令和7年度
「専門職業人材の最新技能アップデートのための専修学校リカレント教育（リ・スキリング）推進事業」

指導者向け研修プログラム資料

情報技術者の技能アップデートのためのリカレント教育推進事業

目次

仮想化・コンテナ・クラウドネイティブ教材　ー指導方法概論ー……1

セクション1　技術パラダイムシフト……2

セクション2　指導者の役割とは……4

セクション3　技術の本質を理解する……7

セクション4　授業の組み立て方……10

セクション5　指導者の学習戦略……13

仮想化技術　ー指導者向け資料ー……16

コンテナ技術基礎　ー指導者向け資料ー……24

クラウドネイティブ概論　ー指導者向け資料ー……33

コンテナ技術システム構築　コンテナ技術システム構築……40

AWSクラウドネイティブ基盤実装　ー指導者向け資料ー……47

コンテナ・クラウドサービスのセキュリティ ー指導者向け資料ー……54

令和7年度文部科学省委託

「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」

仮想化・コンテナ・クラウドネイティブ教材

― 指導方法概論 ―

情報技術者の技能アップデートのためのリカレント教育推進事業

リカレント教育推進　指導方法概論

このVODで学習すること

新しい技術の本質を体系的にご理解いただき、
カリキュラムを扱う際の心構えやあり方についてお話いたします。

セクション構成

1. **技術パラダイムシフトの理解**
   なぜクラウドネイティブ教育が必要か / 20年の技術進化 / 学習ロードマップ
2. **指導者の役割**
   役割および注意するポイント / マインドセット
3. **技術の本質理解**
   原理原則 / トレードオフ / 伝え方
4. **授業の組み立て方**
   カリキュラムの進め方 / 演習
5. **指導者の学習戦略**
   指導者自身の学習戦略 / クロージング

セクション１
技術パラダイムシフト

リカレント教育推進　指導方法概論

なぜクラウドネイティブを伝える必要があるのか？

指導者の皆様がクラウドネイティブなどの新しい技術を**意欲的に**学び、受講者に**効果的に伝える**こと**で、次世代の人材が育ち、日本が直面する**社会課題解決**の貢献**に繋がり**ます。

危機① 2025年の崖
最大12兆円
年間経済損失リスク

- レガシーシステムの老朽化
- 保守・運用を担う人材不足
- 技術的負債の蓄積

危機② DX人材不足
約79万人
2030年までの不足予測

- 急速な技術進化への対応困難
- 実務経験者の育成システム不足
- 企業の競争力低下

危機③ スキルギャップ
クラウドネイティブ
専門人材の深刻な不足

- コンテナ技術の理解不足
- DevOps文化の浸透遅れ
- セキュリティ専門家の不在

リカレント教育の役割

- 実務経験者が「技術の本質」を伝える
- 次世代の即戦力人材を育成
- 学び直しの機会を提供し、スキルギャップを解消

あなたの貢献

開発経験 × 指導力　＝
日本のDX推進を支える人材育成

経験を伝えることが価値になる

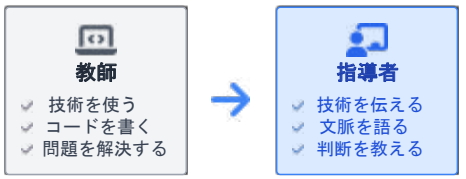
クラウドネイティブやコンテナは比較的新しい技術ですが、単に技術を教えるだけでなく、これまでの開発経験があれば、その含めたストーリーを伝えることで、受講者の共感と理解に繋がります。

「変化の意味」を説明できる
技術の表面だけでなく、なぜその技術が生まれたのか、
どんな課題を解決するために登場したのかを語れる

学習者が最も欲しいのは「文脈」
技術仕様は公式ドキュメントで学べますが、
「いつ使うべきか」「なぜこの選択か」は経験者からしか学べない

トレードオフを理解している
現場で直面した課題、意思決定の背景、失敗から学んだ教訓。
これらの「生きた知識」が指導の価値を生む

教師から指導者へ

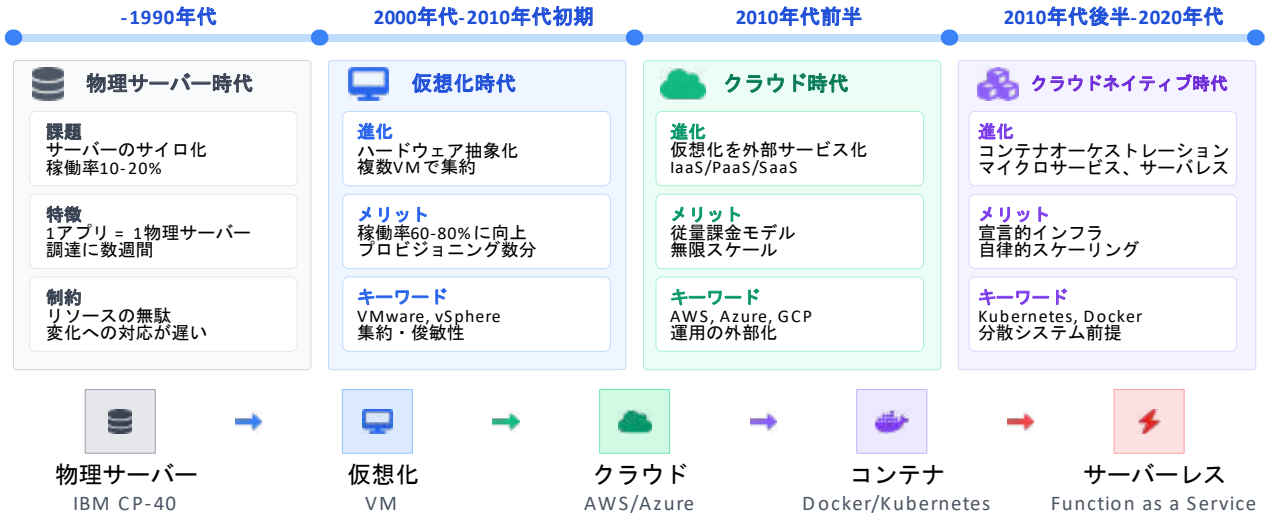


あなただけの価値
現場で培った「なぜ」と「いつ」の知見は、
ドキュメントにない最も価値ある教材

リカレント教育推進　指導方法概論

技術のパラダイムシフト

各時代のビジネス課題を解決するために、物理サーバー、仮想化、クラウドネイティブと段階的に技術発展してきました。技術発展の必要性を抑えることが大切です。



VOD学習ロードマップ

仮想化、コンテナ、クラウドネイティブに関する幅広い領域を取り扱う形になります。
全6つの単元で構成されており、入門、基本、応用・実践の順で学習することを推奨します。

分類	入門	基本	応用・実践
知識	仮想化技術	クラウド ネイティブ概論	
技術習得	コンテナ 技術基礎	コンテナ技術 システム開発	AWSクラウドネイティブ基盤実装 コンテナ・クラウド セキュリティ

1. **仮想化技術**
   基盤技術の理解・なぜ仮想化か？
2. **コンテナ技術基礎**
   軽量なコンテナ実行環境とは？
3. **クラウドネイティブ概論**
   クラウドネイティブの思想を理解
4. **コンテナ技術システム開発**
   Docker、kubernetesの開発技法
5. **AWSクラウドネイティブ基盤構築**
   AWSでの実践的なシステム構築
6. **コンテナ・クラウドセキュリティ**
   コンテナ・クラウド環境の安全性確保

セクション２
指導者の役割とは

技術指導にあたり気を付けるべきこと

クラウドネイティブな技術は変化が速く、すべてを網羅することは難しいです。
「知識を教える」というより、新規領域への興味付けを行い、学習方法、調べ方、考え方を伝え、
「受講者が自ら学習していける力を育む」ことが大切です。

完璧主義の罠
完璧を目指す → 教える範囲が無限に拡大
✓ 細部に拘りすぎない

知識伝達型の罠
一方的に教える → 暗記が目的化
✓ 考え方や調査方法を伝え、学習を促す

最新技術追従の罠
トレンドだけ追う → 本質を見失う
✓ 最新技術の背景や本質も伝える

共通の解決策
「何を教えるか」より
「どう学ばせるか」を設計する

リカレント教育推進　指導方法概論

指導する際のポイント

指導される際には、技術・背景の説明、学習方法、興味付け、といったことを踏まえと良いでしょう。

「なぜ」を説明する力
目的・背景・トレードオフで意思決定を可視化する。
技術選択の理由、制約条件、他の選択肢との比較を語れる。

学習プロセスをデザインする力
調べ方→小さく試す→検証→振り返りのサイクルを設計。
「何を学ぶか」だけでなく「どう学ぶか」を教える。

技術的好奇心をモデリングする力
共に学ぶ姿勢と不確実性の扱い方を示す。
「わからない」を恐れず、探究する姿勢を見せる。



リカレント教育推進　指導方法概論

マインドセット

指導者自身が完璧である必要はありません。
「知らないこと・分からないこと」も一緒に考えながら、共に技術力を深めていくことが大切です。

わからない時：一緒に調べる

答えがわからない時こそ、調べ方を言語化する絶好の機会。
公式ドキュメントの読み方、信頼できる情報源の選び方を実演する。

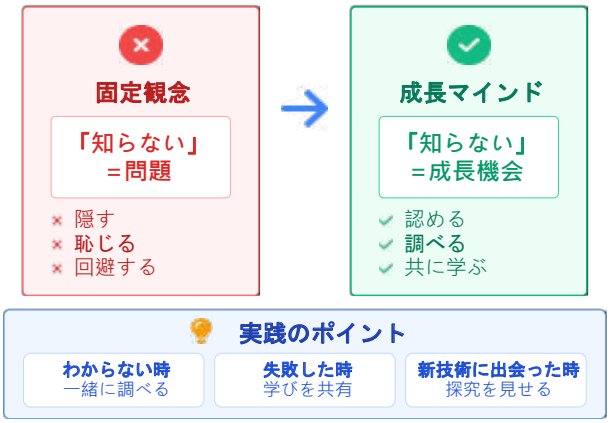
毎回の振り返りを共有

「何がうまくいき、次は何を試すか」を可視化。
指導者自身の学習プロセスを見せることで、学習者の不安を軽減する。

失敗の見せ方

小さく試し、小さく失敗し、学びを可視化する。
「失敗=恥」ではなく「失敗=学習機会」という文化を作る。

マインドセットの転換



リカレント教育推進　指導方法概論

教師ではなくファシリテーター

各単元の技術と解答を教えるだけでなく、新しい技術を体得していくプロセスを支援することが大切です。

教師中心型

知識の伝達者

- 説明 → 暗記
  講義形式で一方的に知識を伝達
- 答え → 正解
  教師が持つ正解を学習者に提示
- 評価 → テスト
  知識の記憶量を測定
- 学習者 → 受動的
  聞く・覚える・再現する

ファシリテーター型

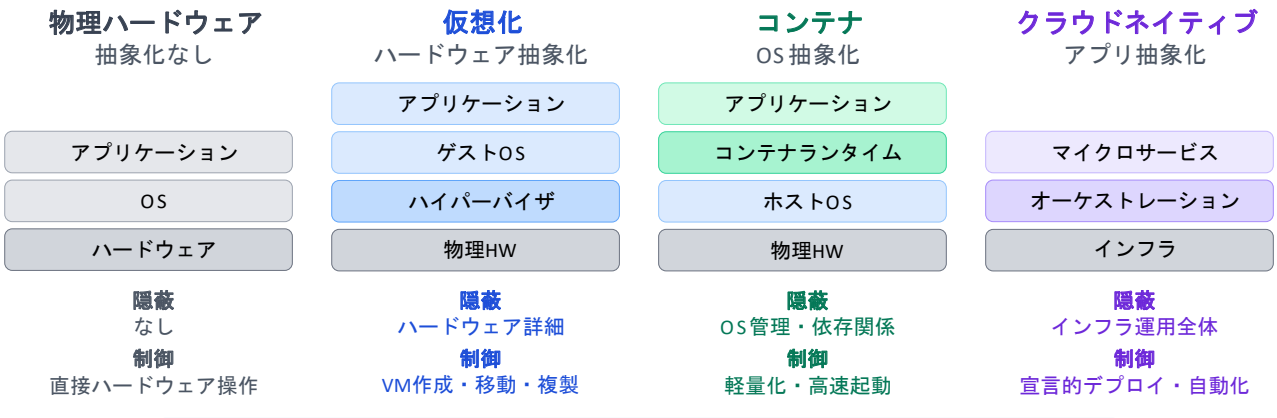
学びの設計者・伴走者

- 問い → 探究
  オープンクエスチョンで思考を促す
- 支援 → フィードバック
  学習プロセスをサポート
- 評価 → 成果物とプロセス
  理解の深さと思考過程を評価
- 学習者 → 能動的
  問う・試す・振り返る

セクション3
技術の本質を理解する

共通原理：抽象化の進化

仮想化技術は連続していて、段階的に抽象化が進んできました。
技術発展の過程で何を隠蔽し、何を制御可能にしたのかを抑えておきましょう。



抽象化の方向
物理制御 → ハードウェア抽象化 → OS抽象化 → アプリケーション抽象化

共通原理：宣言的アプローチ

宣言的アプローチは現代インフラに適した設計思想です。
命令的アプローチと使い分けて利用できるようにしましょう。

2つのアプローチの違い

命令的アプローチ
「どうやるか」順番を指示
→ スクリプト、コマンド手順、Ansible

宣言的アプローチ
「何が欲しいか」状態を指定
→ Kubernetes YAML、Terraform

なぜ宣言的アプローチが現代インフラに適しているか

冪等性
何度実行しても同じ結果

再現性
同じ構成を確実に再現

自動調整
現状と望む状態の差を自動修正

差分適用
変更箇所だけを更新

リカレント教育推進　指導方法概論

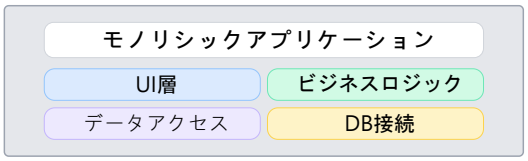
共通原理：マイクロサービスの必要性

分散システム、つまり、システムのマイクロサービス化は、システムの大規模化、複雑化に伴い、利用されることが増えてきました。アーキテクチャの違いを押さえておきましょう。

アーキテクチャの対比

モノリシック
単一プロセスで全機能を実行
→ シンプルだが、スケールと可用性に限界

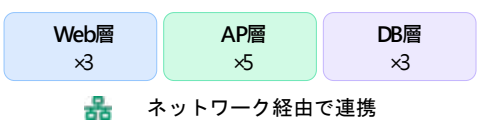
単一サーバー



課題
単一障害点 / スケール限界あり /
部分的な拡張不可

分散システム
多ノード協調で機能を分散
→ 複雑だが、スケールと可用性を実現

複数サーバー・ノード



解決策
スケーラビリティ 水平拡張可能
可用性 冗長性確保
柔軟性 独立変更

技術選択のトレードオフ

全システムは「マイクロサービスにするべき」というのは誤りです。組織の成熟度・運用体制・ビジネス要件に応じて最適なアーキテクチャを選択することが肝要です。

	モノリシック	マイクロサービス
シンプルさ vs 柔軟性	**シンプル 1つのコードベース、統一された設計**	柔軟 サービスごとに技術選択可能
開発速度 vs 運用複雑性	**初期開発が速い**、**全体を一度にデプロイ**可能 運用はシンプル	運用が複雑 分散トレーシング、監視、調整が必要
一貫性 vs スケーラビリティ	**強い一貫性** ACIDトランザクション、 **データ整合性**	高いスケーラビリティ サービスごとに独立してスケール
チーム構成	小規模チーム向き 全員が全体を理解	大規模組織向き サービスごとに独立したチーム
技術スタック	**統一された技術** 1つの言語・フレームワーク	多様な技術選択 サービスごとに最適な技術
適用ケース	スタートアップ、MVP 早期検証、小規模システム	大規模システム 高可用性、独立したスケーリング

リカレント教育推進　指導方法概論

本質の伝え方：可視化とデモ

受講者の理解を深めるため、可視化すること、デモを行うことは効果的な教育手法です。
デモではエラーが発生しうるため、エラーケースを想定し、リアルタイムでトラブルシュートできるように準備しておきましょう。

可視化のアプローチ

アーキテクチャ図

システム全体像、コンポーネント間の関係性、データフローを視覚化。
ホワイトボード、draw.io、mermaidなどを活用。

コマンド実行結果

docker ps、kubectl get pods、ログ出力など、実際の動作を見せる。
「期待される出力」と「実際の出力」を対比。

ダッシュボード・メトリクス

Prometheus、Grafana、CloudWatch等で、システムの状態をリアルタイム可視化。負荷テスト時の変化を観察。

デモの進め方

1. **目的の明示**
   デモの目的および内容を説明
2. **前提条件の確認**
   現状と想定される挙動の説明
3. **実行**
   コマンドの説明および入力
4. **結果の観測**
   挙動の確認
5. **失敗時の復旧パス**
   エラーが出た場合の対処を事前準備

