NIST Special Publication 800-190

アプリケーションコンテナセキュリティガイド

Murugiah Souppaya
John Morello
Karen Scarfone

本書は、以下より無料で利用可能である：
https://doi.org/10.6028/NIST.SP.800-190

C O M P U T E R S E C U R I T Y

この文書は以下の団体によって翻訳監修されています

IPA 独立行政法人情報処理推進機構
Information-technology Promotion Agency, Japan

本文書は、原典に沿ってできるだけ忠実に翻訳するよう努めていますが、完全性、正確性を保証するものではありません。

翻訳監修主体は、本文書に記載されている情報より生じる損失または損害に対して、いかなる人物あるいは団体についても責任を負うものではありません。

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

NIST Special Publication 800-190
アプリケーションコンテナセキュリティガイド

Murugiah Souppaya
Computer Security Division
Information Technology Laboratory

John Morello
Twistlock
Baton Rouge, Louisiana

Karen Scarfone
Scarfone Cybersecurity
Clifton, Virginia

本書は、以下より無料で利用可能である：
https://doi.org/10.6028/NIST.SP.800-190

2017 年 9 月

米国商務省
Wilbur L. Ross, Jr., Secretary

米国国立標準技術研究所
Kent Rochford, Acting Under Secretary of Commerce for Standards and Technology and Acting Director

発行機関

本文書は、連邦情報セキュリティ近代化法（FISMA：Federal Information Security Modernization Act）2014 年、合衆国法典（U.S. Code）第 44 編 第 3541 条 等、および公法（P.L.）113 条-283 条に基づく法的責任により、米国国立標準技術研究所（NIST： National Institute of Standards and Technology、以下、NIST と称す）によって作成された。NIST は、連邦政府情報システムの最低限の要求事項を含む情報セキュリティ標準及びガイドラインを策定する責務があるが、このような標準及びガイドラインを国家安全保障システムに適用するには、このようなシステムについての政策的権限を有する適切な連邦政府機関の明確な承認が必要となる。このガイドラインは、行政管理予算庁（OMB：Office of Management and Budget）による通達（Circular）A-130 号の要求事項と一致している。

本文書のいかなる内容も、商務長官が法的権限に基づき連邦政府に対して義務及び拘束力を与えた標準及びガイドラインを否定するものではない、また、これらのガイドラインは、商務長官、行政管理予算局長、または他のすべての連邦政府当局者の既存の権威に変更を加えたり、これらに取って代わるものと解釈したりしてはならない。本文書は、非政府組織が自由に使用することもでき、米国における著作権の対象ではないが、NIST に帰属する。

National Institute of Standards and Technology Special Publication 800-190
Natl. Inst. Stand. Technol. Spec. Publ. 800-190, 63 pages (September 2017)
CODEN: NSPUE2

本書は、以下から無料で利用可能である：
https://doi.org/10.6028/NIST.SP.800-190

本文書中で特定される商業的組織、機器、または資料は、実験的な手順や概念を適切に説明するためのものである。このような特定は、NIST による推奨または承認を意味するものではなく、これらの組織、機器、または資料が、必ずしもその目的のために利用可能な最善のものであることを意味しているわけではない。

本文書には、NIST に与えられた法的責任に従って現在策定中の他の文書への参照がある場合がある。本書に記載されている情報は、概念及び方法論を含め、そのような関連文書が完成する前であっても、連邦政府機関によって使用される可能性がある。したがって、それぞれの文書が完成するまで、現在の要求事項、ガイドライン、及び手順は、存在する限り効力を有する。連邦政府機関は、計画及び移行の目的のために、NIST によるこれらの新しい文書の策定を綿密に従うことを希望するかもしれない。

パブリックコメント募集期間中に、組織がすべてのドラフト文書をレビューし、NIST へフィードバックを提供することを奨励する。上記以外の多くの NIST サイバーセキュリティ関連文書は、https://csrc.nist.gov/publications において入手可能である。

本文書に対するコメントは以下に提出できます：

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930 Email:
800-190comments@nist.gov

すべてのコメントは、連邦情報公開法（FOIA：Freedom of Information Act）の下で開示の対象となる。

コンピュータシステムの技術に関する報告書

NIST の情報技術ラボラトリ（ITL：Information Technology Laboratory）は、米国の度量衡と標準規格に関する基盤において技術的リーダーシップを提供することにより、米国の経済及び社会福祉に貢献している。ITL は、テストの開発、テスト技法の開発、参照データの作成、概念実証の実施及び技術的分析を通じて、情報技術（IT）の開発と生産的利用の発展に努めている。ITL の責務には、連邦政府の情報システムにおいて、国家安全保障に関連する情報以外の情報に対する費用対効果の高いセキュリティとプライバシーを実現するための、技術面、物理面、管理面及び運用面での標準規格及びガイドラインを策定することが含まれる。Special Publication 800 シリーズでは、ITL の学術研究、ガイドラインについて、情報システムセキュリティ及び産業界、政府機関及び学術機関とのその共同活動における支援活動の取り組みとともに報告する。

要旨

コンテナとも呼ばれる、アプリケーションコンテナ技術は、オペレーティングシステムの仮想化とアプリケーションソフトウェアのパッケージングを組み合わせた一つの形態である。コンテナは、アプリケーションをパッケージ化して実行するための移動可能で再利用可能で自動化可能な手段を提供する。本文書では、コンテナの使用に関連する潜在的なセキュリティ上の懸念事項を説明し、これらの懸念事項に対処するための推奨事項を提供する。

キーワード

アプリケーション；アプリケーションコンテナ；アプリケーションソフトウェアパッケージング；コンテナ；コンテナセキュリティ；分離；オペレーティングシステムの仮想化；仮想化

謝辞

著者は、本文書のドラフト版をレビューし、策定中に技術的な内容に貢献してくれた同僚、特に Intel Corporation の Raghuram Yeluri 氏、Cisco Systems, Inc.の Paul Cichonski 氏、NIST の Michael Bartock 氏と Jeffrey Cichonski 氏、Edward Siewick 氏に感謝の意を表する。また、Docker、Motorola Solutions、StackRox、米国移民局（USCIS）、米国陸軍など、パブリックコメント期間中にフィードバックを提供してくれた組織にも感謝を表する。

対象読者

本文書の対象読者は、システムおよびセキュリティ管理者、セキュリティプログラム管理者、情報システムセキュリティ幹部、アプリケーション開発者、およびアプリケーションコンテナ技術のセキュリティに責任を持つ、あるいは関心を持つその他の人々である。

本文書は、読者がオペレーティングシステム、ネットワーク、セキュリティの専門知識に加え、仮想化技術（ハイパーバイザや仮想マシン）の専門知識を持っていることを前提としている。アプリケーションコンテナ技術の性質は常に変化しているため、読者の皆様には、より最新かつ詳細な情報を得るために、本書に記載されているものを含む、他のリソースを利用することを推奨する。

概要

オペレーティングシステム（OS）の仮想化は、各アプリケーションそれぞれに、OS の仮想化されたビューを提供する。したがって、各アプリケーションは、サーバ上の他のすべてのアプリケーションから分離しており、自身のアプリケーションだけが見え、他のアプリケーションには影響を与えない。近年、OS の仮想化は、使いやすさの向上と開発者のアジリティを主な利点として重視するようになったため、ますます普及してきている。今日の OS 仮想化技術は、アプリケーション（アプリ）をパッケージして実行するために、移動可能で、再利用でき、自動化できる方法を提供することに主に焦点が当たっている。**アプリケーションコンテナ**、または単に**コンテナ**という用語は、これらの技術について言及するために頻繁に使用されている。

本文書の目的は、コンテナ技術に関連するセキュリティ上の懸念事項を説明し、コンテナの計画、実装、および保守を行う際に、これらの懸念事項に対処するための実用的な推奨事項を提供することである。推奨事項の多くは、図 1 に示すコンテナ技術アーキテクチャ内の特定のコンポーネントまたは階層に固有のものである。

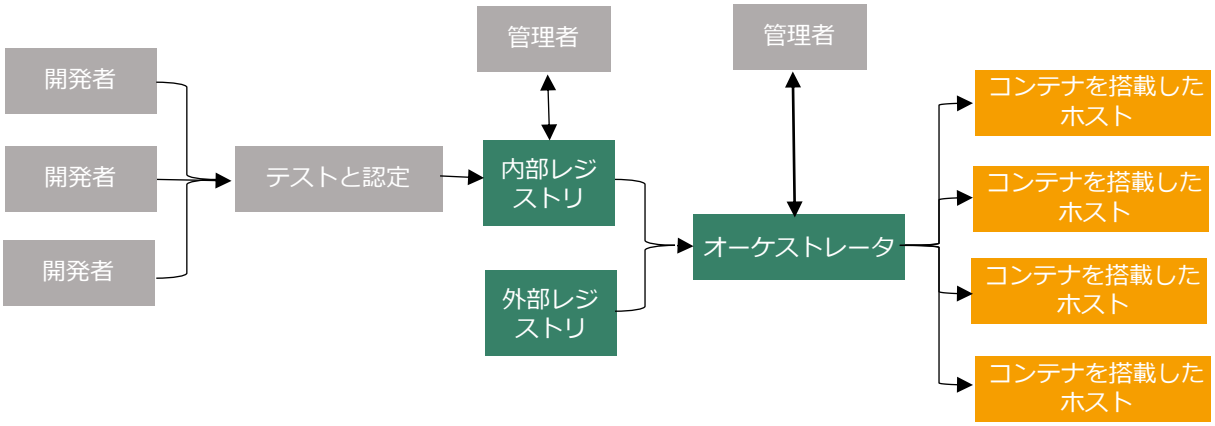

図 1：コンテナ技術アーキテクチャの階層とコンポーネント

組織は、これらの推奨事項に従って、コンテナ技術の実装と使用のセキュリティを確保することが望ましい。

コンテナによって可能となったアプリケーションを開発、実行、サポートする新しい手段を後押しするために、組織の運用文化と技術的プロセスを調整する。

コンテナ技術の導入は、組織内の既存の文化やソフトウェア開発方法論を混乱させる可能性がある。従来の開発手法、パッチ適用技術、およびシステムのアップグレードプロセスは、コンテナ化された環境にそのまま適用出来ない場合があり、従業員が進んで新しいモデルに適応することが重要である。本文書で説明するように、組織は、コンテナ内でアプリをセキュアに構築して運用するために推奨されるプラクティスを採用するよう、スタッフを促し、コンテナをうまく活用するために既存の手順を再検討することが望ましい。また、ソフトウェア開発ライフサイクルに関わるすべての人に対して、技術と運用アプローチの両方をカバーする教育とトレーニングを提供することが望ましい。

アタックサーフェスを減らすために、汎用ホスト OS ではなく、コンテナ専用ホスト OS を使用する。

コンテナ専用ホスト OS は、他のすべてのサービスと機能を無効にし、読み取り専用ファイルシステムやその他のセキュリティ強化のためのプラクティスを採用した、コンテナのみを実行するよう明示的に設計された最小限の OS である。コンテナ専用ホスト OS を使用する場合、アタックサーフェスは通常、汎用ホスト OS よりもはるかに小さいため、コンテナ専用ホスト OS が攻撃・侵害される機会は少なくなる。したがって、組織は可能な限りコンテナ専用ホスト OS を使用して、リスクを軽減することが望ましい。ただし、コンテナ専用ホスト OS は、時間の経過とともに修正が必要な脆弱性を持つことに注意が必要である。

単一のホスト OS カーネル上で、同じ目的、機微性および脅威に対する態勢を持つグループコンテナのみを使用し、更なる多層防御を可能とする。

ほとんどのコンテナプラットフォームは、コンテナ同士やホスト OS からコンテナを分離する効果的な役割を果たしているが、同じホスト OS で異なる機微性レベルのアプリを一緒に実行することは、不必要なリスクとなる可能性がある。コンテナを目的、機微性、および脅威に対する態勢によってセグメント化することで、更なる多層防御が可能となる。このようにコンテナをグループ化することで、組織は、グループ化された中の一つのグループを侵害した攻撃者が他のグループに侵害を拡大することをより困難にする。これにより、侵害を検知して封じ込める可能性が高まり、キャッシュや一時ファイル用にマウントされたローカルボリュームなどの残留データを、セキュリティゾーン内に確実に留める。

何百ものホストと何千ものコンテナを持つ大規模な環境では、実用的な運用のために、このグループ化を自動化する必要がある。幸い、コンテナ技術には通常、アプリをまとめてグループ化できるという概念が含まれており、コンテナセキュリティツールは、アプリ全体に渡ってセキュリティポリシーを強制するために、コンテナ名やラベルなどの属性を使用することができる。

