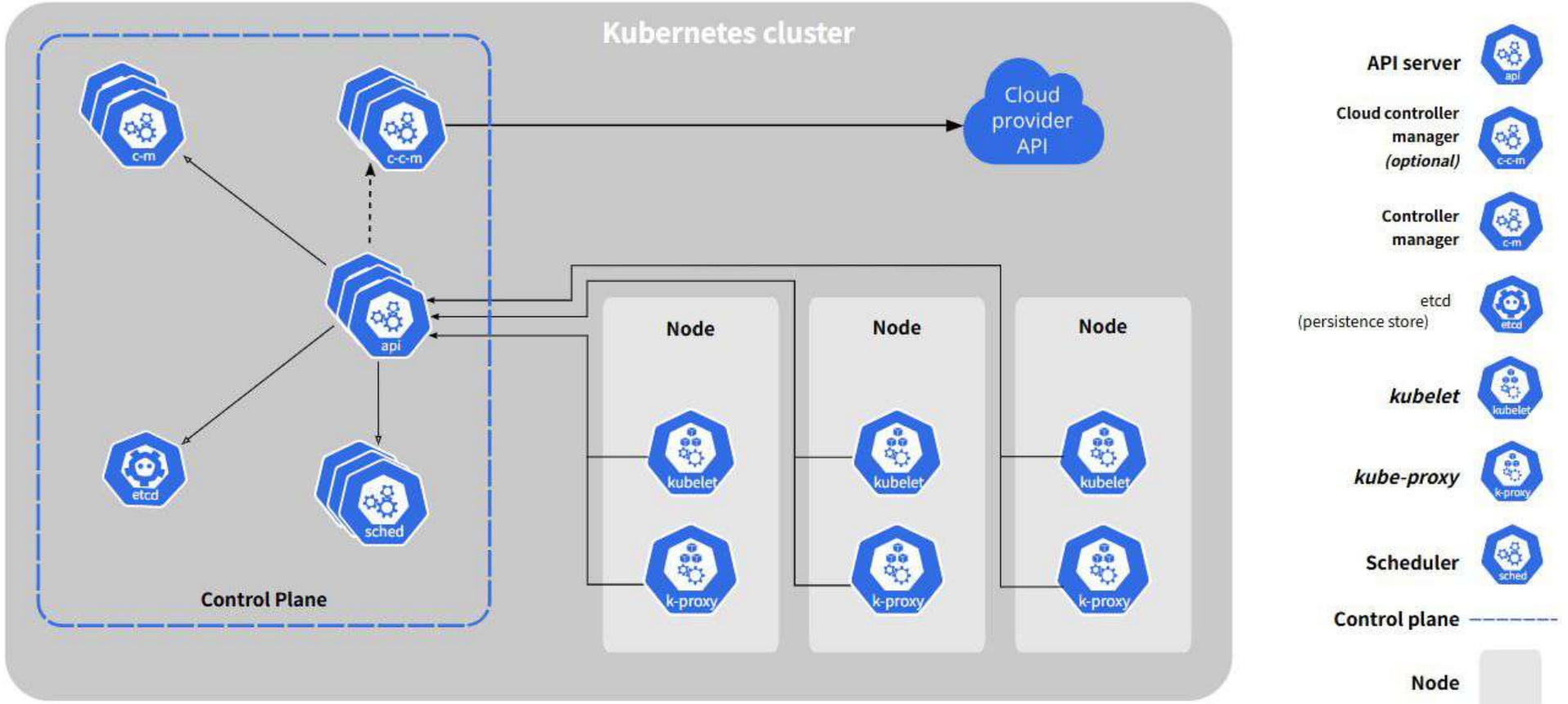
正解：2.docker rmi -f 64c383ce87ae

4.docker image rm -f 64c383ce87ae

おまけ

Kubernetes

本格的にコンテナを運用する場合、Kubernetesという選択肢もある。



https://kubernetes.io/ja/docs/concepts/overview/components/

振り返り

Linux研修オンライン【例題で学ぼう】 ※再掲

アジェンダ
- LinuC、およびその学習方法
- コンテナ/Dockerについて
- 例題と解説

本日のゴール
- コンテナの概念の理解
- Docker環境の操作方法の理解
- LinuCレベル2資格対策の理解

ご清聴ありがとうございました！