本質の伝え方：段階的複雑化

実際の開発に使えるレベルまでクラウドネイティブな技術力を身に付けるには、段階的に学習していくことが肝要です。最小構成から始め、受講者が各ステップで学びを積み重ねられるように指導してください。

1. **最小起動**
   最小構成で動作を確認
   まず動かす体験から
2. **基本機能**
   永続化・設定を追加
   実用に向けた第一歩
3. **実用レベル**
   スケール・冗長・監視
   本格的なシステム構成
4. **本番環境**
   CI/CD・セキュリティ・コスト
   運用を見据えた設計



セクション4
授業の組み立て方

座学中心の単元：概念理解と体系的な知識の構築

一方的な講義に終始せず、講義→確認→応用の3段階で学習者の理解を深めることが重要です。
講義では技術背景、図解やデモで視覚化することが重要です。

60分授業の構成例

- **導入 (5分)**
  前回の振り返り + 今回の学習目標提示
- **講義 (25分)**
  理論・背景・原理 + 図解/デモ
- **確認 (15分)**
  ミニ演習 + ペア説明
- **応用 (10分)**
  ケース討論「この技術をどう使う?」
- **まとめ (5分)**
  要点再確認 + 次回予告

リカレント教育推進　指導方法概論

演習中心の単元：実践的スキルと問題解決力の育成

講義を最小限にし、学習者が実際に手を動かす時間を最大化することが重要です。
個人演習では巡回しながら個別サポートを行い、振り返りも行ってください。

60分授業の構成例

- **導入 (5分)**
  ゴール明示 - 何ができるようになるか
- **事前説明 (10分)**
  最小限の講義 - 手順/注意点
- **個人演習 (25分)**
  ハンズオン実践+ 巡回サポート
- **解答 (15分)**
  解説、つまずき共有、ベストプラクティス
- **まとめ (5分)**
  要点再確認+ 次回予告

リカレント教育推進　指導方法概論

評価方法について

単元の内容 (座学中心 or 演習中心) に応じて、評価方法を変えてください。
確認テストや演習問題の進捗および成果物などで、理解度を把握して評価してください。

	座学中心	演習中心
主な評価方法	• 確認テスト（選択 / 記述問題) • 演習問題の成果物 • （＋α）レポート提出、口頭説明	• 確認テスト（選択問題） • 演習問題の進捗確認リスト • 演習問題の成果物
評価ポイント	知識・理解 概念/用語/背景/原理	技能・思考 実装力/問題解決/試行錯誤
評価タイミング	• 授業後の確認テスト • 演習問題のコマンド履歴、マニフェストの提出	• 授業後の確認テスト • 演習問題の成果物提出 (GitHub、キャプチャ)
該当単元	• 仮想化技術 • コンテナ技術基礎 • クラウドネイティブ概論 • コンテナ技術システム開発	• AWSクラウドネイティブ基盤実装 • コンテナ・クラウドセキュリティ

リカレント教育推進　指導方法概論

演習のファシリテーション

演習では個々のつまずきを丁寧に拾い上げ、それを全体の学びに変えることが重要です。
エラーは失敗ではなく、理解を深めるチャンスです。

演習中の指導者の動き方

1. 全体観察
全体を見渡し、学習者の進捗状況と表情を確認。
手が止まっている、画面を睨んでいるなどの兆候を察知。

2. つまずきポイント発見
エラーメッセージ、実行結果、コマンド履歴を確認。
同じ箇所で複数の学習者が詰まっていないか把握。

3. 個別サポート
エラーメッセージの読み方を教える。解答ではなく、
調べ方・考え方を一緒にデモンストレーション。

4. 全体共有
よくあるつまずきは全体に告知。
「同じエラーが出た人？」と確認し、解決プロセスを共有。

ファシリテーションの3つのコツ

エラーメッセージの読み方を教える
- エラーの種類を識別する（構文エラー、実行時エラー、論理エラー）
- エラーメッセージのどこを読むか（行番号、エラータイプ、詳細メッセージ）
- エラーメッセージを検索クエリに変換するコツ

つまずきを全体に共有してナレッジ化
- 「いい質問です」「同じところで悩む人は多い」と価値づけ
- 解決プロセスを言語化して全体に共有
- 同種のエラーをまとめて解説（パターン認識を促す）

小さく早くフィードバック
- 完成を待たずに中間チェック（方向性の確認）
- 正解だけでなく、思考プロセスを評価
- 次の一歩を具体的に示す（「まずここを確認してみよう」）

セクション5
指導者の学習戦略

指導者自身の学習戦略

指導者自身が継続的に学び続けることで、最新技術への理解が深まるだけでなく、「学ぶ姿勢」を学習者にモデル化できます。

1. **インプット**
   公式Doc/技術ブログ/動画/OSS/コミュニティ
2. **実践**
   小さく試す - 検証用ミニPoC
3. **教える**
   他者に説明 - 社内LT/授業に反映
4. **振り返り**
   学びのログ化 - 次の仮説へ

時間確保のコツ
週1時間の学習タイムを確保
通勤時間を活用
演習準備=学習機会と捉える



再掲）VOD学習ロードマップ

仮想化、コンテナ、クラウドネイティブに関する幅広い領域を取り扱う形になります。
全6つの単元で構成されており、入門、基本、応用・実践の順で学習することを推奨します。

分類	入門	基本	応用・実践
知識	仮想化技術	クラウド ネイティブ概論	
技術習得	コンテナ 技術基礎	コンテナ技術 システム開発	AWSクラウドネイティブ基盤実装 コンテナ・クラウド セキュリティ

1. **仮想化技術**
   基盤技術の理解・なぜ仮想化か？
2. **コンテナ技術基礎**
   軽量なコンテナ実行環境とは？
3. **クラウドネイティブ概論**
   クラウドネイティブの思想を理解
4. **コンテナ技術システム開発**
   Docker、kubernetesの開発技法
5. **AWSクラウドネイティブ基盤構築**
   AWSでの実践的なシステム構築
6. **コンテナ・クラウドセキュリティ**
   コンテナ・クラウド環境の安全性確保

リカレント教育推進　指導方法概論

指導者向け - 技術習得ロードマップ　Step1

クラウドネイティブの入門として、仮想化、コンテナの基礎を学習します。

仮想化技術

必須スキル

ハイパーバイザー / VM作成・管理 / ネットワーク設定

学習方法

1. 仮想化技術の各要素について概要を理解
2. 仮想化技術の各要素についてアーキテクチャを理解
3. VirtualBox/VMware導入
4. Ubuntu/CentOSで仮想VM作成・基本操作
5. VMware Docs/VirtualBox Manualで理論補完

時間目安: 5時間

実践ポイント

- 物理vs仮想の違いを図解で説明（リソース共有・隔離性）
- VMが必要だった背景を説明（統合・マルチテナント）
- 「壊してもOK」マインドセットを伝える

コンテナ技術基礎

必須スキル

Dockerコマンド操作 / Dockerfile作成 / イメージ管理 / ネットワーク・ボリューム / docker-compose

学習方法

1. Dockerのコンセプト・アーキテクチャを理解
2. Docker実行環境構築（Docker desktop）
3. Dockerコマンド操作習得
4. Dockerfileを自作して、コンテナ実装
5. docker-composeでWeb+DB構成のコンテナ実装

時間目安: 5〜10時間

実践ポイント

- 起動時間・リソースでVMとの差をデモ
- 可搬性の価値を説明（開発=本番環境）
- マルチステージ+最小イメージのベストプラクティス

リカレント教育推進　指導方法概論

指導者向け - 技術習得ロードマップ　Step2

クラウドネイティブ、コンテナオーケストレーションを学習し、クラウドネイティブの基礎を押さえます。

クラウドネイティブ概論

必須スキル

クラウドネイティブ / マイクロサービス / CICDパイプライン / Infrastructure as Code / 可観測性 / セキュリティ / AWS

学習方法

1. クラウドネイティブの内容およびアーキテクチャを理解
2. コンテナ、サーバレスの学習、関連ツールの操作
3. アジャイル開発を経験
4. GitHub＋GitHub ActionsでCI/CDパイプライン構築
5. AWSの各種サービス操作

時間目安: 10〜15時間

実践ポイント

- モノリスとマイクロサービスの違いを説明
- 導入背景・ビジネス価値を説明（スケーラビリティ・俊敏性）
- 手動デプロイの課題からIaCやCI/CDの効果を語る

コンテナ技術システム開発

必須スキル

Kubernetes環境構築 / kubectl操作 / Pod / ReplicaSet / Deployment / Service / 他リソース / マニフェスト作成

学習方法

1. Kubernetesのコンセプト・コンポーネントを理解
2. Kubernetes環境の構築
3. kubectl操作および各種リソース操作
4. マニフェストでの各種リソースのデプロイ・動作確認
5. kubectl describe / logs / execでデバッグ

時間目安: 15～20時間

実践ポイント

- 宣言的管理の便利さを演習を通じて経験
- Kubernetesのコンポーネントデプロイおよび動作確認
- Kubectlコマンドでのエラーメッセージの読み方整理

リカレント教育推進　指導方法概論

指導者向け - 技術習得ロードマップ　Step3

本番運用を見据え、AWSシステム構築およびセキュリティ対策を通じて、実践的な技術を体得します。

AWSクラウドネイティブ基盤構築

必須スキル

AWS CDK / CloudShell / CloudFormation / VPC / SG / ALB / WAF / ECR / ECS / GitHub＋GitHubActions / Codeシリーズ

学習方法

1. 各種AWSサービスの理解およびアーキテクチャの設計
2. ネットワーク基盤の設計・構築（VPC / SG / ALB / WAFなど）
3. コンテナ基盤の設計・構築（ECR / ECS / EKS / Fargate）
4. データベースの設計・構築（RDS）
5. パイプライン構築（GitHub / Codeシリーズ）

時間目安: 20〜30時間

実践ポイント

- 物理vs仮想の違いを図解で説明（リソース共有・隔離性）
- VMが必要だった背景を説明（統合・マルチテナント）
- 「壊してもOK」マインドセットを伝える

コンテナ・クラウドセキュリティ

必須スキル

RBAC / NetworkPolicy / PSS / Imageスキャン・署名・検証 / ShiftLeft / CI自動化 / Secrets / Istio / Observability / AWS EKS

学習方法

1. コンテナ技術、Linuxカーネルの理解
2. 多層防御の考え方、各レイヤーのセキュリティ対策を理解
3. Kubernetes環境のセキュリティ対策実施
4. CI/CDパイプラインへのセキュリティゲート導入
5. サービスメッシュおよびObservabilityの導入

時間目安: 15〜20時間

実践ポイント

- 具体的事故例でセキュリティリスクを実感
- 「利便性 vs セキュリティ」のトレードオフ説明
- セキュリティは設計の最初から組み込む（DevSecOps）

クロージング

仮想化から始まるクラウドネイティブの領域は多岐に渡っています。
その技術を一つひとつ紐解いていくと、段階的に積み上げられたものであると感じることができます。

仮想化、コンテナの基本的な概念やアーキテクチャ、操作方法など、
全体を俯瞰しながら、ご自身が身に付けるべき技術を見定めていきましょう。

また体得していくには、実際にクラウドネイティブなサービスやツールを操作することが効果的です。
座学だけでなく、ぜひ楽しみながら、手を動かして体得してください。

各単元について指導者向けのVOD教材も用意しておりますので、
参考にしていただきながら、実践的な学びを深めてください。

令和5年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」

仮想化技術

― 指導者向け資料 ―

情報技術者の技能アップデートのためのリカレント教育推進事業

仮想化技術　指導者向け資料

目的

仮想化とは「物理リソースを抽象化する技術」であり、サーバーに留まらず、ネットワーク、ストレージ、アプリケーションに至る多層に適用されている技術です。

VM・コンテナ・クラウドが同じ仮想化技術の流れにあり、現代ITには欠かせない技術であることを理解・伝達できるように、受講者に説明を行ってください。

仮想化技術　指導者向け資料

なぜ仮想化が必要なのか？— ビジネス課題と解決策

リソースの無駄と運用の硬直性を解決するために仮想化技術が発達しました。昨今は「リソースの節約」から、ビジネスの変化スピードに追従できるよう「柔軟に対応できること」が主目的となっています。

背景と課題

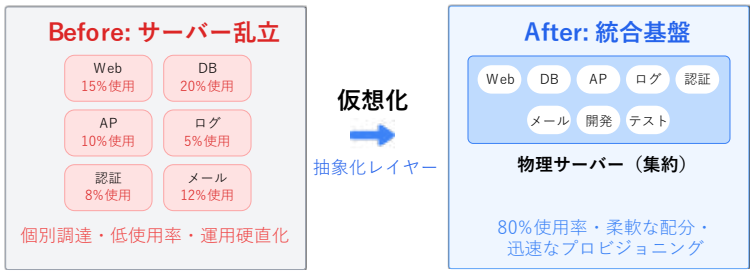
1960年代の課題:
高価なメインフレームを多人数で共有する必要性から仮想化が誕生

2000年代の課題:
- x86サーバーの乱立（サイロ化）
- 平均稼働率10〜20%の低使用率
- ラック/電力/保守コストの逼迫

解決策:
- ハイパーバイザ/コンテナによる論理分割
- イメージ化と自動化で配備時間短縮
- マルチテナント化でスケールメリット

効果:
コスト最適化、可用性向上、俊敏性、セキュリティ強化

仮想化による変革



効果： プロビジョニング時間を数週間から数分に短縮
サーバー台数を60%削減、電力/ラック使用量を40%削減

コスト削減だけではない：仮想化の真の価値

仮想化は“節約”以上に“俊敏性・可用性・品質”をもたらします。

	内容	効果/メリット
リソース効率化	物理サーバーの集約 CPUとメモリの高利用率 電力・ラック・冷却の削減	稼働率向上: 10-20%→60-80% サーバー数削減: 平均70-80%減
俊敏性	イメージ化と自動プロビジョニング セルフサービスポータル サーバー提供時間の短縮	導入時間短縮: 週・日→分・秒 TTM短縮: 30-50%削減
可用性	ライブマイグレーション 高可用性(HA)クラスタリング 障害復旧の自動化	稼働率向上: 99.9%→99.99% RPO/RTO: 時間→分・秒
セキュリティ	論理隔離と最小権限設計 スナップショットによる復元 イミュータブル（不変）インフラ	脆弱性対応時間: 短縮 復旧時間: 大幅短縮
開発効率	環境の標準化と再現性向上 CI/CDパイプラインの実現 Dev/Test/Prodの一貫性	バグ削減: 環境差異の解消 開発速度: 2-3倍向上
コスト構造	CapEx偏重からOpEx最適化へ 従量課金・動的拡張 リソース消費の可視化	TCO削減: 20-40% 投資対効果: 予測可能化

仮想化技術　指導者向け資料

仮想化の歴史：1960年代から現代まで

仮想化はメインフレームからクラウド・コンテナへと連続的に発展してきました。

仮想化技術の発展タイムライン

1960年代-1970年代 — メインフレーム時代

- 1964年: IBM CP-40
  最初の仮想化システム/研究
- 1967年: CP-67
  商用初の仮想化OS
- 1972年: VM/370
  商用仮想化の普及開始

1990年代-2000年代初期 — x86仮想化の始まり

- 1998年: VMware設立
  x86仮想化の民主化
- 1999年: VMware Workstation
  デスクトップ向け仮想化
- 2001年: ESX Server
  エンタープライズ向けハイパーバイザ

2000年代後半-2010年代前半 — クラウド時代

- 2006年: AWS EC2
  クラウド/IaaSの大衆化
- 2010年: OpenStack
  オープンソースクラウド基盤
- 2013年: Docker登場
  コンテナ革命の始まり

2010年代後半-2020年代 — コンテナ・オーケストレーション

- 2014年: Kubernetes
  コンテナオーケストレーション
- 2017年: サーバーレス普及
  AWS Lambda/Fargate
- 2020年代: マルチクラウド
  異種インフラの抽象化

メインフレーム
IBM CP-40/VM
→ x86仮想化
VMware
→ クラウド
AWS/Azure
→ コンテナ
Docker/Kubernetes
→ サーバーレス
Function as a Service