コンテナ専用の脆弱性管理ツールとイメージのプロセスを採用し、侵害を防ぐ。

従来の脆弱性管理ツールは、ホストの耐久性やアプリの更新メカニズムや頻度について多くの想定をしているが、それはコンテナ化されたモデルとは根本的に異なる。例えば、これらのツールは、特定のサーバが一貫した一連のアプリを同じ時間を使用して実行している、と想定していることが多い。しかし実際は、リソースの可用性に基づいて、様々なアプリケーションコンテナが、様々なサーバで各々の任意の時間で実行されている可能性がある。さらに、従来のツールでは、コンテナ内の脆弱性を検知できないことが多いため、誤った安心感をもたらす。組織は、宣言的で段階的なビルドアプローチと、コンテナとイメージのイミュータブル（immutable）を設計に取り入れた、より実用的で信頼性の高い結果を提供するツールを使用することが望ましい。

これらのツールとプロセスでは、イメージソフトウェアの脆弱性と構成設定の両方を考慮することが望ましい。組織は、イメージのセキュアな設定のベストプラクティスへの準拠を検証し、強制するためのツールとプロセスを採用することが望ましい。これには、各イメージのコンプライアンス状態の一元化されたレポートとモニタリング、及びコンプライアンスに準拠していないイメージの実行を防止することが含まれる。

トラステッドコンピューティングの基盤を提供するために、ハードウェアによる対策を検討する。

セキュリティは、コンテナ技術のすべての階層に及ぶことが望ましい。これを実現する現在の手段は、業界標準の Trusted Platform Module (TPM) のようなハードウェアの信頼の基点をセキュリティの基準として使うことである。ハードウェアの信頼の基点内には、ホストのファームウェア、ソフトウェア、および構成データの計測値が保存されている。ホストを起動する前に、保存されている計測値と現在の計測値を対照して検証することで、ホストが信頼できることが保証される。ハードウェアに根差した信頼のチェーンを OS カーネルと OS コンポーネントに拡張して、ブートメカニズム、システムイメージ、コンテナランタイム、コンテナイメージの暗号技術による検証を可能にすることができるトラステッドコンピューティングは、コンテナの構築、実行、オーケストレーション、および管理するためのセキュアな方法を提供する。

コンテナに対応したランタイム防御ツールを使用する。

実行時のコンテナを狙った脅威を防止、検知、および対応できる専用のコンテナセキュリティソリューションをデプロイして使用する。侵入防止システム（IPS）やウェブアプリケーションファイアウォール（WAF）などの従来のセキュリティソリューションは、多くの場合、コンテナを適切に保護できない。これらのソリューションは、コンテナのスケールに応じて運用したり、コンテナ環境の変更の頻度を管理したり、コンテナのアクティビティを可視化することができない場合がある。コンテナ環境をモニタリングし、コンテナ内の異常なアクティビティや悪意のあるアクティビティを正確に検知できるコンテナネイティブのセキュリティソリューションを利用する。

目次

付録

図および表

1 はじめに

1.1 目的とスコープ

本文書の目的は、アプリケーションコンテナ技術に関連するセキュリティ上の懸念事項を説明し、コンテナの計画、実装、保守を行う際に、それらの懸念事項に対処するための実践的な推奨事項を提供することである。コンテナのいくつかの側面は技術によって異なる場合があるが、本文書の推奨事項は、ほとんど、又はすべてのアプリケーションコンテナ技術に適用することを意図している。

仮想マシンなど、アプリケーションコンテナ以外のすべての仮想化の形態は、本文書の範囲外である。

アプリケーションコンテナ技術に加えて、「コンテナ」という用語は、モバイル機器上で企業のデータと個人のデータを分離するソフトウェアや、デスクトップ OS 上でアプリケーションを互いに分離するために使用される可能性のあるソフトウェアなどの概念を指すために使用される。これらは、アプリケーションコンテナ技術といくつかの属性を共有しているかもしれないが、本文書では対象外とする。

本文書は、読者がすでにアプリケーションコンテナ技術をサポートし、関連する技術に精通していることを前提としている。これらには、以下のものが含まれる。

- ハードウェア、ハイパーバイザ、オペレーティングシステムを含む、アプリケーションコンテナ技術の下にある階層（レイヤ）
- コンテナ内のアプリケーションを使用する管理ツール
- コンテナ内のアプリケーションやコンテナ自体を管理するために使用される管理者用エンドポイント

付録 A には、これらの技術のセキュリティに関する情報を提供するリソースへのリンクが含まれている。3 節と 4 節では、コンテナ専用のオペレーティングシステムのセキュリティ上の考慮事項に関する追加情報を記載している。上記のセキュリティ技術に関するこれ以上の説明は、本文書の対象外である。

1.2 本文書の構成

本文書の残りの部分は、以下の節と付録で構成されている。

- 2 節では、コンテナの技術的機能、技術アーキテクチャ、および用途などを紹介する。
- 3 節では、アプリケーションコンテナ技術のコアコンポーネントに対する主要なリスクについて説明する。
- 4 節では、3 節で識別されたリスクに対する対策について提言する。
- 5 節では、コンテナの脅威のシナリオの例を定義する。
- 6 節では、コンテナ技術の計画、実装、運用、保守のための実用的な情報を提示する。
- 7 節では、本文書の結論を述べる。
- 付録 A には、コンテナ技術の非コアコンポーネントを保護するための NIST のリソースをリストアップする。
- 付録 B では、アプリケーションコンテナ技術に最も関連性の高い NIST Special Publication 800-53 のセキュリティ管理と、NIST Cybersecurity Framework のサブカテゴリをリストアップし、それぞれの関連性を説明する。

- 付録 C では、本文書の頭字語と略語のリストを示す。
- 付録 D では、本文書から選択された用語の用語集を示す。
- 付録 E には、本文書の参考文献のリストを示す。

2 アプリケーションコンテナの概要

本節では、サーバアプリケーション（アプリ）に使われるコンテナの概要について説明する。最初に、本文書の残りの部分を理解するために必要なアプリケーション仮想化とコンテナの基本的な概念を定義する。次に、コンテナが、その上で実行するオペレーティングシステムとどのように関係しているのかについて説明する。そして、コンテナ技術の全体的なアーキテクチャを説明し、コンテナの実装で一般的に見られるすべての主要なコンポーネントを定義し、コンポーネント間のワークフローを説明する。最後に、コンテナの一般的な用途について説明する。

2.1 アプリケーション仮想化とコンテナの基本概念

NIST Special Publication (SP) 800-125 [1]では、**仮想化**を「他のソフトウェアが実行されるソフトウェアおよび／またはハードウェアのシミュレーション」と定義している。仮想化は長年にわたって使用されてきたが、クラウドコンピューティングを可能にしたことで最も良く知られている。クラウド環境では、**ハードウェア仮想化**を利用して、1台の物理サーバ上で多数のオペレーティングシステム（OS）のインスタンスを実行し、各インスタンスを個別に維持する。これにより、ハードウェアをより効率的に使用でき、マルチテナンシがサポートされる。

ハードウェア仮想化では、各 OS インスタンスは仮想化されたハードウェアと関わり合う。**オペレーティングシステム仮想化**と呼ばれる仮想化の別の形態にも、1 つの実際の OS カーネルの上に、複数の仮想化 OS を提供するという、同様の概念がある。このアプローチはしばしば **OS コンテナ**と呼ばれ、2000 年代初頭以降、Solaris Zone や FreeBSD jails[1]から始まり、様々な OS コンテナの実装が存在している。2008 年に Linux でのサポートが開始され、Linux コンテナ (LXC) 技術がほぼすべての最新ディストリビューションに組み込まれ、利用できるようになった。OS コンテナは、複数のアプリケーションやサービスが共存する通常の OS のように動作する環境を提供するよう設計されているため、本文書のトピックであるアプリケーションコンテナとは異なる。

近年、アプリケーション仮想化は、その使いやすさが向上し、開発者のアジリティが主要な利点として一層重視されるようになっているため、ますます普及が進んでいる。**アプリケーション仮想化**では、同じ共有 OS カーネルが仮想的に複数の個別のアプリに提供される。OS コンポーネントは、各々のアプリのインスタンスをサーバ上の他のすべてのインスタンスから分離した状態にしておく。この場合、各々のアプリは OS と自分自身しか見えず、同じ OS カーネル上で実行されている可能性がある他のアプリから分離される。

OS の仮想化とアプリケーション仮想化の主な違いは、アプリケーション仮想化では、各仮想インスタンスは通常 1 つのアプリのみを実行するという点である。今日のアプリケーション仮想化技術は、アプリをパッケージ化して実行するための、移動可能で再利用可能で自動化可能な方法を提供することに主に焦点を当てている。**アプリケーションコンテナ**または単に**コンテナ**という用語は、これらの技術を指すために頻繁に使用される。この用語は、異なるコンテンツをグループ化し、それらを互いに分離する標準化された方法を提供する輸送用コンテナに例えられている。

アプリをいくつかの階層（ウェブ、アプリ、データベースなど）に分割し、それぞれの階層にサーバまたは VM があることが多い従来のアプリアーキテクチャとは異なり、コンテナアーキテクチャでは、アプリをより多くのコンポーネントに分割し、それぞれが単一の明確に定義された機能を持ち、通常は独自のコンテナ内で実行される。各々のアプリのコンポーネントは別々のコンテナで動作する。アプリケーションコンテナ技術では、一緒に動作してアプリを構成する一連のコンテナを、**マイクロサービス**と呼ぶ。このアプローチにより、アプリのデプロイは柔軟でスケーラ

[1] jails の概念の詳細については、https://www.freebsd.org/doc/handbook/jails.html. を参照。

ブルになる。また、機能がより自己完結的になるため、開発がより簡単となる。しかし、管理しなければならないオブジェクトが増えるため、アプリの管理、セキュリティツールとプロセスに問題が生じる可能性がある。

ほとんどのアプリケーションコンテナは、イミュータブルの概念を実装している。つまり、コンテナ自体はデプロイされても変更はされないステートレスなエンティティとして運用することが望ましい[2]。実行中のコンテナをアップグレードしたり、内容を変更したりする必要がある場合には、コンテナを単に破棄し、更新された新しいコンテナに置き換える。これにより、開発者やサポートエンジニアは、より速いペースでアプリに変更を加えて配信することができる。組織は、四半期ごとにアプリの新しいバージョンをデプロイしていたのが、毎週または毎日のように新しいコンポーネントをデプロイしても良くなる。イミュータブルは、コンテナとハードウェア仮想化の根本的な運用上の違いである。従来の VM は通常、その寿命を通して、デプロイ、再設定、アップグレードされるステートフルなエンティティとして実行される。従来のセキュリティツールとプロセスは、多くの場合は静的な運用を前提としていることが多く、コンテナ環境の変更の頻度に合わせて調整する必要があるかもしれない。

コンテナのイミュータブルは、データの永続性にも影響を与える。コンテナは、アプリと使用するデータを混在させるのではなく、分離の概念を重視している。データの永続性は、コンテナのルートファイルシステムへの単純な書き込みではなく、データベースやクラスタ対応の永続ボリュームなどの外部の永続データストアを使用して実現することが望ましい。コンテナが使用するデータは、コンテナ自体の外部に保存されることが望ましい。これにより、次のバージョンのアプリが、既存のバージョンを実行しているコンテナをリプレースする際、すべてのデータが新しいバージョンで引き続き使用できる。

最新のコンテナ技術は、開発チームと運用チームの緊密な連携を重視し、アプリの構築と実行の統合強化を目的とした開発と運用（DevOps）のプラクティスの採用とともに、大きく台頭してきた[3]。コンテナの移動可能で宣言的な性質は、開発環境、テスト環境、本番環境の間で優れた一貫性を持つことができるため、これらのプラクティスに特に適している。組織は多くの場合、継続的インテグレーションを利用して、ビルドプロセスによって直接コンテナにアプリを入れる。これにより、アプリのライフサイクルの最初から、ランタイム環境の一貫性が保証される。コンテナイメージ（コンテナの実行に必要なファイルを含むパッケージ）は、通常、マシンや環境間で移植できるように設計されており、開発ラボで作成したイメージは評価のためにテストラボに簡単に移動でき、次に本番環境にコピーして、変更を加えることなく実行できるようになっている。欠点は、コンテナの保護に使用されるセキュリティツールとプロセスが、特定のクラウドプロバイダ、ホスト OS、ネットワークトポロジ、または頻繁に変更される可能性があるコンテナランタイム環境といったその他の側面を想定していないということである。さらに重要なことは、これらのすべての環境において、また開発からテスト、本番までのアプリのライフサイクル全体にわたって、セキュリティが一貫していることが望ましい。

最近では、Docker [2] や rkt [3] などのプロジェクトが、OS コンポーネントの分離機能をより簡単に使用および拡張できるように設計された追加機能を提供している。コンテナ技術は、Windows Server 2016 以降の Windows プラットフォームでも利用できる。これらすべての実装の基本的なアーキテクチャは十分に一貫しているので、本文書では実装にとらわれずにコンテナについて詳細に説明できる。

---

[2] コンテナは不変性を実用的かつ現実的なものにしているが、それを必要としていないため、組織はそれを活用するために運用プラクティスを適応させる必要があることに注意。

[3] 本文書では、DevOps ペルソナによって実行されるタスクについて説明する。これらのペルソナへの言及は、厳密な肩書きやチームの組織構造ではなく、実行されるジョブタスクのタイプに焦点を当てている。

2.2 コンテナとホストオペレーティングシステム

コンテナ内でのアプリのデプロイは、多くの読者がすでに知っているハードウェア仮想化技術の仮想マシン（VM）内でのアプリのデプロイと比較することで容易に説明できる。図 2 は、左が VM のデプロイ、中央が VM なしのコンテナのデプロイ（「Bare Metal」上にインストールされている）、右が VM 内で動作するコンテナのデプロイを示している。

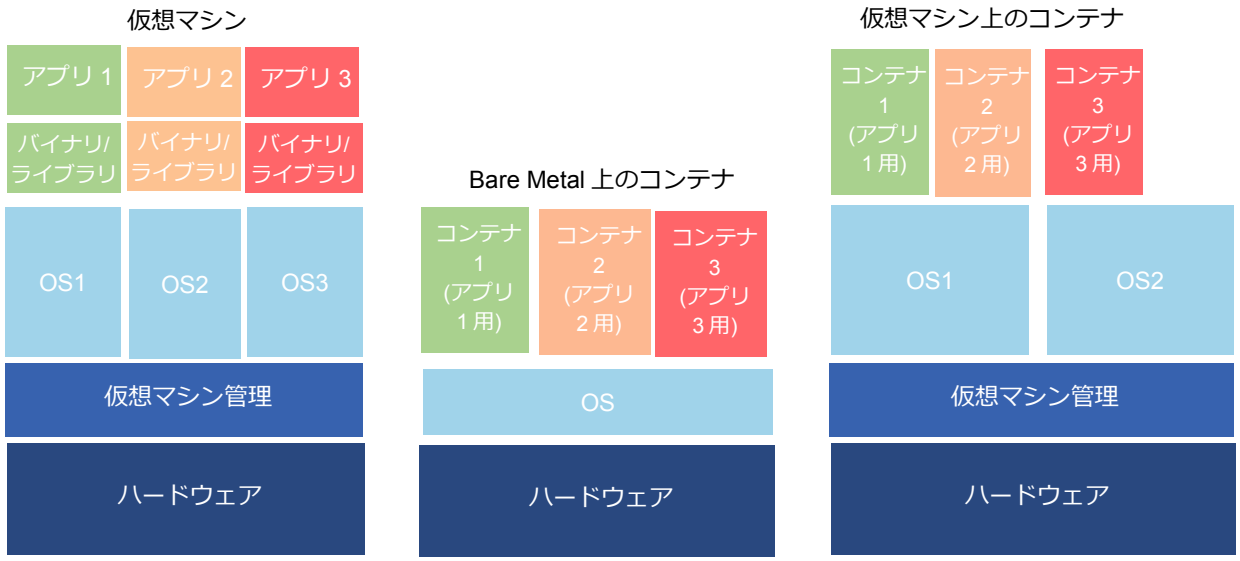

図 2：仮想マシンとコンテナのデプロイ

VM とコンテナの両方で、複数のアプリケーションが同じ物理インフラを共有できるが、それぞれ異なる分離方法を使用している。VM は、ハイパーバイザを使用して、VM 間でリソースをハードウェアレベルで分離する。各 VM は自分自身の仮想ハードウェアを認識し、さらに、アプリとそのデータに加えて完全なゲスト OS を包含する。VM を使用すると、Linux や Windows などの異なる OS が同じ物理ハードウェアを共有できる。

コンテナでは、複数のアプリが同じ OS カーネルインスタンスを共有しているが、互いに分離されている。OS カーネルは**ホストオペレーティングシステム**と呼ばれるものの一部である。ホスト OS はコンテナの下にあり、コンテナに OS 機能を提供する。コンテナは OS ファミリに固有のもので、Linux ホストは Linux 用に構築されたコンテナのみを実行でき、Windows ホストは Windows コンテナのみを実行できる。また、1 つの OS ファミリ用に構築されたコンテナは、そのファミリの最新の OS で実行することが望ましい。

コンテナの実行に使用されるホスト OS には、2 つの一般的なカテゴリがある。Red Hat Enterprise Linux、Ubuntu、Windows Server のような**汎用 OS** は、多くの種類のアプリを実行するために使用することができ、それらにコンテナ専用の機能を追加することができる。CoreOS Container Linux [4]、Project Atomic [5]、Google Container-Optimized OS [6]のような**コンテナ専用 OS** は、コンテナのみを実行するように明示的に設計された最小限の OS である。これらの OS には通常、パッケージマネージャが付属しておらず、コア管理ツールの一部しかなく、コンテナ外でのアプリの実行を能動的に阻止する。多くの場合、コンテナ専用 OS は、読み取り専用のファイルシステム構造を使用して、攻撃者がその中にデータを保持できる可能性を低減する。また、アプリの互換性についての懸念がほと

んどないので、簡素化されたアップグレードプロセスを利用している。

コンテナの実行に使用されるすべてのホスト OS は、各コンテナの環境を確立して維持するバイナリファイルを有しており、これは**コンテナランタイム**とも呼ばれている。コンテナランタイムは、複数の OS コンポーネントを調整し、リソースとリソースの使用を分離することで、各コンテナが自分専用の OS のビューを見て、同時に実行されている他のコンテナから分離されるようにする。実質的に、コンテナとホスト OS は、コンテナランタイムを通じてやり取りする。コンテナランタイムはまた、管理ツールとアプリケーションプログラムインタフェース（API）を提供し、DevOps 担当者などが特定のホスト上でコンテナを実行する方法を指定できるようにする。ランタイムによって、必要なすべての設定を手で作成する必要がなくなり、コンテナの起動、停止、運用のプロセスが簡素化される。ランタイムの例としては、Docker [2]、rkt [3]、Open Container Initiative Daemon [7]などがある。

コンテナランタイムが、ホスト OS が提供することを保証する技術的機能の例には、以下のようなものがある。

- **名前空間(namespace)の分離**によって、コンテナが操作できるリソースが制限される。これには、ファイルシステム、ネットワークインタフェース、プロセス間通信、ホスト名、ユーザ情報、プロセスが含まれる。名前空間の分離は、コンテナ内のアプリケーションとプロセスが、そのコンテナに割り当てられた物理的および仮想的なリソースのみを見ることを保証する。例えば、他の多くのコンテナが他のアプリを実行しているホスト上の、Apache を実行しているコンテナ内で 'ps -A' を実行すると、結果には httpd のみが表示される。名前空間の分離により、各コンテナには、固有のインタフェースや IP アドレスを含む独自のネットワークスタックが提供される。Linux 上のコンテナは、名前空間の分離を実現するためにマスクされたプロセス ID のような技術を使用するが、Windows ではオブジェクトの名前空間が使用される。
- **リソース割り当て**によって、一つのコンテナが消費できるホストのリソースの量が制限される。例えば、ホスト OS の総メモリが 10 ギガバイト（GB）の場合、9 つの個別のコンテナにそれぞれ 1 GB を割り当てることができる。どのコンテナも他のコンテナの操作を妨害してはいけないので、リソース割り当ては、各コンテナが割り当てられたリソースの量だけを利用できるようにする。Linux では、これは主にコントロールグループ（cgroups）[4]で実行されるが、Windows ではジョブオブジェクトが同様の目的を果たす。
- **ファイルシステム仮想化**によって、複数のコンテナが、他のコンテナのストレージにアクセスしたり変更したりせずに、同じ物理ストレージを共有できる。名前空間の分離と似ているのは間違いないが、コンテナがコピーオンライト（copy-on-write）のような技術を通じてホストのストレージを効率的に使用するよう最適化することが多いため、ファイルシステム仮想化は、独立して呼び出される。例えば、同じイメージを使用する複数のコンテナが単一のホストで Apache を実行している場合、ファイルシステム仮想化により、httpd バイナリのコピーが 1 つだけディスクに保存される。コンテナの一つがそれ自身の中でファイルを変更する場合、変更されたビットだけがディスクに書き込まれ、その変更は実行したコンテナにのみ表示される。Linux では、これらの機能は、Advanced Multi-Layered Unification Filesystem (AUFS)のような技術によって提供される。また、Windows では NT File System (NTFS)の一つの拡張機能となっている。

コンテナの技術的機能は、ホスト OS ファミリによって異なる。コンテナは基本的に、各アプリに単一の OS の固有のビューを提供するメカニズムであり、この分離を実現するためのツールは、主に OS ファミリに依存している。例えば、プロセスを互いに分離するために使用される方法は、Linux と Windows で異なる。しかし、基盤となる実装は異なる場合があるが、一般的に使用されているコンテナランタイムは、これらの違いをユーザから大幅に取り除く

[4] cgroups は独立して管理できるプロセスの集まりで、メモリ、プロセッサの使用状況、ディスク I/O などのサブシステムを測定するソフトウェアベースの機能をカーネルに提供する。管理者はこれらのサブシステムを手動またはプログラムで制御できる。

共通のインタフェース形式を提供している。

コンテナは強力な分離を提供するが、VM ほど明確で具体的なセキュリティ境界を提供しない。コンテナは同じカーネルを共有し、ホスト上で様々な機能と権限で実行できるため、コンテナ間のセグメンテーションの度合いは、ハイパーバイザによって VM に提供されるものよりもはるかに小さい。この結果、ぞんざいに設定された環境でも、コンテナは、同じホスト上の複数の VM よりもはるかに簡単かつ直接的に、お互いやホストと相互作用する機能を持つことができる。

コンテナは、ハードウェア仮想化を凌駕する仮想化の次のフェーズとして考えられることもあるが、ほとんどの組織にとっての現実は、進化というより革命と言うべきものである。コンテナとハードウェア仮想化は、単に共存できるだけでなく、非常に頻繁に共存し、実際にお互いの機能を強化し合う。VM には、強力な分離、OS の自動化、ソリューションの幅広くて深いエコシステムなど、多くの利点がある。組織はコンテナと VM のどちらかを選択する必要はない。代わりに、組織は、コンテナを使ってアプリをパッケージ化し、各 VM をより効率的に活用しながら、VM をデプロイ、パーティション、ハードウェアの管理に使い続けることができる。

2.3 コンテナ技術アーキテクチャ

図 3 は、コンテナ技術アーキテクチャの 5 つの階層を示している。

1. 開発者用システム（イメージを生成し、テストや認定へ送る）
2. テスト/認定システム（イメージの内容の妥当性確認・検証、イメージへの署名、レジストリへのイメージの送付）
3. レジストリ（イメージを保存し、要求に応じてオーケストレータに配布する）
4. オーケストレータ（イメージをコンテナに変換し、コンテナをホストにデプロイする）
5. ホスト（オーケストレータの指示通りに、コンテナを実行したり停止したりする）