仮想化の本質は「抽象化」という考え方

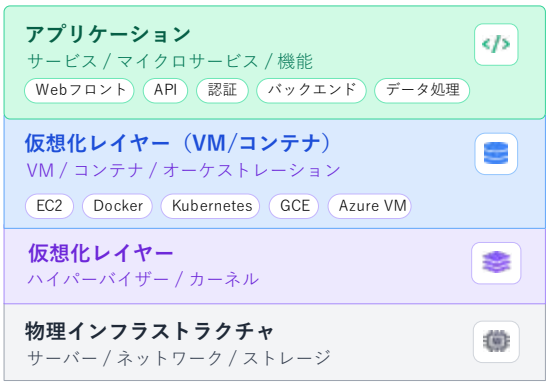
仮想化 = 基盤を隠し、論理リソースとして提供・利用可能にします。

抽象化の考え方

物理と論理の分離:
物理的なリソース（CPU/メモリ/ネットワーク/ストレージ）を論理的なリソースとして提供

複雑さの管理:
- 情報隠蔽によりシステム理解の負荷を軽減
- 明確なAPI境界で複雑な実装を隠蔽
- 一度に考慮すべき要素を削減

柔軟性の獲得:
- 配置/移動/拡張/復旧をソフトウェアで制御
- リソース割り当ての動的変更
- 物理的制約を超えた運用（クローン/スナップショット）



仮想化レイヤーが「橋渡し役」
物理世界の制約から解放された柔軟なIT環境を実現

仮想化技術　指導者向け資料

仮想化なしには現代のITは成立しない

クラウド/DevOps/マイクロサービスは仮想化の上に成立しています。
2020年からは、サーバレスやWebAssemblyなどの新しい仮想化技術の利用も進んでいます。

現代ITと仮想化の依存関係

クラウドコンピューティング:
- IaaS/PaaS/SaaSは全て仮想化で実現
- マルチテナント・動的スケーリング
- リソースの従量課金モデル

DevOps/CI/CD:
- イミュータブルインフラストラクチャ
- 環境の完全な再現性
- インフラのコード化（IaC）

マイクロサービスアーキテクチャ:
- 小さなコンテナ群の連携
- オーケストレーション（Kubernetes）
- サービスメッシュ/API管理

Serverless/Managed Services:
裏側では仮想マシンやコンテナが自動運転

最新の仮想化技術

さらなる軽量化

WebAssembly
ブラウザベース実行環境

AWS Firecracker
マイクロVM技術

サーバーレス
インフラ抽象化のさらなる進化

具体例：先進企業の活用事例
Netflix: 毎日数百万コンテナをAWS上で起動、可用性99.99%
Airbnb: 200+マイクロサービスをKubernetes上で運用
Spotify: Google Cloud上でKubernetesを活用し毎月1.2万デプロイ

サーバー仮想化3方式の違い

用途によって最適な方式が異なります。
各方式の特徴と用途の関係性を強調してください。

仮想化方式の比較

	ホストOS型	ハイパーバイザ型	コンテナ型
ホストOS	必要	不要	必要
ゲストOS	必要	必要	不要
起動時間	数分	数分	数秒
隔離性	高い	高い	中程度
用途	開発・検証	本番サーバー	マイクロサービス
具体例	VirtualBox	VMware ESXi	Docker



仮想化技術　指導者向け資料

VLANとオーバーレイネットワークの違い

VLANはL2ネットワークを仮想的に分割するものでした。
オーバーレイネットワークはL3ネットワーク上に構築するという違いがあります。

	VLAN	オーバーレイネットワーク
動作レイヤー	レイヤー2（データリンク層）	レイヤー3以上（ネットワーク層以上）
仕組み	スイッチポートに VLAN IDタグ（802.1Q）を付与	VXLAN/NVGRE等でカプセル化、 既存IPネットワーク上にトンネル構築
スコープ	同一L2ネットワーク内 （単一データセンター）	複数データセンター・クラウド間 （地理的分散）
制約	最大4096個（12ビットID）、 物理的な境界あり	1600万個以上（24ビットVNI）、 物理境界を超える
用途	部署分離、セキュリティゾーン分離	マルチテナント環境、クラウド間接続、 コンテナネットワーク

仮想化技術　指導者向け資料

最後に

仮想化はITの基礎的な技術で、サーバー、ネットワーク、ストレージなど多層で利用されています。
クラウドやマイクロサービスのベースとなる技術であることを説明したうえで、講座に取り組んでください。演習もあるため、事前にトラブル対処法を準備しておくと良いです。

【授業前に教員が確認すべきポイント】
・仮想化の概念、歴史や背景を理解しているか？
・ホストOS型/ハイパーバイザ型/コンテナ型の違いを図で説明できるか?
・IaaS/PaaS/SaaSの責任分界点を理解しているか?
・VLANとオーバーレイネットワークの違いを説明できるか?
・演習課題に取り組み、トラブル対処法が分かっているか？

仮想化技術　指導者向け資料

指導者が理解しておきたい知識１

用語編

- ◘ 「仮想化」の定義を、「抽象化レイヤー」という言葉で説明できる
- ◘ 「ホストOS」と「ゲストOS」の違いを説明できる
- ◘ 「ハイパーバイザー」が何をするソフトウェアか説明できる
- ◘ 「コンテナ」と「仮想マシン」の違いを「カーネル共有」で説明できる
- ◘ IaaS/PaaS/SaaSを「責任分界点」で区別できる
- ◘ VLANとVPNの違いを説明できる
- ◘ SDNが「ネットワーク制御の集中化」を意味することを理解している
- ◘ NFVが「ネットワーク機器の仮想化」を意味することを理解している
- ◘ 「マルチテナント」がクラウドの本質であることを理解している
- ◘ Linuxカーネルの機能として「namespace」「cgroup」などを理解している

他技術との違い

- ◘ VMwareとDockerの違いを「OS丸ごと vs カーネル共有」で説明できる
- ◘ ホストOS型とハイパーバイザ型の違いを「ホストOSの有無」で説明できる
- ◘ IaaSとPaaSの違いを「OS管理の有無」で説明できる
- ◘ VLANとオーバーレイネットワークの違いをで説明できる
- ◘ SDNとNFVの違いを「制御 vs 機能」で説明できる

仮想化技術　指導者向け資料

指導者が理解しておきたい知識２

動作の仕組み

- ホストOS型が「ホストOS上のアプリ」として動くことを理解している
- ハイパーバイザ型が「ホストOSなし」で動くことを理解している
- コンテナが「Linuxカーネルのnamespace」で隔離されることを知っている
- VLANが「スイッチポートにタグを付ける」仕組みを理解している
- VPNが「トンネリング」で拠点間を繋ぐことを理解している
- SDNコントローラが「OpenFlow」でスイッチを制御することを知っている
- Dockerが「イメージ」からコンテナを起動する仕組みを理解している
- AWS EC2が「ハイパーバイザ型(Xen/Nitro)」で動いていることを知っている
- Kubernetesが「コンテナオーケストレーター」であることを理解している

現場でのユースケース

- ホストOS型が「開発者のPC」で使われることを知っている
- ハイパーバイザ型が「データセンターの本番サーバー」で活用されることが多いことを知っている
- コンテナが「マイクロサービス」で使われることを知っている
- VPNが「リモートワーク」で使われることを知っている

仮想化技術　指導者向け資料

想定質問&解答例

初級編

Q1. 仮想化って、偽物のサーバーを作るってことですか?
「"偽物"ではなく、"柔軟に使える"サーバーです。例えば、物理サーバー1台を、仮想サーバー10台に分割できます。逆に、物理サーバー10台を、仮想サーバー1台に見せることもできます。これにより、故障しても自動で切り替わる仕組みが作れます。」

Q2. 仮想化すると遅くなるんですか?
「現代のVMは、CPUの仮想化支援機能により、物理マシンの90%以上の性能が出ます。ただし、ディスクやネットワークのI/Oは若干のオーバーヘッドがあります。コンテナはさらに軽量で、オーバーヘッドはほぼゼロです。」

Q3. クラウドって仮想化と同じですか?
「クラウドは仮想化技術を使って実現されたサービスです。例えば、AWS EC2は仮想マシンですし、AWS Lambdaはコンテナ技術を使っています。つまり、クラウド=仮想化の応用形です。」

Q4. SaaSって、普通のアプリと何が違うんですか?
「SaaSは、インストール不要でブラウザから使えるアプリです。例えばGmail、Slack、Notionなどです。PCにインストールするアプリ(Word等)と違い、どのPCからでも同じデータにアクセスできます。」

Q5. コンテナって、VMと何が違うんですか?
「VMはOSごと仮想化しますが、コンテナはOSカーネルを共有してアプリだけを隔離します。そのため、コンテナの方が軽量で、起動が数秒で済みます(VMは数分)。ただし、コンテナは同じOS(例:Linux)でしか動かせません。」

仮想化技術　指導者向け資料

想定質問&解答例

中級編

Q1. ホストOS型とハイパーバイザ型の違いは何ですか?
「ホストOS型は、WindowsやLinux上で仮想化ソフト(VirtualBox等)を動かします。ハイパーバイザ型は、ホストOSなしで、ハイパーバイザが直接ハードウェアを制御します。ハイパーバイザ型の方がオーバーヘッドが少なく、本番サーバー向きです。」

Q2. なぜコンテナはVMより速いんですか?
「コンテナはOSを起動しないからです。VMは、ゲストOSを起動するため、数分かかります。コンテナは、ホストOSのカーネルを共有するため、プロセス起動と同じ速度(数秒)です。」

Q3. VLANは、どうやってネットワークを分割するんですか?
「スイッチが、各ポートにVLAN ID(タグ)を付けます。同じVLAN IDのポート同士だけが通信できます。物理的には1台のスイッチですが、論理的には複数のスイッチとして動作します。」

Q4. SDNって、何が便利なんですか?
「従来のネットワークは、各スイッチを個別に設定する必要がありました。SDNは、中央のコントローラが全スイッチを制御するため、ネットワーク構成を一括変更できます。例えば、AWS VPCの設定には、SDN技術が使われています。」

Q5. NFVって、SDNと何が違うんですか?
「SDNは"ネットワークの制御"を仮想化し、NFVは"ネットワーク機器(ルーター、FW)"を仮想化します。NFVを使うと、専用ハードウェアなしで、ソフトウェアでルーターを実現できます。」

仮想化技術　指導者向け資料

想定質問&解答例

上級編

Q1. 仮想化すると、セキュリティは上がるんですか? 下がるんですか?
「仮想化自体は、隔離技術なのでセキュリティ向上に寄与します。ただし、ハイパーバイザに脆弱性があると、全てのVMが侵害されるリスクがあります(VENOM脆弱性など)。そのため、多層防御(ネットワーク分離、アクセス制御など)が重要です。」

Q2. コンテナは、VMより安全ですか?
「いいえ、VMの方が隔離性が高いです。コンテナはカーネルを共有するため、カーネル脆弱性で全コンテナが侵害される可能性があります。VMは、ゲストOSごとに独立したカーネルを持つため、隔離性が高いです。」

Q3. なぜ、Dockerが流行ったんですか?
「マイクロサービスアーキテクチャの普及が背景です。従来の1枚岩のアプリ(モノリス)ではなく、機能ごとに分割したアプリ(マイクロサービス)が主流になりました。各サービスを別々のコンテナで動かすと、独立してスケールできるため、Dockerが重宝されました。」

Q4. オンプレミスとクラウド、どっちが良いんですか?
「一概には言えません。オンプレミスは、初期投資は大きいですが、長期的にはコストが安定します。クラウドは、初期投資ゼロで始められますが、従量課金なので使いすぎると高額になります。最近は、両方を使うハイブリッドクラウドが主流です。」

Q5. 将来、仮想化はなくなるんですか?
「仮想化自体はなくなりませんが、形は変わります。例えば、WebAssembly(Wasm)は、ブラウザ外でもアプリを動かせる新しい技術です。また、Firecracker(AWS Lambda の基盤)は、コンテナとVMの中間のような軽量仮想化です。」

令和5年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」

コンテナ技術基礎

― 指導者向け資料 ―

情報技術者の技能アップデートのためのリカレント教育推進事業

コンテナ技術基礎　指導者向け資料

目的

コンテナ技術基礎は主にDocker、Kuberenetesといった技術について説明するカリキュラムです。

コンテナは昨今大規模なシステムでも利用されているキーとなる技術の一つです。
DockerやKubernetesに関連する要素や操作を体系的に整理して伝えられると良いでしょう。

また、なぜコンテナ技術が必要なのかを受講者が把握したうえで、
Docker→Kubernetes→クラウドサービス（AWSなど）と段階的に理解できるように促してください。

コンテナ（Docker）が広まった背景（2.1.2、2.2）

コンテナは、開発・運用・技術・ビジネスの全ての視点からメリットが大きく、クラウド時代に最適化された結果、業界標準として広まりました。

① 開発・運用の課題

- 「開発環境では動くが、本番環境では動かない」
  2000年代、**環境差分による不具合が深刻化**
- サービスダウンが社会的影響に直結
  速やかな復旧が必須な世の中になった

② 技術革新（コンテナ技術の成熟）

- Namespace / cgroups による**高速・軽量な隔離技術**
- **Dockerfile**による環境のコード化（IaC）
  LXCに比べて簡易にコンテナを実装可能
- レイヤー構造・キャッシュでビルド高速化
- **Docker Hub**による共有・再利用文化の爆発的広がり

③ ビジネススピードの加速

- **サービス提供時間の短縮がビジネス競争力に直結**
- サービス更新頻度が「年月」単位から「日時」単位に
- 高速なデリバリーが不可欠になった
  （ DevOps、CI/CD の浸透で、さらに加速化）

④ クラウド・マイクロサービスとの親和性

- Kubernetes による**自動復旧・自動スケール**
- マイクロサービス化（小さく作り、小さくデプロイ）
- AWS/GCP/Azure 全てがコンテナを標準サポート
- クラウドネイティブアーキテクチャの中心技術に

コンテナ技術基礎　指導者向け資料

コンテナのデータはどこに保存するか?（2.7.4）

コンテナは一時的にデータを保存可能です。コンテナは揮発性のため、削除するとコンテナのデータも消えます。永続的にデータ保存する場合は、永続なボリュームを利用してください。

データ永続化の仕組み

コンテナ内データの特性
コンテナ削除・**再起動**時にデータも消える（揮発性）

ボリュームの役割
- ホストOSや外部ストレージにデータを保存
- コンテナのライフサイクルと分離
- 複数コンテナ間でのデータ共有も可能

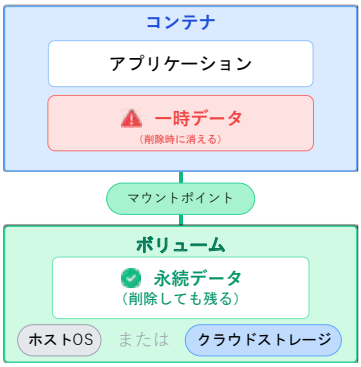
永続化が必要なデータ
- データベースのファイル
- ログファイル
- ユーザーがアップロードしたファイル
- 設定ファイル

実装方法

```
docker run -v /host/path:/container/path
kubectl apply -f pvc.yaml
```

データの永続化

Kubernetesの場合、Statefulset＋PVC＋PVの組合せで利用することが多いです。本単元ではあまり触れていないリソースですが理解しておくと良いです。



コンテナの隔離を支える2つの技術（2.7.5～2.7.7）

なぜコンテナは隔離されるのか？Linuxカーネル機能であるNamespaceとcgroupsで実現しています。
Namespaceでコンテナから見える範囲を制限し、cgroupsで使えるリソースを制限します。

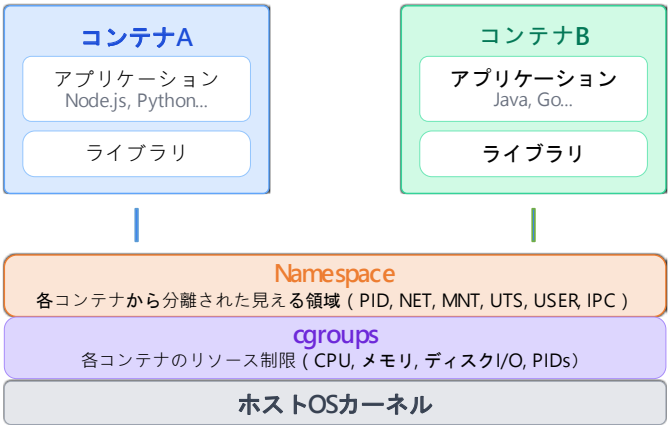
隔離技術の詳細

Namespace:
各コンテナから見える世界を制限する
- **プロセスID (PID)**
- ネットワーク (NET)
- **ファイルシステム (MNT)**
- ホスト名・ドメイン (UTS)
- **ユーザー (USER)**
- プロセス間通信 (IPC)

cgroups (Control Groups):
各コンテナが使えるリソースを制限する
- CPU使用率の上限
- メモリ使用量の上限
- ディスクI/Oの制限
- **プロセス数の制限 (PIDs)**

実現できること:
同一OS上で複数のアプリケーションを安全に隔離実行

コンテナの隔離構造



コンテナ技術基礎　指導者向け資料

イメージはレイヤーを重ねたもの（2.7.10）

イメージは重ね合わせであることを説明してください。docker historyコマンドで実際のレイヤー構造を見せると理解が深まります。　Dockerfileは変更頻度が少ない順で記載する旨をぜひ伝えてください。