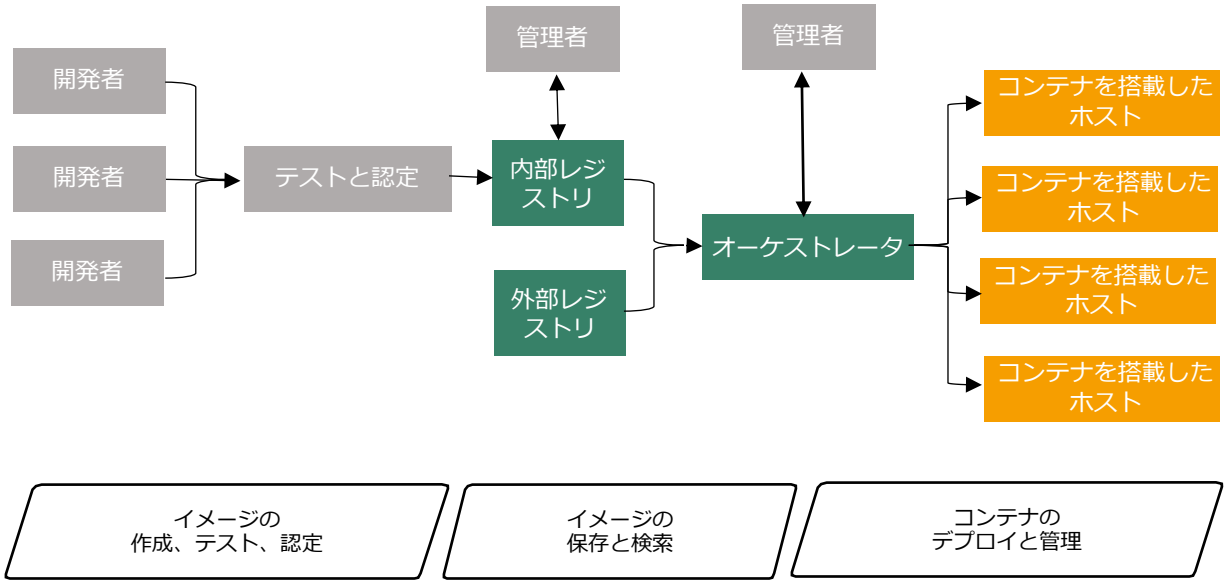

図 3：コンテナ技術アーキテクチャの階層、コンポーネント、およびライフサイクルフェーズ

プロセス全体に関与する管理者用システムの役割は多数あるが、この図は、内部レジストリとオーケストレータの為の管理者用システムのみを示している。

灰色のシステム（開発者用システム、テスト/認定システム、管理者用システム）は、コンテナ技術アーキテクチャの範囲外であるが、コンテナ技術アーキテクチャとの重要な関連がある。コンテナを使用しているほとんどの組織では、開発環境やテスト環境もコンテナを活用しており、この一貫性はコンテナを使用する主な利点の 1 つである。これらの環境のシステムの安全性を確保するための推奨事項は、本番環境のシステムとほぼ同じであるため、本文書ではこれらの環境のシステムには重点を置いていない。緑色のシステム（内部レジストリ、外部レジストリ、オーケストレータ）は、コンテナ技術アーキテクチャの中核となるコンポーネントである。最後に、オレンジ色のシステム（コンテナを搭載したホスト）は、コンテナが使用される場所である。

コンテナ技術アーキテクチャを理解するもう一つの方法は、図 3 の下部に描かれているコンテナライフサイクルのフェーズを考えることである。この 3 つのフェーズについては、以下で詳しく説明する。

組織は通常、一度に多くの異なるアプリを構築してデプロイするため、これらのライフサイクルフェーズは同じ組織内で同時に発生することが多く、成熟度の段階的な進行とは見なせない。むしろ、継続的に稼働しているエンジンのサイクルと考える。この比喩では、各アプリはエンジン内のシリンダーであり、異なるアプリが同時にこのライフサイクルの異なるフェーズにある場合がある。

2.3.1 イメージの作成、テスト、認定

コンテナのライフサイクルの最初のフェーズでは、アプリのコンポーネントが構築され、イメージ（または複数のイメージ）に配置される。**イメージ**とは、コンテナを実行するために必要なすべてのファイルを含むパッケージである。例えば、Apache を実行するイメージには httpd バイナリと関連するライブラリおよび設定ファイルが含まれるであろう。イメージには、アプリ自体に必要な実行可能ファイルとライブラリのみが含まれることが望ましい。他のすべての OS 機能は、基盤となるホスト OS 内の OS カーネルによって提供される。多くの場合、イメージはレイヤ化やコピーオンライト（共有マスタイメージは読み取り専用で、変更は個別のファイルに記録される）などの手法を使用して、ディスク上のサイズを最小限に抑え、運用効率を向上させる。

イメージはレイヤで構成されており、他のすべてのコンポーネントが追加される下層のレイヤは**ベースレイヤ**と呼ばれる。ベースレイヤは通常、Ubuntu や Windows Nano Server などの一般的な OS の最小限のディストリビューションで、OS カーネルが省略されている。ユーザは、これらのベースレイヤの一つから始めて、全部のイメージの構築を開始する。次にアプリケーションフレームワークと独自のカスタムコードを追加して、完全にデプロイ可能な独自アプリのイメージを開発する。コンテナランタイムは、特定のホスト OS バージョンが異なる場合でも、同じ OS ファミリ内のイメージの使用をサポートする。例えば、Docker を実行している Red Hat ホストは、Alpine や Ubuntu などの Linux ベースレイヤ上で作成されたイメージを実行することができる。ただし、Windows ベースレイヤで作成されたイメージは実行できない。

イメージ作成プロセスは、テストへの引き渡し用にアプリのパッケージ化を担当する開発者によって管理される。 イメージの作成には通常、いわゆる「継続的インテグレーション」と呼ばれるプロセスにより支援する、Jenkins [8]や TeamCity [9]などのビルド管理と自動化のためのツールを使用する。これらのツールは、アプリのさまざまなライブラリ、バイナリ、およびその他のコンポーネントを取得してテストを実行し、アプリのイメージを構築する方法を記述した開発者が作成したマニフェストに基づいて、それらからイメージをアセンブルする。

ほとんどのコンテナ技術には、アプリのコンポーネントと要件を記述する宣言的な方法がある。たとえば、Web サーバのイメージには、Web サーバの実行ファイルだけでなく、Web サーバが待ち受けするポートや使用する設定パラメータなどの実行方法を記述するための機械で解析可能なデータも含まれる。

イメージの作成後、組織は通常、テストと認定を行う。例えば、テスト自動化ツールと担当者は、最終形態のアプリケーションの機能の妥当性を確認するために構築されたイメージを使用し、セキュリティチームはこれらの同じイメージに対して認定を行う。一つのアプリに対する成果物をまったく同じように構築、テスト、認定するという一貫性は、コンテナの運用上およびセキュリティ上の重要な利点の 1 つである。

2.3.2 イメージの保存と検索

通常、イメージは中心となる場所に保存され、ホスト間での制御、共有、検索、再利用を容易にする。レジストリは、開発者が作成したイメージを簡単に保存したり、イメージの識別やバージョン管理のためにタグ付けやカタログを作成して発見や再利用を助けたり、他のユーザが作成したイメージを検索してダウンロードしたりすることができるサービスである。レジストリは、自己ホストされている場合もあれば、サービスとして使用されている場合もある。レジストリの例としては、Amazon EC2 Container Registry [10]、Docker Hub [11]、Docker Trusted Registry [12]、Quay Container Registry [13]などがある。

レジストリは、共通的なイメージ関連のタスクを自動化するための API を提供する。例えば、組織は、イメージの作成フェーズで、テストに合格すると自動的にイメージをレジストリに配信するトリガーを持っても良い。レジストリには、追加された新しいイメージのデプロイを自動化するトリガーを用意しても良い。この自動化により、より一貫した結果が得られるプロジェクトをより迅速に繰り返すことができる。

レジストリに保存されたイメージは、簡単に取り出し、DevOps のペルソナによって、コンテナを実行するすべての環境で実行することができる。これはコンテナの移植性の利点の 1 つの例である。すなわち、イメージの作成はパブリッククラウドプロバイダで行われ、プライベートクラウドでホストされているレジストリにイメージをプッシュし、第 3 の場所でアプリを実行するためのイメージを配布するために使用される。

2.3.3 コンテナのデプロイと管理

オーケストレータと呼ばれるツールは、DevOps のペルソナや自動化を代行して、レジストリからイメージを引き出し、それらのイメージをコンテナにデプロイし、実行中のコンテナを管理することを可能にする。このデプロイのプロセスにより、実際に使用可能なバージョンのアプリが実行され、リクエストに応答する準備が整う。イメージがコンテナにデプロイされると、イメージ自体は変更されないが、その代わりにイメージのコピーがコンテナ内に配置され、休止中のアプリコードのセットから実行中のアプリのインスタンスに移行する。オーケストレータの例としては、Kubernetes [14]、Mesos [15]、Docker Swarm [16]などがある。

小さくてシンプルなコンテナの実装では、本格的なオーケストレータが省略される可能性があることに留意すべきである。オーケストレータは、他の状況では回避されたり、不要になったりすることもある。例えば、ホストは、イメージをレジストリからコンテナランタイムに引き出すために、レジストリに直接アクセスすることができる。本文書では、説明を簡略化するために、オーケストレータの使用を前提とする。

オーケストレータによって提供される抽象化により、DevOps のペルソナは、与えられたイメージを実行するために必要なコンテナの数と、それぞれに割り当てる必要があるメモリ、処理、ディスクなどのリソースを簡単に指定することができる。オーケストレータは、クラスタ内の各ホストの状態（各ホストで使用可能なリソースなど）を把握し、どのコンテナがどのホストで実行されるかを決定する。次に、オーケストレータは必要なイメージをレジストリから取り出し、指定されたリソースでコンテナとして実行する。

また、オーケストレーションツールは、コンテナリソースの消費、ジョブの実行、ホスト全体のマシンの正常性のモニタリングも行う。設定によっては、オーケストレータは、最初に実行していたホストに障害が発生した場合、新しいホスト上でコンテナを自動的に再起動することができる。多くのオーケストレータは、ホスト間を跨るコンテナネットワークの構築とサービスディスカバリを可能にする。ほとんどのオーケストレータは**オーバーレイネットワーク**として知られる Software-Defined Networking (SDN) コンポーネントも含んでおり、同じ物理ネットワークを共有するアプリケーション間の通信を分離するために使用することができる。

コンテナ内のアプリを更新する必要がある場合、既存のコンテナは変更されずに破棄され、更新されたイメージから新しいコンテナが作成される。これは、コンテナでの重要な運用上の違いである。最初にデプロイした時のベースラインのソフトウェアは、時間の経過とともに変化してはならず、更新はイメージ全体を一度に置き換えることによって行われる。このアプローチは、各フェーズでまったく同じ構成でまったく同じソフトウェアを構築、テスト、検証、およびデプロイすることができるため、セキュリティ上の大きなメリットとなる可能性がある。アプリの更新が

行われると、組織は、通常はオーケストレータを活用することで、最新のバージョンを確実に使用することができる。オーケストレータは通常、最新のバージョンのイメージをレジストリから引き出すよう設定されているため、アプリは常に最新の状態に保たれる。この「継続的デリバリー」の自動化により、開発者は単にアプリの新しいバージョンのイメージを構築し、イメージをテストしてレジストリに配信し、自動化ツールによってターゲット環境にデプロイすることができる。

つまり、パッチや構成設定を含むすべての脆弱性管理は、通常、新しいイメージのバージョンを構築する際に開発者が処理することになる。コンテナの場合、運用チームではなく、開発者が、アプリとイメージのセキュリティに大きな責任を負う。このような責任の変化により、以前よりもはるかに大きな調整と人員間の協力が必要になることがよくある。コンテナを採用する組織は、各利害関係者グループに対して明確なプロセスとチームの責任が確立されていることを確かにすることが望ましい。

コンテナ管理には、セキュリティ管理とモニタリングが含まれる。しかし、非コンテナ環境用に設計されたセキュリティ管理は、コンテナでの使用にはあまり適していないことが多い。たとえば、IP アドレスを考慮したセキュリティ管理を考えてみる。これは、何ヶ月も、または何年も静的な IP アドレスが同じままの VM や bare metal サーバに適している。逆に、コンテナは通常、オーケストレータによって IP アドレスが割り当てられる、コンテナは VM よりもはるかに頻繁に作成・破棄されるため、これらの IP アドレスも時間の経過とともに頻繁に変化する。このため、IP アドレスに基づいてトラフィックをフィルタリングするファイアウォールのルールセットなど、静的な IP アドレスに依存するセキュリティ技術を使用してコンテナを保護することは困難または不可能となる。さらに、コンテナネットワークには、同じノード上のコンテナ間の通信、異なるノード間の通信、さらにはクラウド間の通信も含めることが出来る。

2.4 コンテナの用途

他の技術と同様に、コンテナは万能ではない。コンテナは多くのシナリオで有用なツールであるが、必ずしもすべてのシナリオにおいて最適な選択とは限らない。例えば、レガシーな既製のソフトウェアを大量に保有している組織では、ベンダがサポートしていない可能性があるため、そのソフトウェアのほとんどを実行するためにコンテナを活用することはできない。しかし、ほとんどの組織にとって、コンテナには複数の有用な用途がある。その例としては、以下のようなものがある。

- アプリが頻繁に更新され、デプロイされるアジャイル開発。コンテナの移動可能で宣言的な性質は、これらの頻繁な更新をより効率的とし、テストもより容易とする。これによって、組織はイノベーションを加速し、ソフトウェアをより迅速に提供できる。また、アプリのコードの脆弱性を修正し、更新されたソフトウェアをより迅速にテストしてデプロイできる。
- 環境の一貫性と区分化により、開発者は、アプリの構築、テスト、実行のための同一でありながら個別の環境を持つことができる。コンテナを使用すると、開発者は、開発用ラップトップシステム上で本番アプリの完全なコピーをローカルで実行することができ、テスト環境の調整や共有の必要性を制限し、古臭いテスト環境の煩わしさを排除することができる。
- 「スケールアウト」シナリオでは、特定の時点での負荷に応じて、アプリが迅速に多くの新しいインスタンスをデプロイまたは廃止する必要がある場合がある。コンテナのイミュータブルによって、各インスタンスが他のすべてのインスタンスと全く同じであることがわかっているため、インスタンスを確実にスケールア

ウトすることが容易になる。さらに、コンテナは一般的にステートレスであるため、不要になったら簡単に廃止することができる。

- 開発者が最初からマイクロサービスアーキテクチャに合わせて構築できるクラウドネイティブアプリでは、アプリの効率的なイテレーションと簡素化されたデプロイを実現する。

コンテナにはさらに利点がある。例えば、コンテナイメージのイミュータブルにより、ビルドパイプラインの効率を高めることができる。コンテナは、本番のコードをインストールする時間と場所を変えることができる。非コンテナシステムでは、アプリのインストールは本番環境（すなわち、サーバランタイム）で行われ、通常、サーバ上でアプリコード（プログラミング言語ランタイム、依存するサードパーティライブラリ、init スクリプト、OS ツールなど）のインストールを管理するための手作りのスクリプトを実行する。つまり、本番前のビルドパイプライン（および開発者のワークステーション）で実行されるテストは、実際の本番の生成物をテストしているのではなく、ビルドシステムに含まれている最も有力な近似値でテストしていることになる。特に、本番システムとビルドシステムを管理するチームが異なる場合は、この本番の近似値が時間の経過とともに本番からずれていく傾向がある。このシナリオは、「私のマシンでは動作する (it works on my machine) 」問題の具体例である。

コンテナ技術では、ビルドシステムは、作成したイメージ内に（つまり、コンパイル時に）アプリをインストールする。イメージは、アプリのすべてのユーザ空間要件（プログラミング言語ランタイム、依存するサードパーティライブラリ、init スクリプト、OS ツールなど）の不変のスナップショットである。本番環境では、ビルドシステムによって構築されたコンテナイメージをダウンロードして実行するだけである。これにより、開発者、ビルドシステム、および本番環境はすべて同じ不変の生成物を実行するため、「私のマシン上では動作する (works on my machine) 」問題が解決される。

最近のコンテナ技術は再利用を重視していることが多く、ある開発者が作成したコンテナイメージを、同じ組織内または他の組織で、他の開発者が簡単に共有して再利用できるようにしている。レジストリサービスは、一元化されたイメージ共有と検出サービスを提供し、開発者が他の人が作成したソフトウェアを簡単に見つけて再利用できるようにする。このような使い勝手の良さから、多くの一般的なソフトウェアベンダやプロジェクトでは、顧客がソフトウェアを簡単に見つけて素早く実行できるようにする方法として、コンテナを使用するようになっている。例えば、MongoDB のようなアプリをホスト OS に直接インストールするのではなく、ユーザは MongoDB のコンテナイメージを実行するだけで済む。さらに、コンテナランタイムは、コンテナをコンテナ同士やホスト OS から分離するので、これらのアプリをより安全かつ確実に実行でき、また、ユーザは基盤となるホスト OS に影響を与えることを心配する必要がない。

3 コンテナ技術のコアコンポーネントの主なリスク

本節では、コンテナ技術のコアコンポーネントであるイメージ、レジストリ、オーケストレータ、コンテナ、ホスト OS の主要なリスクを特定し、分析する。この分析は、コアコンポーネントのみを対象としているため、コンテナ技術、ホスト OS プラットフォーム、ロケーション（パブリッククラウド、プライベートクラウドなど）に関係なく、ほとんどのコンテナのデプロイに適用できる。リスクには、下記の 2 種類があると考えられる。

1. **イメージやコンテナへのセキュリティ侵害。**このリスクは、NIST SP 800-154 [17]に記述されているデータ中心のシステム脅威モデリングアプローチを使用して評価された。最初に保護すべき「データ」は、アプリファイルやデータファイルなどを保持している可能性のあるイメージやコンテナである。次に保護すべきデータは、メモリ、ストレージ、ネットワークインタフェースなどの共有ホストリソース内のコンテナデータである。
2. **他のコンテナ、ホスト OS、他のホスト等を攻撃するためのコンテナの悪用。**

コアコンポーネントに関わるその他のすべてのリスク、および、開発者用システム、テストと認定用システム、管理者用システム、ホストのハードウェアと仮想マシン管理を含む、非コアコンテナ技術コンポーネントに関わるリスクは、本文書の範囲外である。付録 A には、非コアコンテナ技術コンポーネントをセキュアにするための一般的な参考文献へのリンクを記載している。

3.1 イメージのリスク

3.1.1 イメージの脆弱性

イメージは実質的に静的なアーカイブファイルで、特定のアプリを実行するために使用されるすべてのコンポーネントが含まれているため、イメージ内のコンポーネントには重要なセキュリティアップデートが漏れているか、または古くなっている可能性がある。完全に最新のコンポーネントで作成されたイメージは、作成後数日から数週間は既知の脆弱性がないかもしれないが、ある時に 1 つまたは複数のイメージのコンポーネントに脆弱性が発見されると、イメージはもはや最新のものではなくなる。

デプロイされたソフトウェアが実行されるホスト上の「現場」で更新される従来の運用パターンとは異なり、コンテナの場合は、イメージ自体の更新を上流で行い、それを再デプロイする必要がある。このように、コンテナ化された環境での一般的なリスクは、コンテナを生成するために使用されたイメージのバージョンに脆弱性があるため、デプロイされたコンテナが脆弱性を持つことである。

3.1.2 イメージの設定の不備

ソフトウェアの不備に加えて、イメージには設定の不備がある場合がある。例えば、イメージが特定のユーザアカウントとして実行する「run as」に設定されておらず、必要以上に大きな権限で実行されてしまう場合がある。別の例として、イメージに SSH デーモンが含まれている場合がある。これによって、コンテナが不要なネットワークリスクにさらされる。従来のサーバや VM において、設定に不備があれば、完全に最新のシステムであっても攻撃にさら

される可能性があるように、設定に不備があるイメージは、含まれているすべてのコンポーネントが最新の状態であっても、リスクが高まる可能性がある。

3.1.3 埋め込まれたマルウェア

イメージはパッケージ化されたファイルの集まりにすぎないため、悪意のあるファイルが意図的に、または不注意でイメージ内に含まれる可能性がある。このようなマルウェアは、イメージ内の他のコンポーネントと同じ機能を持っているため、環境内の他のコンテナまたはホストを攻撃するために使用される可能性がある。埋め込まれたマルウェアのソースとして考えられるのは、完全な出所が不明なサードパーティによって提供されるベースレイヤやイメージの使用である。

3.1.4 埋め込まれた平文の秘密情報

多くのアプリでは、コンポーネント間の安全な通信を可能にするために秘密情報が必要である。例えば、ウェブアプリはバックエンドのデータベースに接続するために、ユーザ名とパスワードが必要となる場合がある。その他の埋め込まれた秘密情報の例としては、接続文字列、SSH 秘密鍵、X.509 秘密鍵などがある。アプリをイメージにパッケージ化すると、これらの秘密情報をイメージファイルシステムに直接埋め込むことができる。しかし、この方法では、イメージにアクセスできる人なら誰でも簡単に解析してこれらの秘密情報を知ることができるため、セキュリティリスクが生じる。

3.1.5 信頼できないイメージの使用

あらゆる環境において最も一般的なハイリスクのシナリオの 1 つは、信頼できないソフトウェアの実行である。コンテナの移動可能性と再利用のしやすさによって、チームは外部ソースからのイメージを実行したいという誘惑に駆られるが、それらは十分に検証されていないか、信頼できない可能性がある。例えば、ウェブアプリの問題をトラブルシューティングする場合、ユーザはサードパーティが提供するイメージの中に、そのアプリの別のバージョンが利用可能であることを見つけることがある。この外部から提供されたイメージを使用すると、マルウェアの取り込み、データの漏えい、脆弱性のあるコンポーネントの組み込みなど、外部ソフトウェアが従来から持っていたリスクと同じ種類のリスクが発生する。

3.2 レジストリのリスク

3.2.1 レジストリへのセキュアでない接続

イメージには組織独自のソフトウェアや埋め込まれた秘密情報など、機微なコンポーネントが含まれていることがよくある。レジストリへの接続が、セキュアでないチャネルで行われた場合、イメージの内容は、平文で送信される他のデータと同様の機密性に対するリスクにさらされる。また、レジストリを対象としているネットワークトラフィックを傍受し、そのトラフィック内の開発者や管理者の認証情報を盗み出し、詐欺的なイメージや古いイメージをオーケストレータに提供したりする中間者攻撃のリスクも高まる。

3.2.2 レジストリ内の古いイメージ

レジストリは通常、組織がデプロイするすべてのイメージのソースとなる場所であるため、レジストリに保存された一連のイメージに、時がたつにつれて多くの脆弱性のある古いバージョンが含まれる可能性がある。これらの脆弱性のあるイメージは、レジストリに保存されているだけでは直接の脅威とはならないが、既知の脆弱性のあるバージョンが誤ってデプロイされる可能性が高くなる。

3.2.3 認証・認可の不十分な制限

レジストリには、機微なアプリや独自のアプリを実行するイメージや、機微なデータにアクセスするために使用されるイメージが含まれている可能性があるため、認証・認可の要件が不十分であると、知的財産の損失につながり、アプリに関する重要な技術的詳細を攻撃者に晒してしまう可能性がある。さらに重要なのは、レジストリは通常、正当で有効なソフトウェアのソースとして信頼されているため、レジストリが侵害されると、下流のコンテナやホストの侵害につながる可能性がある。

3.3 オーケストレータのリスク

3.3.1 制限のない管理者アクセス

歴史的に、多くのオーケストレータは、それらと対話するすべてのユーザが管理者であることを想定し、これらの管理者が環境全体を制御することを前提に設計されていた。しかし多くの場合、1 つのオーケストレータが様々なアプリを実行し、それぞれが異なるチームによって管理され、様々な機微性のレベルで実行される。ユーザ及びグループに提供されるアクセスが特定の要求に限定されていない場合、悪意のあるユーザまたは不注意なユーザが、オーケストレータが管理する他のコンテナの操作に影響を及ぼしたり、操作を妨害したりする可能性がある。

3.3.2 不正アクセス

多くの場合、オーケストレータには、組織内ですでに使用されている一般的なディレクトリとは別の、独自の認証ディレクトリサービスが含まれている。これらのシステムは厳密に管理されていないため、オーケストレータ内のより弱いアカウント管理の慣行と、「孤立した」アカウントの原因となる可能性がある。これらのアカウントの多くは、オーケストレータ内で高い特権を与えられているので、これらのアカウントの侵害が、システム全体の侵害につながる可能性がある。

コンテナは通常、オーケストレーションツールによって管理されるホスト固有ではないデータストレージボリュームを使用する。なぜなら、コンテナはクラスタ内の任意のノードで実行できるため、コンテナ内のアプリが要求するデータは、コンテナが実行されているホストに関係なく、コンテナで使用できる必要があるからである。同時に、多くの組織では、不正アクセスを防ぐために、保存時に暗号化する必要があるデータを管理している。

3.3.3 コンテナ間ネットワークトラフィックの不十分な分離

ほとんどのコンテナ化された環境では、個々のノード間のトラフィックは、仮想オーバーレイネットワークを介してルーティングされる。このオーバーレイネットワークは通常、オーケストレータによって管理され、既存のネットワークセキュリティおよび管理ツールからは見えない。例えば、従来のネットワークフィルタでは、ウェブサーバから別のホスト上のデータベースコンテナに送信されるデータベースクエリを確認する代わりに、2 つのホスト間を流れる暗号化されたパケットのみを確認し、実際のコンテナのエンドポイントや送信されるトラフィックを確認することはできない。暗号化されたオーバーレイネットワークは、多くの運用上およびセキュリティ上の利点を提供するが、組織が自社のネットワーク内のトラフィックを効果的にモニタリングできないというセキュリティ上の「盲点」のシナリオを生み出す可能性もある。

さらに重要なのは、同じ仮想ネットワークを共有する、異なるアプリからのトラフィックのリスクである。社外に公開しているウェブサイトや社内の財務管理アプリなど、機微性レベルの異なるアプリが同じ仮想ネットワークを共有している場合、機微性の高い社内アプリがネットワーク攻撃のリスクにさらされる可能性がある。例えば、公開しているウェブサイトが侵害された場合、攻撃者は共有されたネットワークを使用して、財務アプリを攻撃できる可能性がある。