Dockerイメージのレイヤー構造

レイヤー構造の仕組み
各Dockerfile命令が1レイヤーとして積み上げられる

レイヤーの共有
- **下位レイヤーは複数イメージで共有される**
- **ベースイメージ（Ubuntu等）の再利用**
- **ストレージ使用量を大幅に削減**

更新効率
- 変更されたレイヤーのみを転送
- キャッシュ機構による高速ビルド
- レジストリからの効率的なプル

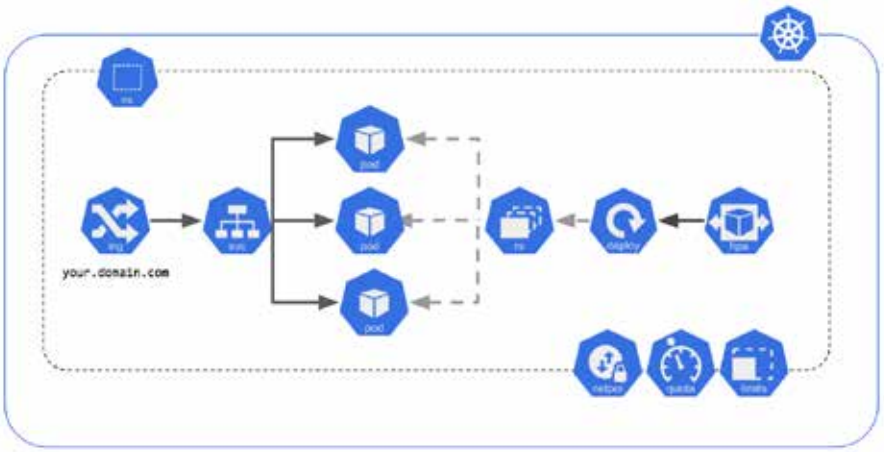
最適化テクニック
変更頻度の高いレイヤーは最後に配置

イメージレイヤーの構造



Kubernetesの大まかな構成（2.10）

Kubernetesの機能・要素はとても多く複雑です。大まかな構成および代表的なリソースの関係性を理解しましょう。**各リソースの役割、関係性、アイコンをセットで覚える**と学習しやすくなります。



https://github.com/octo-technology/kubernetes-icons

コンテナ技術基礎　指導者向け資料

Kubernetesの3層構造を理解しよう（2.11、2.13.15）

Deployment → Replicaset → Podの順で構成されます。各リソースをデプロイすることは可能ですが、実運用ではDeploymentを作成するのが一般的です。あとはReplicaset、Podが自動生成されます。

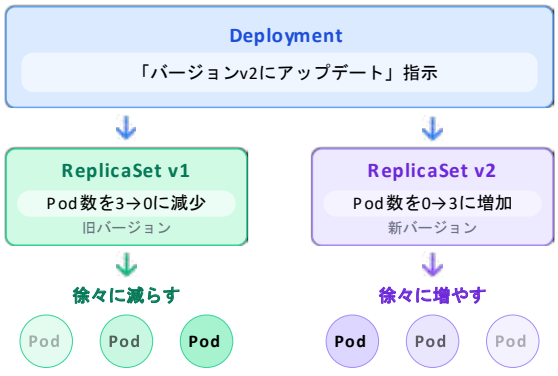
3層構造の役割

Pod:
コンテナを動かす最小単位
- 1つ以上のコンテナをグループ化
- 同じIPアドレスを共有
- 同一ノード上で実行

ReplicaSet:
指定した数のPodを確実に実行
- Pod数が不足したら新規作成
- Pod数が過剰なら削除
- 障害時の自動復旧

Deployment:
ReplicaSetを管理し、更新を制御
- ローリングアップデート実行
- ロールバック（元に戻す）機能
- 更新履歴の管理

ローリングアップデートの流れ



その他の起動方法（2.11、2.13.15）

Deployment / Replicaset / Pod 以外にもアプリケーションを実行するリソースがあります。
本単元では取り扱いませんが、挙動や用途が異なるため、合わせて知っておきましょう。

リソース	目的	Podの扱い	代表的な挙動	向いている用途
Deployment	ステートレスアプリの運用	Podは同一として扱う	ローリングアップデート / 自動再起動 / スケールアウト	Webアプリ、API、フロント
StatefulSet	ステートフルアプリの管理	Podに「順序」「固定ID」「固有PVC」	起動順序の保証 / 永続ボリュームの紐付け	DB、Kafka、Elasticsearch
DaemonSet	全ノードに常駐させる	各ノードに1Pod	ノード追加時、自動でPod追加	ログ集約、メトリクス、CNI
Job	一度だけ実行する処理	完了したら終了	成功回数やリトライ回数を管理	バッチ処理、移行スクリプト
CronJob	定期実行のJob	スケジューラが時間ごとにJobを生成	毎時・毎日など周期実行	バックアップ、定期バッチ

コンテナ技術基礎　指導者向け資料

どのServiceを使うべきか？（2.12.3）

Service TypeはClusterIP（内部通信）とLoadBalancer（外部公開）を利用するのが基本です。
Kubernetesクラスタで発生する通信によって使い分けてください。

Kubernetesのサービスタイプ比較

	ClusterIP	NodePort	LoadBalancer	ExternalName
主な用途	クラスタ内部通信 (一時的な外部公開)	テスト用途 (一時的な外部公開)	本番環境の外部公開 (公式エンドポイント)	外部サービスへのエイリアス (特殊用途)
アクセス範囲	クラスタ内のみ	ノードIP:ノードPort	外部IPアドレス (クラウドLB割当)	DNS名のみ (実体へリダイレクト)
設定の複雑さ	シンプル 自動作成	やや複雑 ポート指定可能	中程度 クラウド連携必要	やや複雑 CNAME設定必要

kubectlコマンド実行のコツ

Kubecltコマンドを使いこなすことで、作業やトラブルシュートの迅速化を図ることができます。

①CLIコマンドで略称を使用

Kubernetesには公式のショートネームが存在し、
長いリソース名を短縮して実行できます。
pod → po、 deployment → deploy 、
service → svc 、namespace → ns など

```
$ kubectl get po –n sample-app
$ kubectl create ns sample-app
```

②コマンド実行の簡略化

毎回 kubectl と入力するのは長いため、
エイリアス k = kubectl を設定することで時短可能です。

設定
```
$ alias k=kubectl
```

実行例
```
$ k apply -f app.yaml
$ k get po -n demo
```

③情報量アップの小技

ノードやIPまで表示	$ kubectl get pod xxx -o wide
実体のマニフェスト表示	$ kubectl get pod xxx -o yaml
適用前の差分チェック	$ kubectl diff -f file.yaml
状態遷移の監視	$ kubectl get pod -n xxx -w

④トラブルシュート時のコマンド

リソースの存在確認	$ kubectl get
リソースの詳細	$ kubectl describe
コンテナのログ確認	$ kubectl logs
コンテナ内部の確認	$ kubectl exec

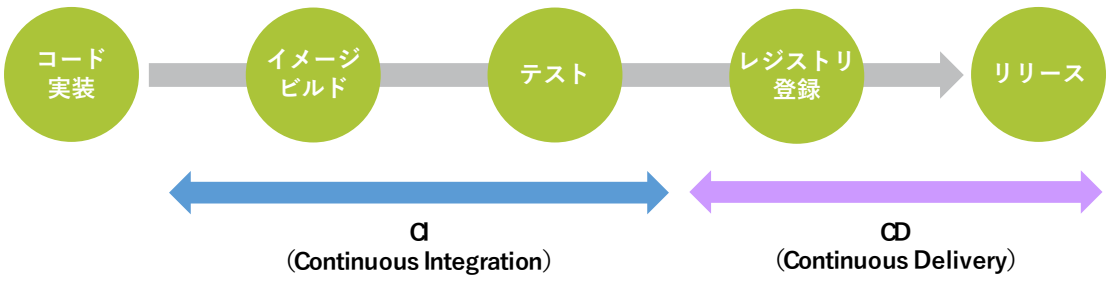
コンテナ技術基礎　指導者向け資料

CI/CDパイプライン（2.2、2.13.12～2.13.14）

自動ビルド・自動デプロイという一連の流れを自動化する仕組みをCI/CDパイプラインと言います。
コンテナのライフサイクルを回す上で必要な仕組みで、 DevOpsにも組み込まれます。

CI： 継続的インテグレーション。コードをマージするたびに、自動で品質チェックする仕組み。
CD：継続的デリバリー。いつでもデプロイできる状態を自動で作る仕組み。

CI/CDパイプライン構成



最後に

コンテナ、Docker、Kubernetes、クラウドサービスという流れを教えます。
複雑な内容は多くありませんが、扱うボリュームが多いです。

そのため、教える際は、「なぜこの技術が必要なのか」という背景を必ず説明し、コマンドの丸暗記ではなく、受講者が設計思想を理解できているかを重視してください。

操作デモや演習を実施する場合は、事前に動作確認をしたうえで、躓きやすいポイントを予めピックアップしておきましょう。

【授業前に教員が確認すべきポイント】
・コンテナの特徴を整理できているか
・Docker、Kubernetesの概念・アーキテクチャ・操作方法を理解しているか
・実際にDocker環境、Kubernetes環境を立ち上げ、コンテナや各種リソース（Deploy、Pod、Serviceなど）の立ち上げや動作確認をしているか
・代表的なクラウドサービス（AWS、GCP、Azureなど）の違いや提供しているサービスを知っているか
・AWSの各サービスを利用し、仮想環境（ネットワーク、サーバー、コンテナ）を立ち上げている

コンテナ技術基礎　指導者向け資料

指導者が理解しておきたい知識１

用語編

- コンテナ:隔離されたプロセス(仮想マシンではない)
- イメージ:コンテナを作るためのテンプレート(レイヤーの集合)
- Dockerfile:イメージをビルドするための設計書(コードで記述)
- レジストリ:イメージを保管・共有する場所(Docker Hub、AWS ECR)
- Pod:Kubernetesの最小デプロイ単位(1つ以上のコンテナ)
- Deployment:Podの複製と更新を管理するリソース
- Service:Podへのアクセスを抽象化するリソース(内部DNS)
- Namespace(K8s):クラスタ内の仮想的な区画(開発/本番の分離)
- Namespace(Linux):プロセスの隔離技術(PID、ネットワーク等)
- cgroups:リソース(CPU、メモリ)の制限技術

他技術との違い

- コンテナはプロセスレベル、VMはハードウェアレベルの仮想化
- DockerはコンテナランタイムとIaC(Dockerfile)を統合したプラットフォーム
- Kubernetesはコンテナオーケストレーター(Dockerの上位ツール)
- ECSはAWS独自、EKSはKubernetes互換(マルチクラウド可能)
- Fargateはサーバーレスコンテナ(EC2インスタンス管理不要)

指導者が理解しておきたい知識２

動作の仕組み

- コンテナはホストOSのカーネルを共有する(独自カーネルを持たない)
- イメージはレイヤーを積み重ねて構成される(Copy-on-Write)
- Dockerfileの各命令が1つのレイヤーになる
- レイヤーは読み取り専用、コンテナ実行時に書き込みレイヤーが追加される
- docker runは新しいコンテナを作成、docker startは既存コンテナを再起動
- Kubernetesは宣言的な設定(あるべき状態を宣言、実現方法は自動)
- Deploymentを作ると、自動的にReplicaSetとPodが作成される
- Serviceはラベルセレクタで Podを選択(IPアドレスではない)
- kube-schedulerは、リソース状況を見てPodをどのノードに配置するか決定
- kubeletは、マスターからの指示を受けて実際にコンテナを起動

現場でのユースケース

- ローカル開発環境を本番環境と一致させる(環境差異問題の解決)
- CI/CDパイプラインで、コードpush→自動ビルド→自動デプロイ(デプロイの承認プロセスは必要)
- マイクロサービスは、サービスごとにコンテナ化し、個別にスケール
- ローリングアップデートではなく、Blue/Greenデプロイ、カナリアリリースで段階的に新バージョンを投入

コンテナ技術基礎　指導者向け資料

指導者が理解しておきたい知識３

セキュリティ観点で注意すべき点

- Docker デーモンはroot権限で動作→コンテナ実行ユーザーにroot同等の権限
- 公式イメージ以外は、マルウェアが含まれる可能性(Docker Hub)
- コンテナ内でrootユーザーで実行しない(非特権ユーザーを使う)
- SecretsやConfigMapで機密情報を管理(ハードコード禁止)
- ネットワークポリシーで、Pod間通信を制限(デフォルトは全許可)
- IAMロールでAWSリソースへのアクセスを最小権限化

コンテナ技術基礎　指導者向け資料

想定質問&解答例

初級編

Q1:「コンテナと仮想マシンは、どちらが優れているんですか?」
「優劣」ではなく「用途による使い分け」です。コンテナは軽量で起動が速いため、開発やマイクロサービスに向いています。一方、仮想マシンは隔離が強力で、異なるOSを完全に分離したい場合(例:Windowsサーバーの仮想化)に向いています。現実には、両方を組み合わせる(VMの中でコンテナを動かす)ことも多いです。

Q2:「コンテナはどうやって隔離されているんですか?」
LinuxカーネルのNamespaceとcgroupsという機能で実現されます。Namespaceは「プロセスから見える範囲」を制限し、cgroupsは「使えるCPUやメモリ」を制限します。これにより、同じOS上で動作してもコンテナ同士は互いに干渉しません。

Q3:「DevOpsって、開発と運用が仲良くするってことですか?」
精神論だけではありません。DevOpsは「開発と運用のサイクルを自動化し、迅速にサービスを改善する手法」です。コンテナ技術は、このサイクルを加速させる道具です。例えば、開発環境と本番環境で同じコンテナを使えば、「俺の環境では動く」問題が解消されます。

Q4:「マイクロサービスとモノリスは、どっちが良いんですか?」
これも用途次第です。マイクロサービスはスケーラビリティと開発速度に優れますが、管理が複雑です。小規模なサービスや、チームが少ない場合は、モノリスの方がシンプルです。実務では、ハイブリッド構成(一部だけマイクロサービス化)も多いです。

Q5:「サーバーレスって、サーバーがないんですか?」
いいえ、裏ではサーバーが動いています。「サーバーレス」は「サーバーを意識しなくて良い」という意味です。AWS Lambdaなどのサービスでは、コードのアップロードだけすればよく、サーバー管理はクラウド側が自動でやってくれます。

コンテナ技術基礎　指導者向け資料

想定質問&解答例

中級編

Q6:「Dockerイメージは、なぜレイヤー構造なんですか?」
ディスク容量とダウンロード時間を節約するためです。例えば、Ubuntu OSをベースにした10個のイメージを作っても、Ubuntuの部分は1回だけ保存されます。また、アプリコードだけを更新した場合、差分のレイヤーだけをダウンロードすればOKです。これを「Copy-on-Write」と呼びます。

Q7:「docker runを何度も実行すると、コンテナが増えるんですが…」
その通りです。docker runは毎回新しいコンテナを作成します。既存のコンテナを再起動したい場合は、docker start <コンテナ名>を使ってください。不要なコンテナはdocker rmで削除しましょう。コンテナー一覧はdocker ps -aで確認できます。

Q8:「ポートマッピング(-p 8080:80)の意味が分かりません」
これは「ホストマシンのポート8080を、コンテナのポート80に転送する」という意味です。コンテナは隔離されているため、外部から直接アクセスできません。ポートマッピングで「窓口」を開けてあげる必要があります。ブラウザでlocalhost:8080にアクセスすると、コンテナ内の80番ポートに届きます。

Q9:「Dockerfileのキャッシュが効かないんですが…」
Dockerfileは上から順に実行され、変更行以降は全て再実行されます。例えば、COPY app.pyを変更すると、その下の命令も全て再実行されます。そのため、変更頻度の低い命令を上に、高い命令を下に配置すると、キャッシュが効きやすくなります。

Q10:「PodとDeploymentの違いが分かりません」
Podは「1人の作業員」、Deploymentは「作業員を3人確保して、バージョンアップする指示書」です。Deploymentを作ると、自動的にReplicaSetとPodが作られます。実務では、Podを直接作ることは稀で、Deploymentだけ作れば良いです。

想定質問&解答例

上級編

Q11:「本番環境でDocker Composeを使っても良いですか?」
小規模なら可能ですが、推奨されません。Docker Composeは単一ホストでの実行を前提としており、複数サーバーにまたがる構成や、自動復旧、負荷分散などの機能がありません。本番環境では、KubernetesやAWS ECS/EKS を使うのが一般的です。

Q12:「LoadBalancerタイプのServiceを作ったのに、外部IPが割り当てられません」
AWS、GCP、Azureなどのクラウド環境では動作しますが、ローカル環境によってはLoadBalancerは機能しません。kubectl port-forwardコマンドでポート転送する、NodePortタイプのServiceを使うなどして対処してください。

Q13:「ECSとEKSは、どちらを使うべきですか?」
ECS:AWS独自の仕様で、設定がシンプル。AWS に最適化されており、学習コストが低い。
EKS:Kubernetes標準で、マルチクラウド対応。他のクラウドや オンプレミスにも移行しやすいが、学習コストが高い。
「将来的に他のクラウドに移行する可能性」や「Kubernetesのスキルを身につけたいか」で判断してください。

Q14:「Docker Hubのイメージは、全部安全ですか?」
いいえ。公式イメージ以外は、第三者が作成したもので、マルウェアが含まれる可能性があります。公式イメージを優先する、ダウンロード数や評価を確認する、Dockerfileを書くなどして、信頼できるベースイメージを利用してください。

Q15:「コンテナは、どこまで小さくすべきですか?」
「1コンテナ1プロセス」が原則ですが、極端に小さくする必要はありません。例えば、WebサーバーとPHPを同じコンテナに入れるのは普通です(nginx + PHP-FPM)。重要なのは「独立してスケールできるか」「障害が他に波及しないか」です。

令和6年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」

クラウドネイティブ概論

― 指導者向け資料 ―

情報技術者の技能アップデートのためのリカレント教育推進事業

クラウドネイティブ概論　指導者向け資料

はじめに

クラウドネイティブは、現在ITサービスを提供するうえでも、業界標準で適用されている考え方です。

昨今の技術革新やビジネススピードの高速化に伴い、クラウドネイティブを理解していることはエンジニアとして必須です。

本教材は、クラウドネイティブについて体系だって学ぶことができます。
しかし、取り扱う内容が広く、クラウドネイティブの考え方および技術の紹介になりがちです。

クラウドネイティブを俯瞰しながら、各単元の位置づけを理解し、受講者をサポートすることを心掛けてください。

クラウドネイティブ概論　指導者向け資料

本教材の位置づけ

クラウドネイティブに関する幅広い領域について概念や仕組みを説明する単元です。
仮想化やコンテナの基礎を理解していることが前提となります。

分類	入門	基本	応用・実践
知識	仮想化技術	**クラウドネイティブ概論**	
技術習得	コンテナ技術基礎	コンテナ技術システム開発	AWSクラウドネイティブ基盤実装 コンテナ・クラウドセキュリティ

クラウドネイティブ概論　指導者向け資料

教材について

本教材の特徴

1. **抽象度が高い概念が多い**
   - 「クラウドネイティブは考え方」「DevOpsは文化」など具体的な製品やコードではなく思想・哲学を扱う
   - 学習者は「何を実装すればいいのか」が見えにくい
2. **技術要素が広範で相互依存**
   - コンテナ → Kubernetes → CI/CD → IaC → Observability → セキュリティ と続く
   - 単独の章だけでは理解が浅く、関係性が見えにくい
3. **実務経験がないと難しい箇所がある**
   - マイクロサービスの結果整合性、サービスメッシュ、ゼロトラスト等　唐突に出てくる項目があります。
   - 「なぜそれが必要なのか」の実感が湧きにくい。

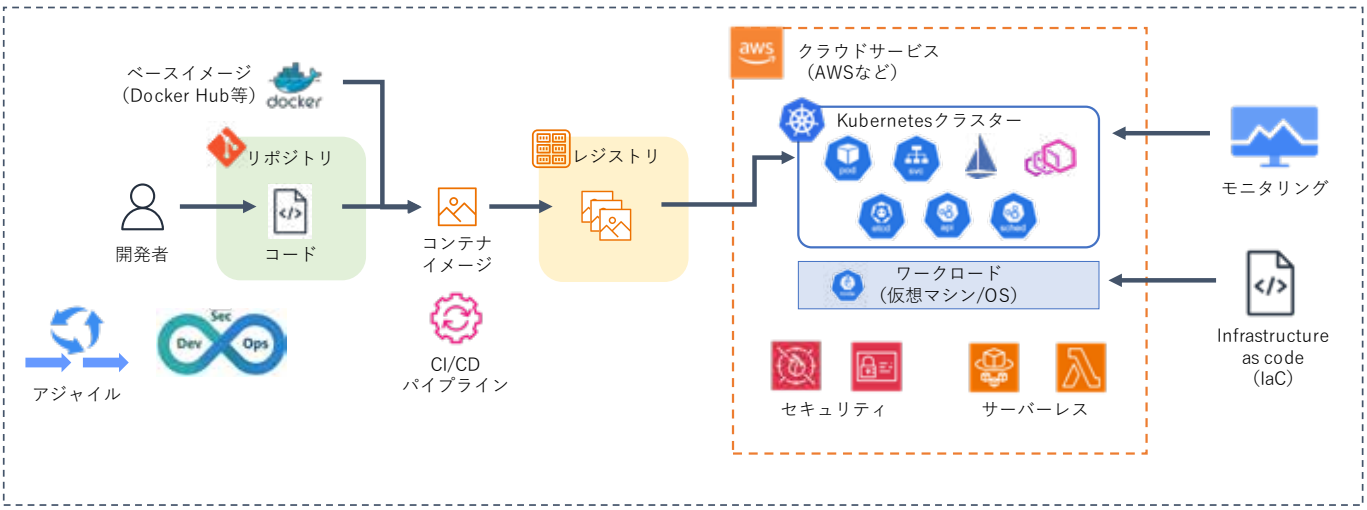
教える際の課題

1. **抽象的な概念をどう具体化するか**
   - 「クラウドネイティブとは考え方」と言われても、何をすればクラウドネイティブなのか判断できない
   - 抽象的な思想から具体的な実装への橋渡しが必要
2. **技術の羅列で終わらせない**
   - Docker、Kubernetes、Terraform…と用語を並べても、「なぜそれらが必要なのか」のストーリーが欠けると暗記学習になる。
   - 技術同士の関連性と導入理由を明確にする必要がある。
3. **実践とのギャップをどう埋めるか**
   - 教材は体系的だが、実際の現場では「全部やる」わけではありません。場合によっては内容の割愛が必要。
   - 優先順位や段階的導入の現実を教えないと、学習者が実務で困ります。

クラウドネイティブ概論　指導者向け資料

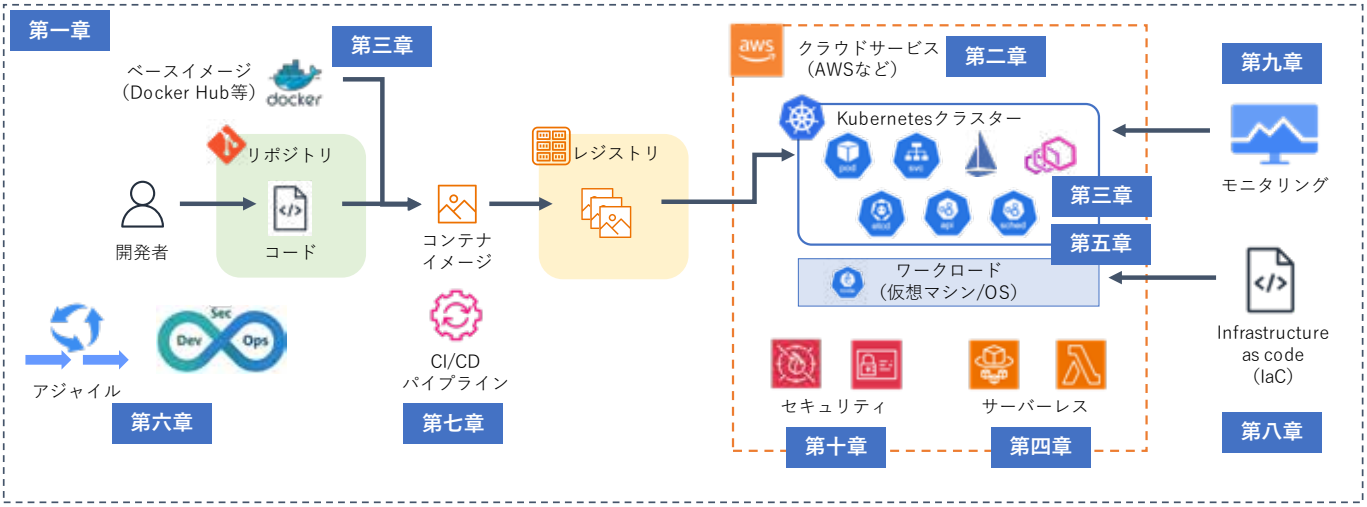
学習マップ

クラウドネイティブ概論で取り扱う範囲はとても広いです。
クラウドネイティブの全体構成から見て、各章がどんな位置づけなのかを常に把握してください。



学習マップ（マッピング版）

クラウドネイティブ概論で取り扱う範囲はとても広いです。
クラウドネイティブの全体構成から見て、各章がどんな位置づけなのかを常に把握してください。



なぜクラウドネイティブなのか？

クラウドネイティブは、ビジネス、技術、運用といった観点から、現代システムの標準的な考え方になっています。背景や考え方を理解しつつ、優先順位をつけて、各技術を体得していくと良いでしょう。

① ビジネススピードの課題

従来	CN後
新機能追加に数ヶ月	数時間〜数日で本番リリース
競争に対応できない	市場変化に迅速対応

解決手段
CI/CD自動化、マイクロサービス、
コンテナ、**アジャイル**

② スケールの課題

従来	CN後
急激なアクセス増→サーバーダウン	自動スケーリングで負荷に対応
機会損失・信頼低下	安定性と顧客満足度向上

解決手段
クラウドの弾力性、コンテナ軽量化、
コンテナオーケストレーション

③ コストの課題

従来	CN後
ピーク時に合わせた過剰なリソース確保	使った分だけ課金、無駄なコスト削減
リソースの無駄と高コスト	コスト効率と投資最適化

解決手段
従量課金、サーバーレス、
リソース最適化

歴史的必然性

- 2006年 AWS誕生
- 2013年 Docker
- 2015年 Kubernetes
- 2020年代 CN標準化

学ばなければ、現代のシステム開発に参加できない

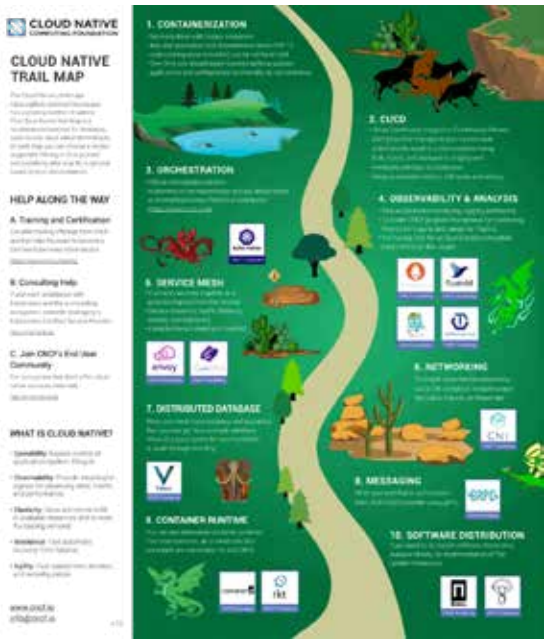
クラウドネイティブ トレイルマップ

クラウドネイティブな仕組みは無数にありますが、
CNCFのプロジェクトではクラウドネイティブへの標準的な道のりを提示しています。

トレイルパス

1. コンテナ化
2. CI/CD
3. オーケストレーション
4. 可観測性・解析
5. サービスメッシュ
6. ネットワーク
7. データベース
8. メッセージ
9. コンテナランタイム
10. ソフトウェア

https://www.cncf.io/blog/2018/03/08/introducing-the-cloud-native-landscape-2-0-interactive-edition/



クラウドネイティブ概論　指導者向け資料

最後に

クラウドネイティブ概論についてポイントを説明してきました。
抽象的かつ幅広い内容を取り扱いました。概念と具体例をセットで説明し、指導者も受講者も理解が深めてください。また各章がどんな位置づけなのか、全体像を踏まえたうえで指導されることも大切です。

知識ベースで止まると理解が浅いままになりがちです。
どんな技術として利用されているのか、ぜひご自身で実際に各ツールを利用して頂き、知識と技術習得の両輪で理解を深めてください。

【授業前に教員が確認すべきポイント】
・クラウドネイティブは「技術」ではなく「考え方」であり、その考え方を説明できる
・全10章の位置づけ、相互の依存関係を意識できている
・抽象概念は必ず具体例とセットで説明できる（例えば、冪等性、オーケストレーションなど）
・実際にDocker、Kubernetes環境を立ち上げ、操作している

クラウドネイティブ概論　指導者向け資料

指導者が理解しておきたい知識１

基本的な概念

- ◘ クラウドネイティブの定義を説明できる
- ◘ クラウドネイティブと従来型の違いを、3つ以上挙げ、その内容を説明できる
- ◘ IaaS、PaaS、SaaSの違いを、責任範囲で説明できる
- ◘ コンテナとVMの違いを説明できる
- ◘ マイクロサービスとモノリスの違いを、利点と欠点を含めて説明できる
- ◘ DevOpsの意味を、「開発と運用の協働」以上に説明できる
- ◘ CI/CDの違い（CI = 継続的インテグレーション、CD = 継続的デリバリー）を説明できる
- ◘ Infrastructure as Code（IaC）の意味と、なぜ必要かを説明できる
- ◘ 可観測性（Observability）と監視（Monitoring）の違いを説明できる
- ◘ シフトレフトの意味を説明できる

クラウドについて

- ◘ 主要クラウドプロバイダー（AWS、Azure、GCP）の特徴が分かっている
- ◘ オンプレミス、クラウド、ハイブリッドクラウドの違いと、それぞれのメリット・デメリットを説明できる
- ◘ クラウドネイティブが適さないシステム例を挙げられる

クラウドネイティブ概論　指導者向け資料

指導者が理解しておきたい知識２

技術的な仕組み

- ◘ Dockerイメージとコンテナの違いを、「設計図と家」の例えで説明できる
- ◘ Dockerfileの基本構文（FROM, RUN, COPY, CMD）を理解している
- ◘ Kubernetesの役割を、「コンテナのオーケストレーター」と説明できる
- ◘ CI/CDパイプラインで実現できる内容をCI、CDを分けて説明できる
- ◘ IaCツールの宣言型、手続型の違いを説明できる
- ◘ サーバーレス（FaaS）の仕組みと、コンテナとの違いを説明できる
- ◘ リポジトリとレジストリの役割を理解している
- ◘ 可観測性の3つの柱と役割を理解している
- ◘ クラウド環境のセキュリティ対策を説明できる

その他

- ◘ ウォーターフォールとアジャイル、開発手法の違いを説明できる
- ◘ 主だったブランチ戦略を説明できる
- ◘ コンテナのデプロイ戦略を説明できる
- ◘ ゼロトラストの考え方について説明できる

クラウドネイティブ概論　指導者向け資料

想定質問&解答例

初級編

Q1. クラウドネイティブって、クラウド上で動くアプリのことですよね？
違います。『クラウド上で動く』ではなく『最初からクラウドの特性（自動スケール・柔軟性・自動化）を活かす設計思想』のことです。

Q2. コンテナって、VMと何が違うんですか？
VMはOS全体を仮想化、コンテナはOSカーネルを共有してアプリ層だけ仮想化するため軽量で速いです。

Q3. マイクロサービスって、小さいサービスってことですか？
『小さい』ではなく『独立して動くサービスの集合』で、商品管理・注文処理など機能ごとに分けて、1つが止まっても他は動き続ける設計です。

Q4. DevOpsって、開発と運用を一緒にやることですか？
半分正解。開発と運用の『壁を壊して協働する文化』で、両者が同じツールを使い自動化を進める働き方の変革です。

Q5. Kubernetesって、何ですか？
コンテナのオーケストレーター（指揮者）で、100個のコンテナを自動で起動・停止・スケール・復旧してくれるので手動管理が不要になります。Docker composeでも複数コンテナを管理できますが、コンテナの数が増えるほどに管理が困難です。

Q6. サーバーレスって、サーバーがないんですか？
サーバーはありますが『サーバー管理が不要』という意味で、クラウド事業者が管理してくれるのであなたはコードを書くだけでOKです。

クラウドネイティブ概論　指導者向け資料

想定質問&解答例

中級編

Q7. DockerイメージとDockerコンテナは、どう違うんですか？
イメージは設計図（Dockerfileから作成）、コンテナは設計図から建てた実際のインスタンスです。
1つのイメージから同一のコンテナを何個でも作れます。

Q8. Dockerfileって、何ですか？
「Dockerイメージの設計図を書くファイルで、FROM（ベースOS）、RUN（インストール）、COPY（ファイル配置）、CMD（起動コマンド）などの手順を記述します。