3.3.4 ワークロードの機微性レベルの混合

オーケストレータは通常、主にワークロードの規模と密度を高めることに重点を置いている。つまり、デフォルトでは、機微性レベルの異なるワークロードを同じホストに配置することができる。例えば、デフォルトの設定では、オーケストレータは、公開している ウェブサーバを実行しているコンテナと、機微性の高い財務データを処理しているコンテナを同じホストに配置することがあるが、これは単に、そのホストがデプロイ時に最も利用可能なリソースを持っているからである。ウェブサーバに致命的な脆弱性がある場合、これにより、機微性の高い財務データを処理するコンテナが侵害の危険にさらされるリスクが大幅に高まる可能性がある。

3.3.5 オーケストレータノードの信頼

環境内のノード間の信頼の維持には特別な注意が必要である。オーケストレータは最も中心となるノードである。脆弱なオーケストレータの設定は、オーケストレータと他のすべてのコンテナ技術コンポーネントをリスクにさらす可能性がある。考えられる結果の例としては、以下のようなものがある。

- 不正なホストがクラスタに参加し、コンテナを実行
- 一つのクラスタのホストの侵害が、クラスタ全体に侵害をもたらす。例えば、同じ認証用の鍵ペアが、すべてのノードで共有されている場合など。
- オーケストレータと DevOps 担当者、管理者、およびホスト間の通信が暗号化されず、認証もされない。

3.4 コンテナのリスク

3.4.1 ランタイムソフトウェア内の脆弱性

比較的まれではあるが、悪意のあるソフトウェアが他のコンテナやホスト OS 自体のリソースを攻撃する「コンテナエスケープ」シナリオを、ランタイムソフトウェア内の脆弱性が可能にする場合は非常に危険である。また、攻撃者が脆弱性を悪用してランタイムソフトウェア自体を侵害し、そのソフトウェアを改変して他のコンテナにアクセスしたり、コンテナ間の通信をモニタリングしたりすることができる可能性もある。

3.4.2 コンテナからの無制限のネットワークアクセス

ほとんどのコンテナランタイムのデフォルトでは、個々のコンテナはネットワークを介してお互いにアクセスしたり、ホスト OS にアクセスしたりすることができる。コンテナが侵害され、悪意を持って動作している場合、このネットワークトラフィックの許可が、環境内の他のリソースを危険にさらす可能性がある。例えば、侵害されたコンテナは、攻撃者が悪用するための他の弱点を見つけるために、接続されているネットワークをスキャンすることに使用される可能性がある。このリスクは、3.3.3 節で説明したように、分離が不十分な仮想ネットワークのリスクと関連しているが、アプリの「漏話（cross talk）」シナリオではなく、コンテナから任意の外部への流れに焦点を当てている点が異なる。

コンテナ化された環境では、接続の多くがコンテナ間で仮想化されているため、外向けのネットワークアクセス (egress network access) の管理は、より複雑になる。したがって、あるコンテナから別のコンテナへのトラフィックは、最終的なソース、宛先、またはペイロードを直接示すことなく、単にネットワーク上でカプセル化されたパケットとして見える場合がある。コンテナに対応していないツールや運用プロセスは、このトラフィックを検査したり、脅威かどうかを判断したりすることができない。

3.4.3 セキュアでないコンテナランタイムの設定

コンテナランタイムは通常、多くの設定可能なオプションを管理者に提供している。それらを不適切に設定すると、システムの相対的なセキュリティが低下する可能性がある。例えば、Linux コンテナホストでは、許可される一連のシステムコールは、多くの場合、デフォルトではコンテナの安全な操作に必要なものだけに限定されている。このリストを拡大すると、侵害されたコンテナによるコンテナとホスト OS のリスクを増大させる可能性がある。同様に、コンテナが特権モードで実行されている場合、コンテナはホスト上のすべてのデバイスにアクセスできるため、実質的にホスト OS の一部として動作し、その上で実行されている他のすべてのコンテナに影響を与えることができる。

セキュアでないランタイムの設定のもう一つの例は、コンテナがホスト上の機微性の高いディレクトリのマウントを許可することである。コンテナがホスト OS のファイルシステムに変更を加えることはほとんどなく、ホスト OS の基本的な機能を制御する場所（例えば、Linux コンテナの場合は、/boot や/etc、Windows コンテナの場合は C:\Windows）に変更を加えることもほとんどない。侵害されたコンテナがこれらのパスを変更することを許可された場合、特権を昇格させ、ホスト自身やホスト上で実行されている他のコンテナを攻撃するために使用される可能性がある。

3.4.4 アプリの脆弱性

組織が、本文書で推奨されている予防策を講じている場合でも、コンテナが実行するアプリの不具合により、コンテナが侵害される可能性がある。これはコンテナ自体の問題ではなく、コンテナ環境内の典型的なソフトウェアの不具合の現れに過ぎない。例えば、コンテナ化された Web アプリは、クロスサイトスクリプティングの脆弱性にさらされる可能性があり、データベースフロントエンドのコンテナは SQL インジェクションの対象となる可能性がある。コンテナが侵害されると、機微情報への不正アクセスを許可したり、他のコンテナやホスト OS に対する攻撃を可能にしたりするなど、様々な方法で悪用される可能性がある。

3.4.5 未承認コンテナ

未承認（rogue）コンテナとは、環境内で計画されていない、または許可されていないコンテナのことである。これは、特にアプリ開発者がコードをテストする手段としてコンテナを起動する可能性がある開発環境では、よくあることである。このようなコンテナは、脆弱性スキャンや適切な設定がきちんと行われていないと、悪用されやすくなる可能性がある。そのため、未承認コンテナは、特に開発チームやセキュリティ管理者に気付かれずに環境に留まっている場合、組織にさらなるリスクをもたらす。

3.5 ホスト OS のリスク

3.5.1 大きなアタックサーフェス

すべてのホスト OS にはアタックサーフェスがある。これは攻撃者がホスト OS の脆弱性にアクセスして悪用しようとするすべての方法の集合体である。例えば、ネットワークからアクセス可能なサービスは、攻撃者に潜在的なエントリーポイントを提供し、アタックサーフェスを追加する。アタックサーフェスが大きければ大きいほど、攻撃者が脆弱性を見つけてアクセスできる可能性が高くなり、ホスト OS とそのうえで実行されているコンテナの侵害につながる。

3. 5.2 共有カーネル

コンテナ専用 OS は、汎用 OS に比べてアタックサーフェスがはるかに小さい。例えば、汎用 OS がデータベースアプリやウェブサーバアプリを直接実行できるようにするライブラリやパッケージマネージャはコンテナ専用 OS には含まれていない。ただしコンテナは、ソフトウェアレベルでのリソースの強力な分離を提供するが、共有カーネルを使用することで、コンテナ専用 OS であっても、ハイパーバイザで見られるよりも大きなオブジェクト間アタックサーフェスが必ず発生する。言い換えれば、コンテナランタイムが提供する分離のレベルは、ハイパーバイザが提供するものほど高くはない。

3.5.3 ホスト OS コンポーネントの脆弱性

コンテナ専用 OS であっても、すべてのホスト OS は、リモート接続の認証に使用される暗号ライブラリや、一般的なプロセスの呼び出しと管理に使用されるカーネルプリミティブなど、基本的なシステムコンポーネントを提供している。他のソフトウェアと同様に、これらのコンポーネントにも脆弱性がある可能性があり、コンテナ技術アーキテクチャの下位に存在するため、これらのホストで実行されるすべてのコンテナとアプリに影響を与える可能性がある。

3.5.4 不適切なユーザアクセス権

対話型のユーザログオンはまれであるため、コンテナ専用 OS は通常、マルチユーザシナリオをサポートするようには最適化されていない。ユーザがコンテナを管理する目的では、オーケストレーションレイヤを経由してログオンするよりも、直接ホストにログオンする際に、組織はよりリスクにさらされる。直接管理することで、システムとその上にあるすべてのコンテナに対する広範な変更が可能になり、特定のアプリのコンテナを管理したいだけのユーザが、他の多くのコンテナに影響を与えることが潜在的に可能となってしまう。

3.5.5 ホスト OS ファイルシステムの改ざん

セキュアでないコンテナの構成は、ホストボリュームをファイル改ざんのリスクにさらす可能性がある。例えば、あるコンテナがホスト OS 上に機微性の高いディレクトリをマウントすることを許可されている場合、そのコンテナはそのディレクトリ内のファイルを変更することができる。これらの変更は、ホストおよびその上で実行されている他のすべてのコンテナの安定性とセキュリティに影響を与える可能性がある。

4 主なリスクへの対策

この節では、3 節で識別した主なリスクに対する対策を推奨する。

4.1 イメージの対策

4.1.1 イメージの脆弱性

コンテナ技術に特化した脆弱性管理ツールおよびプロセスが必要となる。従来の脆弱性管理ツールは、ホストの耐久性やアプリの更新メカニズムや頻度について多くの仮定をしているが、それらはコンテナ化されたモデルとは根本的にずれている。これらのツールは、コンテナ内の脆弱性を検知できないことが多く、誤った安心感をもたらす。

組織は、より実用的で信頼性の高い結果を提供するために、パイプラインベースのビルドアプローチとコンテナおよびイメージのイミュータブルを設計に取り入れたツールを使用することが望ましい。効果的なツールとプロセスの主要な側面としては、以下のようなものがある。

1. ビルドプロセスの最初から、組織が使用しているレジストリ、ランタイムに至るまでの、イメージのライフサイクル全体の統合。
2. イメージのベースレイヤだけでなく、組織が使用しているアプリケーションフレームワークやカスタムソフトウェアなど、イメージのすべてのレイヤにおける脆弱性の可視性。可視性は、組織全体で一元化され、組織のビジネスプロセスに合わせた柔軟なレポーティング及びモニタリングビューを提供することが望ましい。
3. ポリシー主導の強制。組織はビルドとデプロイのプロセスの各段階において「品質ゲート」を作成し、組織の脆弱性ポリシーと構成ポリシーを満たすイメージのみが進行を許可されることを確実にすることが望ましい。例えば、組織は、ビルドプロセスにおいて、選択されたしきい値を超える共通脆弱性評価システム（CVSS）[18]の評価を持つ脆弱性を含むイメージの進行を防止するためのルールを設定できることが望ましい。

4.1.2 イメージ設定の不具合

組織は、セキュアな設定のベストプラクティスへの準拠を検証および実施するためのツールとプロセスを採用することが望ましい。例えば、イメージは非特権ユーザとして実行されるよう設定することが望ましい。採用することが望ましいツールとプロセスには、以下のようなものがある。

1. ベンダの推奨事項やサードパーティのベストプラクティスを含む、イメージ設定の検証。
2. 組織レベルでの弱点やリスクを識別するための、継続的で、 絶えず更新され、一元管理された、イメージのコンプライアンス状態のレポーティングとモニタリング。
3. コンプライアンスに準拠していないイメージの実行を任意で防止することによる、コンプライアンス要件の強制。
4. 信頼できるソースからのベースレイヤのみを使用し、ベースレイヤを頻繁に更新し、Alpine Linux や Windows Nano Server のような最小限の技術からベースレイヤを選択して、アタックサーフェスの領域を減らす。

イメージ設定の最終的な推奨事項としては、SSH やその他のリモートシェルをホストに提供するように設計されたリモート管理ツールは、コンテナ内では決して有効にしないことが望ましい。コンテナは、その使用による最大のセキュリティ上のメリットを享受するために、変更できない方法で実行されることが望ましい。リモート管理ツールを使ってコンテナへのリモートアクセスを有効にすることは、この原則に違反し、ネットワークベースの攻撃のリスクを高めるような変更であることを意味する。その代わり、コンテナのすべてのリモート管理はコンテナランタイム API を介して行うことが望ましい。コンテナランタイム API はオーケストレーションツールを介してアクセスするか、コンテナが実行されているホストへのリモートシェルセッションを作成することでアクセスできる。

4.1.3 埋め込まれたマルウェア

組織は、埋め込まれたマルウェアがないか、すべてのイメージを継続的にモニタリングすることが望ましい。モニタリングプロセスには、マルウェアのシグネチャセットと、主に実際の「出回っている」攻撃に基づいた行動検知ヒューリスティックが使用されていることが望ましい。