Q9. PodとContainerは、どう違うんですか？
Container=Dockerが作るアプリ実行環境、Pod=Kubernetesが管理するコンテナのグループで、基本は1Pod=1Containerですが複数入れることも可能です。例えばIstioなどのサービスメッシュでは、サイドカーコンテナを追加しています。

Q10. ローリングアップデートって、何ですか？
サービスを止めずに1台ずつ新バージョンに切り替える方法で、Kubernetesが自動でやってくれるのでダウンタイムがゼロになります。リリース戦略を考えた場合、Blue/Greenデプロイメント、カナリアリリースなど別のリリース方式も知ったうえで最適なリリース方法を選びましょう。

想定質問&解答例

上級編

Q11. マイクロサービスって、常にモノリスより優れているんですか？
いいえ。独立デプロイや障害局所化は利点ですが、複雑性と運用コストが増すため、小規模チームにはモノリスの方が適していることが多いです。

Q12. コンテナイメージの脆弱性スキャンって、何ですか？
イメージ内のOSやライブラリのセキュリティ脆弱性をTrivyなどのツールでチェックし、CI/CDパイプラインで危険なイメージのデプロイを防ぐ仕組みです。

Q18. シークレット情報って、どう管理すればいいですか？
絶対にコードに書かず、環境変数・Kubernetesのシークレット・専用ツール（AWS Secrets Manager、Vault）を使い、Gitへの漏洩を防ぎます。

Q19. ログって、どうやって集約するんですか？
マイクロサービスの複数ログをFluentbit（収集）→Elasticsearch（保存・検索）→Kibana（可視化）といったツールで一箇所に集めることで、問題の原因を追跡可能にします。

令和6年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」

コンテナ技術システム構築

― 指導者向け資料 ―

情報技術者の技能アップデートのためのリカレント教育推進事業

コンテナ技術システム構築　指導者向け資料

はじめに

Kubernetesを利用し、コンテナを使ったシステム構築のエッセンスを体得する単元です。
現代ITシステムの標準であるコンテナを理解することは、現代ITを体得するために必須となります。

すでに仮想化、コンテナ、クラウドネイティブについて学習をしていることが前提となります。
背景や技術のベースを学習したうえで、コンテナを使ったシステム構築の手法を学んでいきましょう。

コンテナ技術システム構築　指導者向け資料

本教材の位置づけ

Kubernetesを利用してコンテナ技術を習得します。
仮想化・クラウドネイティブの概念を理解し、コンテナを操作できることが前提となります。

分類	入門	基本	応用・実践
知識	仮想化技術	クラウド ネイティブ概論	
技術習得	コンテナ 技術基礎	**コンテナ技術 システム開発**	AWSクラウドネイティブ基盤実装 コンテナ・クラウドセキュリティ

前提知識チェック

1. **Dockerの基本コンセプト**
   イメージ、コンテナ、Dockerfile の役割と関係性を説明できるか
2. **Linux基本コマンド**
   ls cd cat grep curl 等の操作に慣れているか
3. **Webアプリの構造**
   クライアント-サーバー、HTTPプロトコル、ポート番号の概念理解
4. **クラウドネイティブのコンセプト**
   マイクロサービス、CICD、クラウドサービスなどの概念理解

この教材に含まれる範囲

この教材では、コンテナの基本的な仕組みを取り扱います。
含まれない内容については、別教材で取り組みます。

含まれる内容　Focus

- **Docker / Kubernetesの基本**
  イメージ作成、コンテナ起動、Podの管理
- **主要リソースの使い方**
  Deployment, Service, ConfigMap, Volumeなど
- **クラウドサービス導入の基礎**
  マネージドK8s（EKS/GKE/AKS）の基本操作

含まれない内容　Out of Scope

- **コンテナセキュリティ**
  RBAC, Network Policy, Pod Security Standardsなど
  →別教材「コンテナセキュリティ」へ
- **CI/CDパイプライン**
  GitHub, GitHub Actions, AWS Codeシリーズ
  →別教材「AWSクラウドネイティブ開発」へ
- **高度な周辺ツール**
  Helm, ServiceMesh, Observabilityなど
  →別教材「コンテナセキュリティ」へ

コンテナ技術システム構築　指導者向け資料

kubectl デバッグのコツ

コンテナが「動かない」時は、kubectlコマンドを利用することで、作業やトラブルシュートの迅速化を図ることができます。

①ステータス確認

```
$ kubectl get pod
```

確認ポイント
- STATUS：Runningか？
- READY：0/1になっていないか？
- RESTARTS：回数が多くないか？

異常値の例
- Pending
- Error
- CrashLoopBackOff
- ImagePullBackOff

②詳細・イベント確認

```
$ kubectl describe pod xxx
```

確認ポイント
- EVENT：異常なイベントがないか？

EVENTは時系列で表示

Warning FailedMount ...
Warning Failed
ImagePull.

③ログ確認

```
$ kubectl logs xxx
```

確認ポイント
以下の様な内容が出ていないか
- アプリのエラー出力
- DB接続エラー
- 設定ファイル読み込み失敗

Tips

--previous
前回の実行ログを見る

④コンテナ調査

```
$ kubectl exec -it <name> --
/bin/sh
```

確認ポイント
- ファイル存在確認（ls, cat ）
- ネットワーク疎通（curl, ping ）
- プロセス確認（ps aux ）

注意点
- bashがない場合は、/bin/sh を利用してください
- 本番環境の場合、Podへのexec権限がない可能性があります

その他Tips
- Pod名は毎回変わるため、都度kubectl get podでPod名を確認してから詳細を確認しましょう
- Podを削除するとログなどの情報が削除されるため、トラブル時にすぐPod再起動をしないようにしましょう

YAML記述の基本とエラー対処

Kubernetesの利用にあたりマニフェスト（YAMLファイル）を利用することが一般的です。
YAML記述の基本をおさえることで、デプロイエラーを低減できます。

YAML記述の注意点

1. **インデントは半角スペースのみ**
   タブ、全角スペースは厳禁
2. **インデント幅を統一**
   通常はスペース2つで統一
3. **コロンの後にスペース**
   key: value の形

エラーメッセージの読み方

```
error converting YAML to JSON: yaml: line 5: found character that cannot start any token
```

Line X の行番号を確認
その行の前後も含めてチェック
インデントの縦並びを確認

Error 1: タブ/全角スペースの使用

見た目では分からないため最凶

Bad

```
apiVersion: v1
kind: Pod
metadata:
    name: my-app
```

↑タブや全角スペースが入っている

対処法

- エディタで「タブをスペースに変換」
- 制御文字を表示する設定にする

Error 2: インデント数の不一致

階層構造が壊れる

Bad

```
spec:
  replicas: 3
    mplate:        ←インデントのズレ
  metadata:
```

Good

```
spec:
  replicas: 3
  template:        ←インデントが一致
    metadata:
```

Error 3: コロンを含む値

パースエラーの原因

Bad

```
image: myapp:v1.0
```

↑値の中の「:」が区切りと誤認

Good

```
image: "myapp:v1.0"
```

↑引用符で囲む

コンテナ技術システム構築　指導者向け資料

どのServiceを使うべきか？

Service TypeはClusterIP（内部通信）とLoadBalancer（外部公開）を利用するのが基本です。
Kubernetesクラスタで発生する通信によって使い分けてください。

Kubernetesのサービスタイプ比較

	ClusterIP	NodePort	LoadBalancer	ExternalName
主な用途	クラスタ内部通信 (一時的な外部公開)	テスト用途 (一時的な外部公開)	本番環境の外部公開 (公式エンドポイント)	外部サービスへのエイリアス (特殊用途)
アクセス範囲	クラスタ内のみ	ノードIP:ノードPort	外部IPアドレス (クラウドLB割当)	DNS名のみ (実体へリダイレクト)
設定の複雑さ	シンプル 自動作成	やや複雑 ポート指定可能	中程度 クラウド連携必要	やや複雑 CNAME設定必要

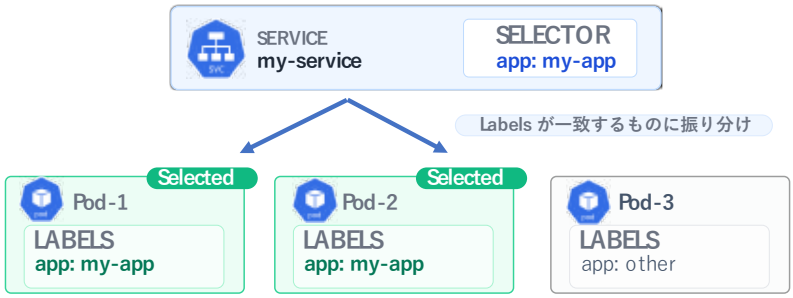
ラベルセレクター

リソースにラベル（key: value）を付与し、ラベルで対象を選ぶ仕組みです。
ServiceとPodなどを動的に関連付けします。その他リソースの紐づけにも使われる重要な技術です。

ラベル (Labels)
リソースに付けるタグ (Key: Value)

セレクタ (Selector)
ラベル条件で対象を選択する仕組み

推奨ラベル例
app: my-app
tier: frontend
version: v1.0
env: prod




```yaml
name: my-app
  replicas: 3
  ...
  labels:
    app: my-app
    tier: frontend
    env: prod
```

補足
- 複数ラベルを付与することも可能
- Service、Pod以外のリソース紐づけにもラベルセレクターが使われます

コンテナ技術システム構築　指導者向け資料

最後に

本単元は、Kubernetesを使ってコンテナ技術を習得する内容となっています。
一連のカリキュラムに取り組むことによって、仮想化から始まるコンテナやクラウドネイティブについての理解および技術を習得できます。Kubernetes特有の仕組みや内容も多く、座学だけではなかなか使い方を理解し覚えることが難しいことでしょう。

ぜひ指導者の皆様自身で演習に取り組み、各リソースの役割と挙動を理解してください。想定外の挙動をすることも多いので、トラブルシュートを通じてコンテナに対する理解を深めてください。
現場では、演習で扱った各要素を包括してシステムを構築していくことになりますが、まずこの単元でKubernetesや各リソースがどんな挙動をするのかを体得して説明できるようになりましょう。

【授業前に教員が確認すべきポイント】
・ 全10章の演習課題を自分で実施した
・ 各章の「コアコンセプト」を自分の言葉で説明できる
・kubectl の基本コマンドを実行できる
・YAMLファイルの基本構造を理解している
・ ハンズオン環境をセットアップし動作確認した

コンテナ技術システム構築　指導者向け資料

指導者が理解しておきたい知識１

基本概念

- コンテナと仮想マシンの違いを、カーネルの観点から説明できる
- コンテナを隔離する仕組み（namespace、cgroups、コンテナランタイム）を説明できる
- Dockerイメージとコンテナの違いを具体例で説明できる
- コンテナのレイヤー構造を説明できる
- マイクロサービスアーキテクチャのメリット・デメリットを説明できる

Docker関連

- Dockerfileおよび基本命令（FROM、RUN、COPY、CMD、ENTRYPOINT）を説明できる
- Docker Composeの役割と限界を説明できる
- Docker Hubとプライベートレジストリの違いを説明できる
- マルチステージビルドの目的を説明できる
- dockerignoreファイルの役割を説明できる

Kubernetes基礎

- Kubernetesのマスターノードとワーカーノードの役割を説明できる
- Pod、ReplicaSet、Deploymentの階層関係を図示できる
- 宣言的管理と命令的管理の違いを具体例で説明できる

コンテナ技術システム構築　指導者向け資料

指導者が理解しておきたい知識２

Kubernetesコンポーネント

- API Server、etcd、Scheduler、Controller Managerの役割を説明できる
- kubelet、kube-proxy、コンテナランタイムの役割を説明できる
- Serviceのタイプ（ClusterIP、NodePort、LoadBalancer）の違いを説明できる
- Ingressの役割とServiceとの違いを説明できる
- ConfigMapとSecretsの違いを説明できる
- リソース管理（request / limit、ResourceQuota、HPA）の仕組みを説明できる
- Probe（Liveness / Readiness/ Startup）の違いを説明できる
- Kubernetesコントローラー（Deployment、Statefulset、Daemonset）の違いを説明できる

現場でのユースケース

- コンテナがCI/CDパイプラインでどう使われるか説明できる
- GitOpsという考え方を説明できる
- Helmの役割（Kubernetesのパッケージマネージャー）を説明できる
- サービスメッシュ（Istio）の役割を説明できる

コンテナ技術システム構築　指導者向け資料

想定質問&解答例

初級編

Q1. コンテナと仮想マシンの違いは何ですか？
両方とも仮想化技術で実現したものです。仮想マシンはゲストOSごと起動しますが、コンテナはホストOSのカーネルを共有しプロセスレベルで隔離するため軽量で高速に起動します。

Q2. イメージとコンテナの違いが分かりません
イメージは実行可能なファイルシステムのスナップショット（設計図）、コンテナはイメージから起動した実行中のプロセス（実体）という違いがあります。

Q3. なぜPodという単位が必要なのですか？
密結合なコンテナ（Webサーバーとログコレクターなど）をまとめてIPアドレスやボリュームを共有し、同時に起動・停止するためです。

Q4. Docker ComposeとKubernetesは何が違うのですか？
ocker Composeは単一ホストの複数コンテナ管理、Kubernetesは複数ホストでの自動スケーリング・自動復旧・ゼロダウンタイムデプロイが可能。

Q5. Serviceは何のためにあるのですか？
PodのIPアドレスは一時的に付与されるものでPodが起動するたびに変わります。Serviceがラベルセレクターで対象Podを自動検出し、固定エンドポイントとロードバランシングを提供することができます。

コンテナ技術システム構築　指導者向け資料

想定質問&解答例

中級編

Q6. Kubernetesで「宣言的管理」とは何ですか？
「どうするか」ではなく「どうあるべきか」を定義し、Kubernetesが現在の状態と望ましい状態の差分を自動で埋め続ける（Reconciliation Loop）ようになります。マニフェスト（YAMLファイル）で定義します。

Q7. Podのスケジューリングはどう決まるのですか？
Schedulerがノードのリソース（CPU/メモリ）、Affinity/Anti-Affinity、Taint/Tolerationsを評価して最適なノードを自動選択します。

Q8. Kubernetesのネットワークはどうなっているのですか？
CNIプラグイン（Calico、Flannelなど）がオーバーレイネットワークやルーティングを使い、全PodがNATなしでフラットに通信できる環境を実現しています。

Q9. ReplicaSetとDeploymentの違いは何ですか？
ReplicaSetは指定数のPodレプリカを維持するだけ、Deploymentはその上位でローリングアップデートやロールバックなどライフサイクル管理を担当します。基本Deployementのみデプロイし、ReplicaSetとPodは自動生成に任せることが多いです。

Q10. ConfigMapとSecretsの違いは何ですか？
ConfigMapは機密性のない設定データ用、SecretsはパスワードやAPIキーなど機密データ用でBase64エンコード＋etcd暗号化可能といった違いがあります。

想定質問&解答例

上級編

Q11. コンテナのログはどこに保存されますか？
デフォルトではノードのローカルディスクだがPod削除で消えるため、実務ではFluentd/ELKスタック/Lokiなどで外部集中ログ管理が必須。

Q12. Kubernetesでステートフルなアプリケーション（データベース）を動かせますか？
StatefulSetを使えば可能です。永続的ID・順序保証・PersistentVolume・バックアップ戦略が必要です。
実務ではマネージドDBサービスの利用が推奨されます。

Q13. IngressとServiceの違いは何ですか？
ServiceはL4（IPアドレス・ポート）、IngressはL7（HTTPパス・ホスト名）でルーティングし、SSL終端やURLリライトも可能です。

Q14. Kubernetesのアップグレードはどうやるのですか？
3ヶ月ごとにリリースされる新バージョンをスキップせず順次適用します。マスターノード→ワーカーノードの順で、マネージドサービス利用が実務では推奨されます。アプリケーションのパッケージのアップグレードを伴う場合があるため、アップグレード戦略も考える必要があります。

令和7年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」

AWSクラウドネイティブ基盤実装

― 指導者向け資料 ―

情報技術者の技能アップデートのためのリカレント教育推進事業

AWSクラウドネイティブ基盤実装　指導者向け資料

はじめに

AWSを利用してクラウドネイティブなシステムを実装する単元です。
座学は最低限で、主に演習に取り組んで頂く内容となっています。

クラウドネイティブの要素が多く含まれています。
ぜひクラウドネイティブの要素を意識しながら、一つひとつ演習に取り組んでください。

演習は、主にIaC（CloudFormation）を利用して進める形をとっており、
GUIと比較して、準備に時間が掛かり、難易度も高くなっています。

ですが、開発現場ではインフラもアプリケーションも、IaC・CI/CDによる自動化が主流となっています。
IaC・CI/CDの体得に繋がりますので、ぜひ積極的に演習に取り組んで頂きたいです。

指導者の皆様におかれましては、IaCやCI/CDの仕組みやコード内容をご理解いただき、
必要に応じてトラブルシュートできるよう、躓きやすいポイントをフォローできるようにしましょう。

AWSクラウドネイティブ基盤実装　指導者向け資料

本教材の位置づけ

クラウドネイティブに関する幅広い領域について概念や仕組みを説明する単元です。
仮想化やコンテナの基礎を理解していることが前提となります。

分類	入門	基本	応用・実践
知識	仮想化技術	クラウドネイティブ概論	
技術習得	コンテナ技術基礎	コンテナ技術システム開発	**AWSクラウドネイティブ基盤実装** コンテナ・クラウドセキュリティ

AWSクラウドネイティブ基盤実装　指導者向け資料

演習の進め方 1

AWS CDKを活用して、段階的にクラウドネイティブなシステムを構築します。
演習は全部で4つあります。

	関連するクラウドネイティブな要素
1　三層アーキテクチャのWebアプリケーション構築 AWSを利用し、三層アーキテクチャ構成のWebアプリケーションを構築します。 環境構築はIaCで自動化し、開発からデプロイまでを一貫して体得します。	クラウド、コンテナ、サーバレス、Git、 オーケストレーション、CI/CD、IaC
2　ロードバランサーのHTTPS移行 HTTP対応のロードバランサーをHTTPSに移行します。 Webアプリケーションはそのままで、システムをセキュアにします。	クラウド、セキュリティ、IaC
3　CI/CDパイプラインのユニットテスト追加 既存の CI/CDパイプラインを拡張して、テストステージを追加します。 テストも含めて自動化し、品質ゲートを追加します。	クラウド、Git、CI/CD
4　総合演習 これまでの演習を踏まえて、新規Webアプリケーションを構築します。 IaCを拡張し、複数アプリケーションの構築に耐えうる形にします。	クラウド、コンテナ、サーバレス、Git オーケストレーション、CICD、IaC

AWSクラウドネイティブ基盤実装　指導者向け資料

演習の進め方 2

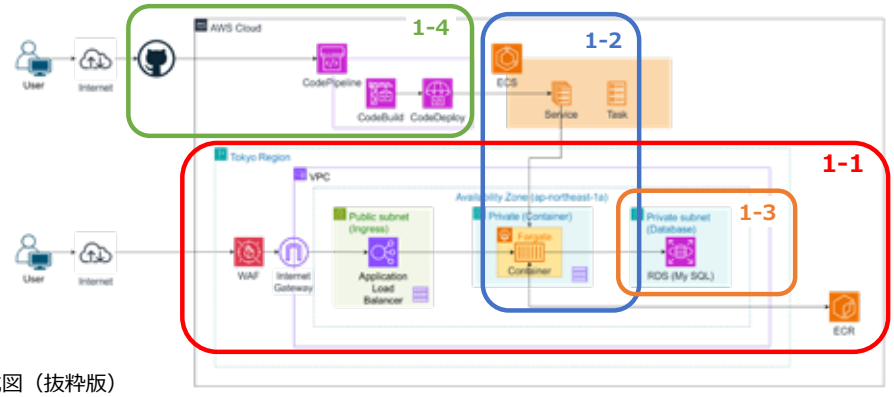
AWS CDKを活用して、段階的にクラウドネイティブなシステムを構築します。
演習は全部で4つあります。演習1については大きく5段階に分けて進めていきます。

1-0　AWS CDKの準備	開発環境をセットアップし、CDKを初期化 CloudShell/CDK/CloudFormationなど
1-1　ネットワーク構築	セキュアなネットワーク基盤を構築 VPC/サブネット/SG/ALB/WAF
1-2　コンテナ構築	リポジトリ、コンテナ基盤を構築し、アプリケーションをデプロイ ECR/ECS/Fargate
1-3　DB構築	DB基盤を構築し、データ連動可能なシステムを構築 RDS/SecretsManager
1-4　CI/CDパイプライン構築	GitHubと連動したパイプラインを構築し、B/Gデプロイを実現 GitHub/CodePipeline/CodeBuild/CodeDeploy

演習内容と全体構成図の関係性

演習1と全体構成図を紐づけると、以下の流れでシステムを構築します。

1-1 **ネットワーク構築**（VPC・サブネットなどのネットワーク基盤およびロードバランサー、FWの実装）
1-2 **コンテナ構築**（ECS Fargate、ECRを利用し、サーバレスなコンテナ基盤およびアプリケーションをデプロイ）
1-3 **DB構築**（RDSの構築およびアプリケーションが参照するデータの投入）
1-4 **CI/CDパイプライン構築**（GitHub、Codeシリーズを利用し、CI/CDパイプラインでのB/Gデプロイを実現）



全体構成図（抜粋版）

AWSクラウドネイティブ基盤実装　指導者向け資料

参考リンク

公式ドキュメント
- AWS公式ドキュメント
  https://docs.aws.amazon.com/ja_jp/
- GitHub公式ドキュメント
  https://docs.github.com/ja

学習リソース
- 演習解答（GitHub）
  https://github.com/cloudnative-prac901/answer_cloudnative
- 演習1～3：顧客情報管理アプリケーション（GitHub）
  https://github.com/cloudnative-prac901/customer-info
- 演習4：おみくじ占いアプリケーション（GitHub）
  https://github.com/cloudnative-prac901/fortune-telling

最後に

AWSクラウドネイティブ基盤実装についてポイントを説明してきました。
インフラからアプリケーションまでを自動化するというコンセプトの単元となっております。

内容からも分かるように、アプリ、インフラの境界が合間になってきており、フルスタックなエンジニアとしての能力が昨今求められるようになってきました。

演習を解くだけでも良いですが、さらにご自身で新しいアプリケーションを用意してデプロイしたり、他サービスを利用するなどしてみてください。この単元を切っ掛けとして、AWSを操作するほどにクラウドネイティブな知識や技術力が身についていくことを実感できると思います。

【授業前に教員が確認すべきポイント】
・各演習に取り組み、AWS CDKを使って2つのアプリケーションを立ち上げられているか
・各サービスの概念および設定について説明できるか
・三層アプリケーションやパイプラインについて説明でき、自分で実装できているか

AWSクラウドネイティブ基盤実装　指導者向け資料

指導者が理解しておきたい知識１

基本的なAWSサービス

- AWS CDKはCloudFormationのフロントエンドで、インフラをIaCで自動構築する
- ALBはL7ロードバランサーで、HTTP/HTTPSを解析できる
- WAFはL7ファイアーウォールで、ルールベースで境界防御できる
- ECS Fargateはサーバーレスだが、裏でAWS管理のEC2が動いている
- RDSはマネージドRDBMSで、バックアップ・パッチ適用が自動化されている

動作の仕組み

- cdk synthはCloudFormationテンプレートを生成、cdk deployはAWSにリソース作成
- ALBはリスナーでリクエストを受け、ターゲットグループに従って宛先に転送する
- ECRでコンテナイメージを登録、ECSから接続してイメージを取得可能
- ECSのタスク定義は設計図、サービスはタスクの維持、タスクは実体
- Secrets Managerのシークレットは実行時にAPIで取得することができる
- ACMの証明書は自動更新され、秘密鍵はAWS管理で触れない
- IAMで権限制御、サービスにIAMで権限を付与することにより、関連サービスを操作できる

AWSクラウドネイティブ基盤実装　指導者向け資料

指導者が理解しておきたい知識２

技術の比較

- ECS vs EKS：　ECSはAWS独自で簡易にコンテナを管理、EKSは移植性とカスタマイズ性が高い
- ALB vs NLB：　ALBはL7（パスベース通信）、NLBはL4(IPとポートのみ)
- Fargate vs EC2起動タイプ：　Fargateはサーバーレス、EC2は仮想サーバー。どちらかを指定してコンテナを起動（CaaS）
- CDK vs Terraform：　CDKはAWS特化、TerraformはマルチクラウドとHCL言語
- Secrets Manager vs Parameter Store：Secrets Managerは自動ローテーション対応

現場でのユースケース

- 三層アーキテクチャはWebアプリの標準構成として広く使われる
- ALBはパスベースルーティングで、マイクロサービスの入口として機能。HTTPは遮断し、HTTPSで通信させるのが一般的
- ECSはローリングアップデートでダウンタイムなしでデプロイ可能。Blue/Greenデプロイでのサービス切り替えも可能
- RDS Read Replicaは読み取り負荷分散に使われる
- ACM証明書はCloudFront、ALB、API Gatewayで利用可能
- コンテナのヘルスチェックは、コンテナ自体とALBからのダブルチェックで実現

AWSクラウドネイティブ基盤実装　指導者向け資料

指導者が理解しておきたい知識３

セキュリティ観点

- RDSはプライベートサブネットに配置し、インターネットに公開しない（サブネットの役割を整理）
- SG（セキュリティグループ）で許可する通信は最小限にする
- ECSタスクロールは最小権限の原則で、必要な権限のみ付与
- リクエストは、ALB＋WAFでセキュアな窓口を用意し、境界防御を機能させる
- Secrets Managerのシークレットは KMSで暗号化される
- GitHubとAWSはOIDC連携などで接続、外部サービスと接続する場合は最小権限で、接続元と操作権限を最小限に絞る
- ACM証明書はTLS 1.2以上を強制し、古いプロトコルを無効化
- IAMの基本的な考え方は最小権限、各サービスで実施できること（他サービスを操作）は最小限にする

AWSクラウドネイティブ基盤実装　指導者向け資料

想定質問&解答例

初級編

Q1. なぜ三層アーキテクチャに分けるのですか？1つのサーバーじゃダメなのですか？
スケール・保守・セキュリティを独立させるため、1つにまとめるより、役割を分割して3層で管理する方がに安全で柔軟です。

Q2. ALBなしでECSに直接アクセスさせるのはダメですか？
セキュリティ・負荷分散・ヘルスチェック・TLS管理のため ALB で一度リクエストを受ける必要があります。

Q3. なぜ通信はHTTPSが必要なのですか？
HTTPは盗聴・改ざん・なりすましの危険があり、HTTPSで通信を暗号化することが標準的な考え方です。
昨今Webサイトは基本HTTPSです。

Q4. CDKとCloudFormationの違いは？
CDKはコードで書くCloudFormation生成ツールで、CloudFormationはAWS純正のYAML/JSON定義です。

Q5. Fargateとはなんですか？EC2とどう違うのですか？
Fargateはサーバーレスなサービスでサーバの管理が不要です。
一方、EC2はOS管理が必要となるため、柔軟性はありますが運用コストがかかります。

AWSクラウドネイティブ基盤実装　指導者向け資料

想定質問&解答例

中級編

Q6. ECSのタスク定義とタスクの違いが分かりません
タスク定義は設計図、タスクはその設計図から起動された実コンテナです。
あとサービスはタスクを維持・管理してくれます。

Q7. なぜALB→ECSはHTTPのままで良いのですか？
VPC内の閉域ネットワークで安全性が高く、TLS終端をALBに任せられるためです。またECSはプライベートサブネットの配置し、ECSに対するアクセスもSGでALBおよび関連サービスに制限することで十分にセキュアになります。

Q8: ヘルスチェック失敗の原因は？
アプリ不備・SG設定漏れ・メモリ不足・DB不調といった原因が考えられます。
コンテナやALBのエラーログを見て原因解析をして対処してください。

Q9: タスクロールとタスク実行ロールの違いは？
タスクロールはアプリ用、タスク実行ロールはECSエージェント用の権限です。
IAMでロールに必要な操作権限を設定し、各サービスに紐づけてください。基本思想は最小権限です。

Q10. なぜSecrets Managerを使う必要があるのですか？
平文漏洩を防ぎ、暗号化・ローテーション・アクセス制御を実現するためです。
API経由で必要なシークレット情報を取得できるため、アプリケーション側でシークレット情報を持つ必要がなくなります。

想定質問&解答例

上級編

Q11. Blue/Greenデプロイとは？
旧版(v1)と新版(v2)のアプリケーションを並行稼働させ、安全に切替・即ロールバックを可能にする手法です。
ローリングアップデートに比べて安全なリリース方式です。ALB＋ECSの組み合わせで実現できます。

Q12. RDSのRead Replicaとは何ですか？
読み取り専用の複製DBで、読み込み負荷を分散するために使います。

Q13. なぜRDSをパブリックサブネットに置いてはいけないのですか？
DBをインターネット公開すると攻撃や漏洩リスクが極端に高まるためです。

Q14. WAFとはなんですか？ALBとはどう連携すればいいのですか？
WAFはWeb攻撃を防ぐL7防御で、ALBの前段に置いて不正リクエストを遮断します。

Q15: ECSのオートスケールはどう設定すればいいですか？
CPU利用率などのメトリクスを基に、Application Auto Scalingでタスク数を自動調整します。

Q16: ACM証明書の検証方法は？
DNSレコード追加（推奨）またはメール承認でドメイン所有を確認します。

Q17. ECSへのCI/CDデプロイの流れは？
コードpush → ビルド/ECR push → 新タスク定義作成 → ECS更新 → Blue/Greenで切替です。

令和7年度文部科学省委託
「専門職業人材の最新技能アップデートのための専修学校リカレント教育推進」

コンテナ・クラウドサービスのセキュリティ

― 指導者向け資料 ―

情報技術者の技能アップデートのためのリカレント教育推進事業

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

はじめに

Kubernetesや各種ツールを利用しながら、コンテナセキュリティの技術を体得する単元です。
座学は少なく、主に演習に取り組んで頂く内容となっています。

クラウドネイティブなコンテナシステムは様々なレイヤーで構成されている分、各レイヤーに対するセキュリティ対策が必要不可欠です。

クラウドネイティブで取り扱う領域が広いように、コンテナ・クラウドセキュリティの領域も広いです。
コンテナ・クラウドのライフサイクルから全体像を把握したうえで、各演習がどの部分に対するセキュリティなのかを把握し、受講者に説明・補足できるようにしてください。

また演習ですので、指導者の皆様自身で事前に取り組んで頂き、
演習の流れや躓きやすいポイントを確認頂いて、フォロー頂くようにお願いします。

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

本教材の位置づけ

コンテナ・クラウドサービスのセキュリティについて取り扱います。
クラウドネイティブの概念を理解し、Kubernetesの基本的な操作をできることが前提となります。

分類	入門	基本	応用・実践
知識	仮想化技術	クラウド ネイティブ概論	
技術習得	コンテナ 技術基礎	コンテナ技術 システム開発	AWSクラウドネイティブ基盤実装 **コンテナ・クラウドセキュリティ**

前提知識チェック

1. **Kubernetes操作**
   kubectlを使ったKubernetesのコンポーネントを操作できるか
2. **Linux基本コマンド**
   ls cd cat grep curl 等の操作に慣れているか
3. **クラウドネイティブのコンセプト**
   マイクロサービス、CI/CD、クラウドサービスなどの概念理解
4. **クラウド基盤のセキュリティ対策**
   VPC/SG、FW、IAM、暗号化などの実装経験があるか

演習の進め方

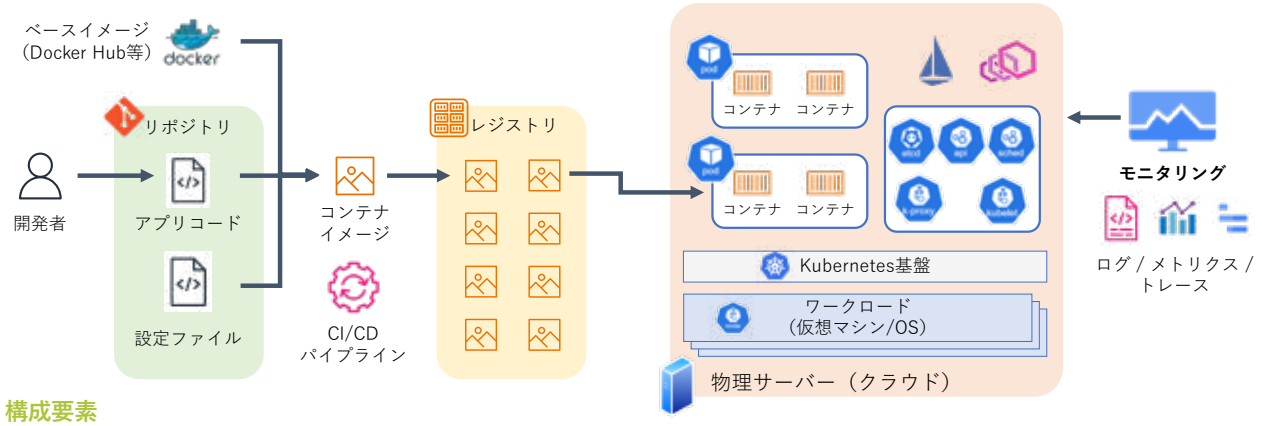
kubernetes環境を利用してコンテナセキュリティについて理解を深めましょう。
コンテナの構成要素別に、5つの演習を用意しています。