4.1.4 埋め込まれた平文の秘密情報

秘密情報はイメージの外に保存し、実行時に動的に提供されるのが望ましい。Docker Swarm や Kubernetes などのほとんどのオーケストレータには、秘密情報の管理が元から含まれている。これらのオーケストレータは、秘密情報のセキュアな保存とコンテナへの「ジャストインタイム」インジェクションを提供するだけでなく、秘密情報の管理を、ビルドとデプロイのプロセスに統合することをはるかに容易にする。例えば、組織はこれらのツールを使用して、ウェブアプリケーションコンテナにデータベース接続文字列をセキュアに提供できる。オーケストレータは、ウェブアプリケーションコンテナだけがこの秘密情報にアクセスできるようにし、それがディスクに保存されずに、ウェブアプリがデプロイされるたびに秘密情報が提供されようにすることができる。

組織は、すでに非コンテナ環境で秘密情報を保持するために使用されている既存の企業秘密情報管理システムと、コンテナのデプロイを統合してもよい。これらのツールは通常、コンテナがデプロイされる際に秘密情報をセキュアに取得するための API を提供しており、イメージ内に秘密情報を保持する必要がない。

選択したツールに関わらず、組織は、事前に定義されて管理者が制御する設定に基づいて、秘密情報を必要とする特定のコンテナにのみ秘密情報が提供されるようにし、認証済みの暗号モジュールに含まれる連邦情報処理標準（FIPS）承認の暗号アルゴリズム[5]を使用して、保存時およびデータ送信時に常に秘密情報が暗号化されることを確実にすることが望ましい。

4.1.5 信頼できないイメージの使用

組織は、信頼できるイメージとレジストリのセットを維持し、このセットからのイメージだけが環境内での実行を許可されるようにすることで、信頼できないコンポーネントや悪意のあるコンポーネントがデプロイされるリスクを軽減することが望ましい。

これらのリスクを軽減するために、組織は以下のような多層的なアプローチをとることが望ましい。

[5] NIST-validated cryptographic implementations の詳細については、the Cryptographic Module Validation Program (CMVP) のページ https://csrc.nist.gov/groups/STM/cmvp/. を参照。

- 環境内で、信頼できるイメージとレジストリを厳密に一元管理する機能
- NIST で認証済みの実装[6]を使用した暗号署名による各イメージの個別識別
- 環境内のすべてのホストが、これらの承認済みリストのイメージのみを実行することを確実にする強制
- イメージが信頼できるソースからのものであり、改ざんされていないことを確実とするために、イメージの実行前にイメージの署名を検証
- 脆弱性や構成要件の変更に応じて、リポジトリ内のイメージがメンテナンスされ更新されていることを確実とするための継続的なモニタリングとメンテナンス

4.2 レジストリの対策

4.2.1 レジストリへのセキュアでない接続

組織は、暗号化されたチャネルでのみレジストリに接続するよう、開発ツール、オーケストレータ、およびコンテナランタイムを設定するのが望ましい。具体的な手順はツールによって異なるが、重要な目標は、レジストリにプッシュされたり、レジストリからプルされたりするすべてのデータが、信頼できるエンドポイント間でやりとりされ、伝送中に暗号化されていることを確実とすることである。

4.2.2 レジストリ内の古いイメージ

古くなったイメージを使用するリスクは、2 つの基本的な方法で軽減することができる。一つ目の方法は、組織が、もう使用することがない、脆弱性があって安全ではないイメージの登録を削除することである。このプロセスは、時間をトリガーとして、イメージに関連付けられたラベルに基づいて自動化することができる。2 つ目の方法は、運用プラクティスにおいて、使用するイメージの個別のバージョンを特定するイミュータブルな名前を使用してイメージにアクセスすることが望ましい。例えば、my-app と呼ばれるイメージを使用するようにデプロイのジョブを設定するのではなく、my-app:2.3 や my-app:2.4 のように、特定のバージョンのイメージをデプロイするように設定し、各ジョブの一部として、明確に良品であるイメージのインスタンスが確実にデプロイされるようにする。

もう一つのオプションは、イメージに「最新」タグを使用し、デプロイの自動化でこのタグを参照することである。しかし、このタグはイメージにつけられたラベルに過ぎず、最新であることを保証するものではないため、組織はこのタグを過度に信頼しないよう注意することが望ましい。組織が個別の名前を使用するか、「最新」タグを使用するかに関わらず、自動化が最新で固有の名前を使用していることや、「最新」とタグ付けされたイメージが実際に最新のバージョンを表していることを確実にするためのプロセスが導入されていることが重要である。

4.2.3 不十分な認証・認可制限

プロプライエタリなイメージや機微なイメージを含むレジストリへのアクセスは全て、認証を必要とすることが望ましい。信頼できるエンティティからのイメージのみを確実にレジストリに追加できるよう、レジストリへの書き込みアクセスはすべて認証を必要とするのが望ましい。例えば、開発者には、どのリポジトリも更新できるようにするのではなく、自分が担当する特定のリポジトリにだけイメージをプッシュできるようにする。

[6] NIST-validated cryptographic implementations の詳細については，the Cryptographic Module Validation Program (CMVP) のページ https://csrc.nist.gov/projects/cryptographic-module-validation-program. を参照。

組織は、自社のアカウントやクラウドサービスのプロバイダのディレクトリサービスなどの既存のアカウントとの連携を考慮し、それらのアカウントに対してすでに実施されているセキュリティ管理を活用することが望ましい。レジストリへの書き込みアクセスはすべて監査され、機微なイメージの読み取り操作も同様にログに記録されることが望ましい。

レジストリはまた、コンテキストに対応した認可の制御の適用をアクションに提供する。例えば、組織は、継続的インテグレーションプロセスを構成して、イメージが脆弱性スキャンとコンプライアンス評価に合格した後にのみ、権限を与えられた担当者によって署名され、レジストリにプッシュされるようにすることができる。組織はこれらの自動スキャンをプロセスに統合し、脆弱性のあるイメージや設定ミスのあるイメージの昇格やデプロイを防ぐことが望ましい。

4.3 オーケストレータの対策

4.3.1 無制限の管理アクセス

特に、オーケストレータは制御範囲が広いため、最小権限のアクセスモデルを使用することが望ましい。このモデルでは、ユーザには、ジョブロールが必要とする特定のホスト、コンテナ、イメージに対して、特定のアクションを実行する能力のみが与えられる。例えば、テストチームのメンバには、テストに使用されるイメージと、その実行に使用されるホストへのアクセス権のみを与え、作成したコンテナのみを操作できるようにすることが望ましい。テストチームのメンバは、、本番環境で使用されるコンテナへのアクセスが制限されているか、アクセスできないことが望ましい。

4.3.2 認可されていないアクセス

クラスタ全体の管理アカウントへのアクセスは、環境内のすべてのリソースに影響を与えることができるため、厳重に管理することが望ましい。組織は、パスワードだけではなく多要素認証を要求するなど、強力な認証方法を使用することが望ましい。

組織は、適用可能な場合には、シングルサインオンを実装することが望ましい。シングルサインオンは、オーケストレータの認証を簡素化し、ユーザが強力な認証情報をより簡単に使用できるようにし、アクセスの監査を一元化することで、異常検知をより効果的にする。

保存されているデータの暗号化のための従来のアプローチでは、多くの場合、コンテナと互換性がないかもしれないホストベースの機能を使用する。したがって組織は、コンテナで使用されるデータを暗号化するためのツールを使用して、実行中のノードに関係なくコンテナからデータに適切にアクセスできるようにすることが望ましい。このような暗号化ツールは、NIST SP 800-111 [19] で定義されているものと同じ暗号化アプローチを使用して、認可されていないアクセスや改ざんに対して同じ防壁を提供することが望ましい。

4.3.3 コンテナ間ネットワークトラフィックの不十分な分離

オーケストレータは、ネットワークトラフィックを機微性のレベルごとに個別の仮想ネットワークに分離するように設定されることが望ましい。アプリごとにセグメント化することも可能だが、ほとんどの組織やユースケースでは、機微性のレベルごとにネットワークを定義するだけで、管理可能な程度の複雑さでリスクを十分に軽減することがで

きる。例えば、社外に公開されるアプリは、仮想ネットワークを共有し、社内のアプリは別の仮想ネットワークを使用し、両者間の通信は少数の明確に定義されたインタフェースを介して行われることが望ましい。

4.3.4 ワークロードの機微性レベルの混合

オーケストレータは、機微性のレベルによって、特定のホストへのデプロイを分離するよう設定されることが望ましい。これを実行するための個別のアプローチは、使用しているオーケストレータによって異なるが、一般的なモデルは、高い機微性のワークロードが、低い機微性のワークロードを実行しているホストと同じホストに配置されることを防ぐルールを定義することである。これは、オーケストレータ内でホストの「ピニング（pinning）」を使用したり、機微性レベルごとに個別に管理されたクラスタを使用したりすることによって実現できる。

ほとんどのコンテナランタイム環境は、コンテナ同士やホスト OS からコンテナを分離する効果的な役割を果たしているが、場合によっては、機微性のレベルが異なるアプリを同じホスト OS 上で実行することは、不必要なリスクとなることがある。目的、機微性、および脅威の態勢によってコンテナをセグメント化することで、さらなる多層防御が提供される。アプリのデプロイを計画する際には、アプリの階層化、ネットワークとホストのセグメンテーションなどの概念を考慮することが望ましい。例えば、あるホストが財務データベースと社外に公開されたブログの両方のためにコンテナを実行しているとする。通常、コンテナランタイムはこれらの環境を互いに効果的に分離するが、DevOps チームには、各アプリをセキュアに運用して不必要なリスクを排除する共同責任がある。ブログアプリが攻撃者によって侵害された場合、2 つのアプリが同じホスト上で動作していれば、データベースを保護するための防御層は、はるかに少なくなってしまう。

したがって、コンテナを関連する機微性でグループ化し、所与のホストカーネルが単一の機微性レベルのコンテナのみを実行することを確実とすることが、ベストプラクティスである。このセグメント化は、複数の物理サーバを使用することで実現できるかもしれないが、最新のハイパーバイザは、これらのリスクを効果的に軽減するのに十分で強力な分離も提供している。先の例では、これは組織がコンテナに対して 2 つの機微性のレベルを持っていることを意味しているかもしれない。1 つは財務アプリ用で、データベースはそのグループに含まれている。もう 1 つはウェブアプリ用で、ブログはそのグループに含まれている。組織は、それぞれが単一の重大度レベルのコンテナをホストする 2 つの VM のプールを持つことになる。例えば、vm-financial と呼ばれるホストは、財務データベースと税務申告ソフトウェアを実行しているコンテナをホストし、vm-web と呼ばれるホストは、ブログと公開用のウェブサイトをホストする。

この方法でコンテナをセグメント化することによって、セグメントの 1 つを侵害する攻撃者が、その侵害を他のセグメントに拡大することはかなり困難となる。一つのサーバを侵害する攻撃者は、同様の機微性レベルの他のコンテナに対して偵察や攻撃を実行できる限られた能力は有するだろうが、それを超えた更なるアクセスはできないだろう。この手法では、キャッシュや一時ファイル用にマウントされたローカルボリュームなどの残りのデータも、データのセキュリティゾーン内に確実に留まる。先の例では、このゾーニングは、ローカルにキャッシュされた財務データと、コンテナ終了後に残された財務データが、より低い機微性レベルのアプリを実行しているホストでは決して利用できないことを確かにするであろう。

何百ものホストや何千ものコンテナを持つ大規模な環境では、このセグメンテーションを自動化して運用する必要がある。幸い、一般的なオーケストレーションプラットフォームには通常、アプリをグループ化できるという概念が含まれており、コンテナセキュリティツールは、コンテナ名やラベルなどの属性を使用して、アプリ全体にセキュリティポリシーを適用できる。このような環境では、単純なホストの分離を超えて多層防御の追加層が、このセグメンテ

ーションを活用する場合もある。例えば、組織は独立したホスティングゾーンかネットワークを実装し、ハイパーバイザ内でこれらのコンテナを分離するだけでなく、ネットワークトラフィックをより個別に分離し、ある機微性レベルのアプリへのトラフィックと、他の機微性レベルへのトラフィックを分けることができる。

4.3.5 オーケストレータのノードの信頼

オーケストレーションプラットフォームは、実行するすべてのアプリに対してセキュアな環境を作成する機能を提供するように設定されることが望ましい。オーケストレータは、ノードがクラスタにセキュアに導入され、ライフサイクル全体を通じて永続的な同一性を保ち、ノードとその接続状態の正確な全てのインベントリを提供できることを確実にすることが望ましい。組織は、オーケストレーションプラットフォームが、クラスタ全体のセキュリティを損なうことなく、個々のノードの侵害に対してレジリエンスを持つように特別に設計されていることを確実とすることが望ましい。侵害されたノードは、クラスタ全体の運用を中断させたり、低下させたりすることなく、クラスタから分離および削除できなければならない。最後に、組織は、クラスタのメンバ間で相互認証されたネットワーク接続と、クラスタ間トラフィックへエンドツーエンドの暗号化を提供するオーケストレータを選択することが望ましい。コンテナは移植性が高いため、多くのデプロイが、組織が直接管理していないネットワーク上で行われる可能性がある。したがってこのシナリオでは、セキュアバイデフォルトの姿勢が特に重要である。