演習 1 **Kubernetes環境の診断と防御**	RBAC、NetworkPolicy、PSSの設定を通じて、Kubernetes環境における基本的なアクセス制御を体得します。 **関連要素：オーケストラ**
演習 2 **安全なコンテナ構築とランタイム防御**	脆弱なDockerfileを改善し、イメージ署名やseccompなどの適用により、イメージ・実行時の安全性を高めます。 **関連要素：コンテナ、イメージ、ホスト**
演習 3 **パイプラインへのセキュリティゲート導入**	GitHub Actionsを使ってコード解析・脆弱性スキャンを自動化することで、Shift-Leftを実現し、DevSecOpsの基本を学習します。 **関連要素：コード、イメージ、リポジトリ**
演習 4 **サービス間通信の暗号化とシークレット管理**	Istioを導入し、相互TLS（mTLS）でサービス間通信を暗号化し、AuthorizationPolicyによるアクセス制御を行います。 **関連要素：オーケストラ、コンテナ**
演習 5 **Kialiによる可視化と通信分析**	Kiali / Prometheus / Grafanaを導入し、サービス間通信を可視化することで、エラー率などの分析が出来るようにします。 **関連要素：オーケストラ、ホスト**

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

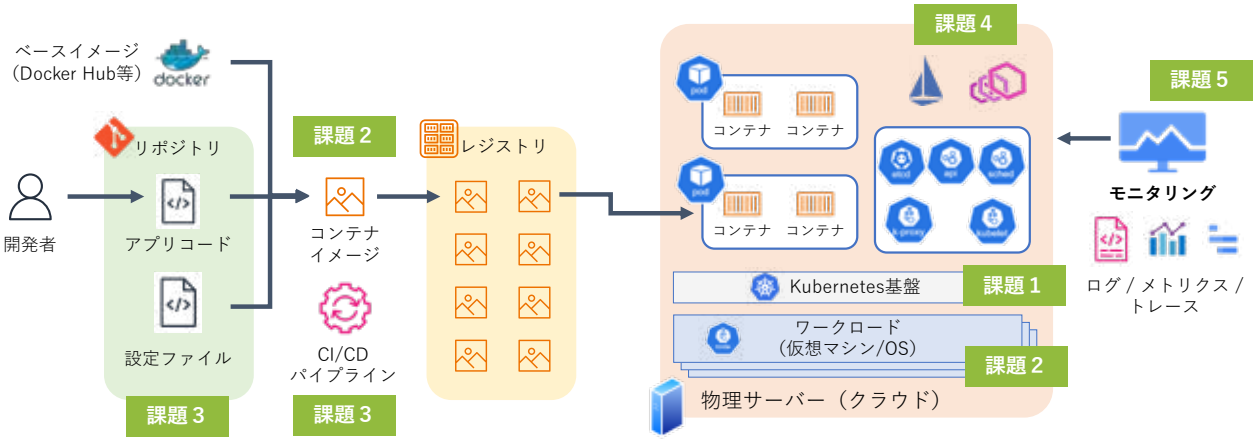
コンテナのライフサイクル

コンテナセキュリティを捉える前に、コンテナのライフサイクルを説明できるようにしましょう。
そのうえで、ライフサイクルと紐づけながら演習に取り組ませるようにしてください。



コンテナのライフサイクル（マッピング版）

コンテナセキュリティを捉える前に、コンテナのライフサイクルを説明できるようにしましょう。
そのうえで、ライフサイクルと課題を紐づけながら課題に取り組ませるようにしてください。



クラウドネイティブセキュリティの4C

4C（Four Cs of Cloud Native Security）はクラウドネイティブ環境の防御構造を4層で捉える多層防御モデルです。Cloud → Cluster → Container → Codeの4層で構成され、外側から内側まで**全ての層でセキュリティを確保するべき**と定義しています。設計段階からセキュリティを考慮するよう伝えてください。

Cloud（クラウド）

インフラ基盤（ネットワーク・IAM・暗号化）を
安全に構成し、外部からの侵入を防ぎます。

Cluster（クラスター）

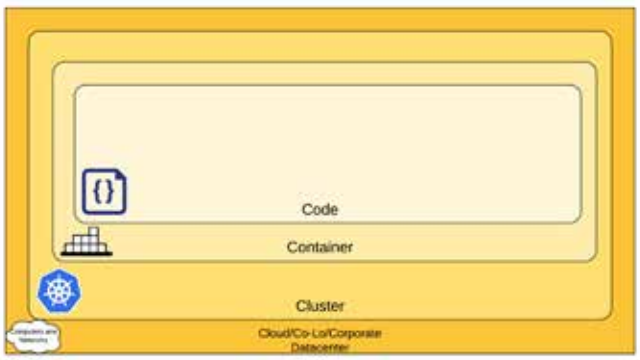
Kubernetes制御面を保護し、
アクセス権限や通信経路を最小化します。

Container（コンテナ）

安全なコンテナイメージと実行環境を維持し、
特権利用や脆弱な設定を防ぎます。

Code（コード）

安全なアプリ開発と依存管理を行い、
脆弱性や秘密情報の混入を防ぎます。



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

コンテナ・クラウドセキュリティの全体像とカバー範囲

4Cモデルでご説明した通り、多層防御でシステムを保護することが求められます。
今回の演習だけで、セキュリティ対策を網羅できる訳ではありません。
クラウドシステムを運用する場合は、更なる対策が必要であることを認識してください。



コンテナ・クラウドサービスのセキュリティ　指導者向け資料

本単元で扱わなかった重要セキュリティ

4Cモデルの全層で多層防御する必要があります。
本単元では扱っていないものの、他に理解しておくべきセキュリティも抑えておきましょう。

Cloud層：クラウドインフラ基盤のセキュリティ

技術	役割	本単元との関係
IAM	クラウドサービスの権限管理 ※各サービスの保護も別途必要	演習1：RBACと組み合わせて利用 演習3：GitHubとAWSの連携
VPC/サブネット/SG	仮想ネットワークの構築と境界防御 サブネットとSGによるネットワーク分割	演習1：EKSクラスター構築時に自動生成 EKSクラスターの外側の通信を制御
WAF	仮想ファイアーウォール 外部からのWeb攻撃をL7レイヤーで境界防御	演習1：NetworkPolicyによる通信制御 演習4：Itsioによる通信制御
ホストOS保護	仮想サーバの保護 OSパッチ、サーバへのアクセス制御	演習2：seccomp/AppArmor、非root化 によるホストへのアクセス

Code層：アプリケーションのセキュリティ

技術	役割	本単元との関係
認証・認可	API保護、ユーザー認証	演習4：PeerAuthenticationの サービス認証
ログ監査/ ランタイム監視	ログの全操作の記録と異常検知 ランタイムの正常性監視	演習5：Observability

クラウドサービスにおける責任共有モデル

利用するクラウドサービスのモデルによって責任境界が変わります。
自分たちの責任範囲を把握したうえで、セキュリティ対策することを念頭に置いてください。



クラウドサービスを提供しているデータセンターや物理サーバー、サービスはクラウド事業者が管理しています。
クラウド事業者でセキュリティ対策を実施している分、利用者は意識することなくサービスを利用できます。

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

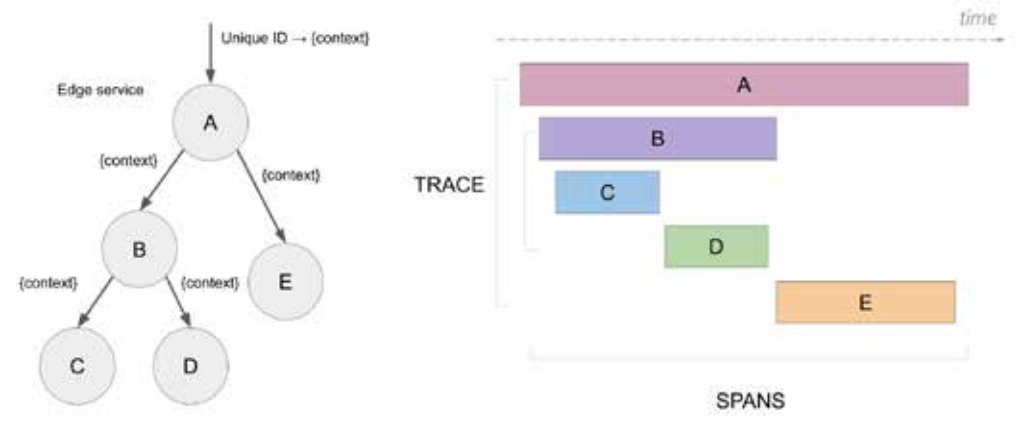
参考リンク

公式ドキュメント
- Kubernetesセキュリティ
  https://kubernetes.io/ja/docs/concepts/security/overview/
- コンテナセキュリティガイド（NIST SP800-190）
  https://csrc.nist.gov/pubs/sp/800/190/final
- Istio公式ドキュメント
  https://istio.io/latest/docs/

学習リソース
- 演習解答（GitHub）
  https://github.com/cloudnative-prac901/answer_container_security
- 演習用アプリケーション（GitHub）
  https://github.com/cloudnative-prac901/sample-app

トレース

分散システムで、1つのリクエストがサービス間をどのように流れたかを可視化する仕組みです。
TraceとSpanから、各サービスでの処理時間や障害時のエラー箇所を特定するのが容易になります。
代表的なツールとしては、jeagerやzipkin、OpenTelemetryなどがあります。



https://www.jaegertracing.io/docs/2.13/architecture/terminology/

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

最後に

本単元は「コンテナ環境の多層防御を実務レベルで設計・運用できる力」を育成します。
単にツールやマニフェストを適用して実現できることを確認するだけではなく、「なぜ必要か」「どう攻撃されるか」までセットで理解して説明できるようになることが重要です。

またこの単元で学べるのは、コンテナセキュリティの基礎です。
セキュリティも常に進化し続けていますので、「セキュリティに対する考え方（ゼロトラストベースで多層防御）」を身に付けて頂きつつ、最新のセキュリティ動向もキャッチアップしましょう。

【授業前に教員が確認すべきポイント】
・各演習の内容および、その対策が必要な背景を自分の言葉で説明できる
・演習を自分で実施している
・今回取り扱う演習だけでは網羅できない領域のセキュリティ対策について説明できる
・クラウドネイティブや仮想化・コンテナ技術について理解をしている
・kubectl の基本コマンドを実行できる
・YAMLファイルの基本構造を理解している

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

指導者が理解しておきたい知識 1

コンテナセキュリティ基礎

- ◘ コンテナが抱える5つのリスクを説明できる
  （カーネル共有、脆弱性継承、過剰権限、ネットワーク露出、シークレット情報混入）
- ◘ 4Cモデル(Cloud/Cluster/Container/Code)の各層で「なぜ対策が必要か」を説明できる
- ◘ 多層防御の原則（1層だけでは不十分）を説明できる
- ◘ 「公式イメージは安全」が誤解である理由を説明できる
- ◘ コンテナ内rootユーザーの危険性を説明できる
- ◘ サプライチェーン攻撃（ベースイメージ/依存パッケージ/悪意のイメージ）を説明できる
- ◘ ゼロトラストの原則（内部通信も信頼しない）を説明できる

オーケストレーションのセキュリティ

- ◘ RBACの3要素(ServiceAccount, RoleBinding, Role)の関係を説明できる
- ◘ ServiceAccountトークンの悪用シナリオを説明できる
- ◘ 最小権限の原則をRBACで実装する方法を説明できる
- ◘ NetworkPolicyの動作原理(デフォルト拒否、明示的許可)を説明できる
- ◘ PSSの3段階(Privileged/Baseline/Restricted)の違いと用途を説明できる
- ◘ Restrictedレベルの主な制約（runAsNonRoot, seccomp等）を説明できる

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

指導者が理解しておきたい知識 2

イメージ・ランタイムのセキュリティ

- ◘ Trivyの脆弱性検出の仕組み（CVE突合）を説明できる
- ◘ イメージ署名（Cosign ）の目的と検証フローを説明できる
- ◘ 最小イメージ（alpine/distroless ）のメリットを説明できる
- ◘ SeccompおよびAppArmorの役割を説明できる
- ◘ capabilitiesの削除（drop: [“ALL”] ）の必要性を説明できる

CICD・サービスメッシュのセキュリティ

- ◘ Shift-Leftの考え方を説明できる
- ◘ CI/CDでのセキュリティゲート配置を説明できる
- ◘ コード解析や脆弱性チェックの必要性を説明できる
- ◘ mTLS(相互TLS認証)の仕組みとIstioでの自動化を説明できる
- ◘ AuthorizationPolicyとNetworkPolicyの違い（L7 vs L3/L4 ）を説明できる

Observability

- ◘ Observabilityの概念、3つの柱（メトリクス・ログ・トレース）について説明できる
- ◘ Istio/Prometheus/Grafanaの構成で、メトリクスを取得・可視化するアーキテクチャを説明できる
- ◘ Kialiによる可視化およびトラブルシュートの大まかな進め方を理解している

想定質問&解答例

初級編

Q1. なぜコンテナはこんなにセキュリティ対策が必要なのですか？
コンテナはライフサイクルおよびアーキテクチャの観点から様々なリスクを持つため、従来の境界防御ではなく、4Cに基づいた多層防御の徹底が必須です。

Q2. 公式イメージは安全ですか？
公式イメージでも脆弱性を含むため、安全ではありません。どんなイメージも安全性は保証されていないため、イメージスキャンを実施し、重大な脆弱性をなくす必要があります。また新CVEが常に出るため、定期的に再スキャンすることも大切です。

Q3. Kubernetesを使えば自動でセキュアになりますか？
Kubernetesはデフォルトの設定が緩いためセキュアな環境ではありません。
最低限、明示的なRBAC・NetworkPolicy・PSS設定などのセキュリティ対策は必要です。

Q4. なぜ多層防御が必要なのですか？
1つが破られても他層で攻撃を阻止するためです。多層防御はクラウドセキュリティの基本です。

Q5. コンテナ内でrootユーザーで実行しても大丈夫ですか？
コンテナ内rootはホストと同等で危険です。攻撃された場合、ホストノードや他コンテナにまで影響が出うるため、非root実行を推奨します。

コンテナ・クラウドサービスのセキュリティ　指導者向け資料

想定質問&解答例

中級編

Q6. NetworkPolicyがDocker Desktopで動かないのはなぜですか？
Docker DesktopのCNIがNetworkPolicy非対応のためで、Calico等の導入が必要です。

Q7. ServiceAccountトークンはどこに保存されていますか？
Pod内の自動マウント領域に保存されています。SAを適用すればRBACでPodに対する操作を制限することができます。
設定に誤りがある場合は、API悪用に繋がるため、最小限の権限を付与するようにしてください。

Q8. Trivyはどうやって脆弱性を検出しているのですか？
すべてのレイヤーとパッケージ情報を解析し、保持しているCVEのデータベースと突合して脆弱性を検出します。

Q9. NetworkPolicyとIstio AuthorizationPolicyの違いは何ですか?
NetworkPolicyはL3/L4、AuthorizationPolicyはL7を制御するため役割が異なります。

Q10. PSSのRestrictedは厳しすぎませんか？
厳しいですが現代の標準です。Restrictedで動作しないサービスについても、段階的移行を前提に最終的にRestrictedを目指すべきです。

想定質問&解答例

上級編

Q11. CI/CDでスキャンすれば本番は安全ですか？
CI/CDだけでは不十分で、本番環境の継続監視とランタイム防御が必要です。
また新しい脆弱性が見つかる場合もあるため、定期的に既存パッケージの脆弱性をチェックし、必要に応じて本番環境のコンテナを更新してください。

Q12. Istioは小規模プロジェクトには不要ですか？
小規模では必須ではありませんが、mTLS自動化だけでも大きな価値があります。
サービスによってはマイクロサービス化が難しい場合もあるため、その時は利用しないという選択肢もありです。

Q13. 可観測性を高めるにはどうしたらいいですか？
ログ・メトリクス・トレースを柱として、Prometheus、Grafana、Kialiなどで各情報を収集して可視化するツールを整えてください。ログはELK、メトリクスはPrometheus＋Grafana、トレースはjeagerなどのツールを使います。

Q14. ゼロトラストとは具体的に何をすればいいですか？
全通信の認証・暗号化・最小権限化・監査を行い「内部も信頼しない」仕組みを作ることです。
認証認可、mTLS通信、IAM、定期的な監査などを実現するツールおよび設定を追加しましょう。

令和７年度「専門職業人材の最新技能アップデートのための専修学校リカレント教育（リ・スキリング）推進事業」
情報技術者の技能アップデートのためのリカレント教育推進事業

指導者向け研修プログラム資料

令和８年２月
一般社団法人全国専門学校情報教育協会
〒164-0003 東京都中野区東中野 1-57-8 辻沢ビル３F
電話：03-5332-5081 FAX.03-5332-5083

●本書の内容を無断で転記、掲載することは禁じます。