4.4 コンテナの対策

4.4.1 ランタイムソフトウェア内の脆弱性

コンテナランタイムは、脆弱性が注意深くモニタリングされ、問題が検知された場合には迅速に修正されなければならない。脆弱性のあるランタイムは、ホスト自体だけでなく、サポートしているすべてのコンテナを潜在的に重大なリスクにさらすことになる。組織は、ツールを使用して、デプロイされたランタイムの共通脆弱性識別子（CVE）の脆弱性を探し、リスクのあるインスタンスをアップグレードし、オーケストレータが適切にメンテナンスされたランタイムへのデプロイのみを許可することを確実とすることが望ましい。

4.4.2 コンテナからの無制限ネットワークアクセス

組織は、コンテナが送信する外向けのネットワークトラフィック (egress network traffic) を制御することが望ましい。少なくとも、これらの制御はネットワークの境界で行い、コンテナが従来のアーキテクチャで使用していたパターンと同様に、セキュアなデータをホストする環境からインターネットへ送信するなど、機微性のレベルが異なるネットワーク間でトラフィックを送信できないことを確実にすることが望ましい。しかし、コンテナ間のトラフィックの仮想化されたネットワーキングモデルは、さらなる課題をもたらす。

通常、複数のホストにまたがってデプロイされたコンテナは、仮想の暗号化されたネットワークを介して通信するため、従来のネットワークデバイスではこのトラフィックを把握できないことが多い。さらに、コンテナには通常、オーケストレータによってデプロイされると自動的に動的な IP アドレスが割り当てられ、これらのアドレスは、アプリのスケーリングや負荷分散に応じて継続的に変化する。したがって、理想的には、組織は既存のネットワークレベルのデバイスと、よりアプリに対応したネットワークフィルタリングを組み合わせて使用することが望ましい。アプリに対応したツールは、コンテナ間のトラフィックを確認するだけでなく、コンテナ内で実行されているアプリのあ

る特性に基づいて、このトラフィックをフィルタリングするために使用されるルールを動的に生成できることが望ましい。このような動的なルール管理は、コンテナ化されたアプリのスケールと変更の頻度、およびその一時的なネットワーキングトポロジのために重要である。

具体的には、アプリに対応したツールは、以下の機能を提供することが望ましい。

- インバウンドポートとプロセスポートのバインディングの両方を含む、適切なコンテナネットワーキングサーフェスの自動決定。
- 「有線」のトラフィックとカプセル化されたトラフィックの両方を介した、コンテナと他のネットワークエンティティ間のトラフィックフローの検知。
- 組織のネットワーク内の予期しないトラフィックフロー、ポートスキャン、または潜在的に危険な宛先へのアウトバウンドアクセスなどのネットワークの異常の検知。

4.4.3 セキュアでないコンテナランタイムの設定

組織は、コンテナランタイム設定の標準への準拠を自動化することが望ましい。
Center for Internet Security Docker Benchmark [20] などの文書化された技術的な実装ガイダンスは、オプションと推奨設定の詳細を提供しているが、このガイダンスの運用は自動化に依存する。組織は様々なツールを使用して、ある時点で「スキャン」してコンプライアンスを評価することができるが、そのようなアプローチではスケーリングできない。代わりに、組織は環境全体の設定を継続的に評価し、それらを積極的に実施するツールまたはプロセスを使用することが望ましい。

さらに、SELinux [21] や AppArmor [22] のような強制アクセス制御（MAC）技術は、Linux OS を実行するコンテナの強化された制御と分離を提供する。例えば、これらの技術を使用して、セグメンテーションを追加し、コンテナが特定のファイルパス、プロセス、およびネットワークソケットにのみアクセスできるようにして、侵害されたコンテナがホストや他のコンテナに影響を与える能力をさらに制限することができる。MAC 技術は、ホスト OS レイヤでの保護を提供し、特定のファイル、パス、プロセスのみがコンテナ化されたアプリにアクセスできるようにする。組織は、すべてのコンテナのデプロイにおいて、ホスト OS が提供する MAC 技術を使用することが推奨される。

セキュアコンピューティング（seccomp）[7] プロファイルは、コンテナが実行時に割り当てられるシステムレベルの機能を制限するために使用できるもう一つのメカニズムである。Docker などの一般的なコンテナランタイムには、安全ではなく、通常コンテナの操作には不要なシステムコールを制限するデフォルトの seccomp プロファイルが含まれている。さらに、カスタムプロファイルを作成してコンテナランタイムに渡すことで、機能をさらに制限することができる。組織は、少なくとも、コンテナがランタイムによって提供されるデフォルトのプロファイルで実行されることを確実とし、リスクの高いアプリには追加のプロファイルを使用することを考慮することが望ましい。

4.4.4 アプリの脆弱性

既存のホストベースの侵入検知プロセスやツールは、前述のとおり技術的なアーキテクチャや運用方法が異なるため、コンテナ内の攻撃を検知して防止することができないことがよくある。組織は、コンテナを認識し、コンテナで一般的に見られるスケールと変更の頻度で動作するように設計された追加のツールを実装することが望ましい。これ

[7] seccomp の詳細については、https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt. を参照。

らのツールは、振る舞いの学習を使用してコンテナ化されたアプリを自動的にプロファイリングし、セキュリティプロファイルを構築して、人間の操作を最小限に抑えることができるようにする。これらのプロファイルは、以下のようなイベントを含む実行時の異常を検知し、防止できることが望ましい。

- 無効な、または予期せぬプロセスの実行
- 無効な、また予期せぬシステムコール
- 保護された設定ファイルとバイナリの変更
- 予期せぬ場所やファイルタイプへの書き込み
- 予期せぬネットワークリスナーの作成
- 予期せぬネットワークの宛先に送信されたトラフィック
- マルウェアの保存または実行

コンテナは、ルートファイルシステムを読み取り専用モードで実行することが望ましい。このアプローチは、具体的に定義されたディレクトリへの書き込みを分離し、前述のツールでより簡単にモニタリングすることができる。さらに、読み取り専用のファイルシステムを使用すると、改ざんはこれら特定の場所に分離され、アプリの残りの部分から簡単に分離できるため、コンテナはより侵害に強くなる。

4.4.5 未承認コンテナ

組織は、開発、テスト、本番、およびその他のシナリオに対して個別の環境を構築し、それぞれの環境で、コンテナのデプロイおよび管理といったアクティビティに対する役割ベースのアクセス制御を提供するための具体的な制御を行うことが望ましい。アクティビティの明確な監査証跡を提供するために、すべてのコンテナ作成は、個々のユーザ ID に関連付けてログに記録することが望ましい。さらに、組織は、イメージの実行を許可する前に、脆弱性管理とコンプライアンスのためのベースライン要件を強制できるセキュリティツールを使用することが推奨される。

4.5 ホスト OS の対策

4.5.1 大きなアタックサーフェス

コンテナ専用 OS を使用している組織の場合、OS はコンテナをホストするために特別に設計されており、他のサービスや機能が無効化されているため、脅威は通常、最初は最小限に抑えられている。さらに、これらの最適化された OS は、コンテナをホストするために特別に設計されているため、通常は読み取り専用のファイルシステムをもっており、デフォルトでその他の強化プラクティスも採用している。組織は可能な限り、これらの最小限の OS を使用してアタックサーフェスを減らし、汎用 OS に関連する典型的なリスクと強化アクティビティを軽減することが望ましい。

コンテナ専用 OS を使用できない組織は、NIST SP 800-123, *Guide to General Server Security* [23]のガイダンスに従って、ホストのアタックサーフェスを可能な限り減らすことが望ましい。例えば、コンテナを実行するホストは、コンテナのみを実行し、ウェブサーバやデータベースなどのコンテナの外の他のアプリを実行しないようにすることが望ましい。また、ホスト OS は、プリントスプーラのような、アタックサーフェスやパッチの適用領域を増やすような不要なシステムサービスを実行することは望ましくない。最後に、ホストは継続的に脆弱性をスキャンし、コンテ

ナランタイムだけでなく、コンテナがセキュアで区分化された運用を行うために依存しているカーネルのような下位レベルのコンポーネントに対しても、迅速にアップデートを適用することが望ましい。

4.5.2 共有カーネル

コンテナのワークロードを機微性レベルでホストにグループ化することに加えて、組織はコンテナ化されたワークロードとコンテナ化されていないワークロードを同じホストインスタンス上に混在させることは望ましくない。例えば、ホストが ウェブサーバコンテナを実行している場合、ホスト OS 内に定常的に直接インストールされたコンポーネントとしてウェブサーバ（またはその他のアプリ）を実行することは望ましくない。コンテナ化されたワークロードをコンテナ専用のホストに分離しておくことで、コンテナを保護するために最適化された対策や防御策をより簡単かつ安全に適用することができる。

4.5.3 ホスト OS コンポーネントの脆弱性

組織は、ベース OS の管理と機能のために提供されるコンポーネントのバージョンを検証するための、管理プラクティスとツールを実装することが望ましい。コンテナ専用 OS は、汎用 OS に比べてはるかに最小限のコンポーネントのセットであるが、脆弱性を有しており、修正が必要である。組織は、OS ベンダやその他の信頼できる組織が提供するツールを使用して、OS 内で使用されるすべてのソフトウェアコンポーネントを定期的にチェックし、アップデートを適用することが望ましい。OS は、セキュリティアップデートだけでなく、ベンダが推奨する最新のコンポーネントアップデートも含めて、常に最新の状態に保つことが望ましい。これらのコンポーネントの新しいリリースでは、単に脆弱性を修正するだけでなく、セキュリティ保護や機能が追加されることが多いため、これはカーネルやコンテナランタイムのコンポーネントにとって特に重要である。一部の組織では、既存のシステムをアップデートするのではなく、必要なアップデートを施した新しい OS インスタンスを単に再デプロイすることを選択する場合がある。このアプローチも有効だが、より高度な運用方法が必要となることが多い。

ホスト OS は、データや状態がホスト上に一意かつ持続的に保存されず、ホストによるアプリケーションレベルでの依存関係もない、イミュータブルな方法で運用することが望ましい。その代わりに、すべてのアプリのコンポーネントと依存関係はコンテナにパッケージ化してデプロイすることが望ましい。これにより、ホストはほぼステートレスな方法で運用され、アタックサーフェスを大幅に減らすことができる。さらに、異常やコンフィギュレーションドリフトを識別するためのより信頼できる方法を提供する。

4.5.4 不適切なユーザアクセス権

ほとんどのコンテナのデプロイは、ホスト間でジョブを分散するためにオーケストレータに依存しているが、組織は OS へのすべての認証を監査し、ログインの異常をモニタリングし、特権操作を実行するための昇格がすべてログに記録されることを確実とすることが望ましい。これにより、個人がホストに直接ログオンし、特権コマンドを実行してコンテナを操作するなどの異常なアクセスパターンを識別することができる。

4.5.5 ホストファイルシステムの改ざん

コンテナが必要な最小限のファイルシステムの権限のセットで実行されていることを確実とする。コンテナがローカルファイルシステムをホストにマウントすることはほとんどない。代わりに、コンテナがディスクに残す必要がある

ファイルの変更は、この目的のために特別に割り当てられたストレージのボリューム内で行われる。コンテナがホストのファイルシステム上の機微性の高いディレクトリ、特にオペレーティングシステムの構成設定を含むディレクトリを決してマウントできるようにしてはならない。

組織は、コンテナによってマウントされているディレクトリをモニタリングし、これらのポリシーに違反するコンテナのデプロイを防止できるツールを使用することが望ましい。

4.6 ハードウェアの対策

NIST SP 800-164 [24]で認識されているように、ソフトウェアベースのセキュリティは定期的に破られている。NISTは、NIST SP 800-147 [25]、800-155 [26]、および 800-164 でトラステッドコンピューティングの要件を定義している。NIST では「トラステッド」とは、ソフトウェアのインベントリが正確であること、構成設定とセキュリティ制御が適切に行われていること、および望ましい動作をしていることなど、プラットフォームが期待通りに動作することを意味する。また「トラステッド」は、不正な人物がホスト上のソフトウェアやその設定を改ざんしていないことがわかっていることも意味する。ハードウェアの信頼の基点（root of trust）はコンテナに特有の概念ではないが、コンテナ管理およびセキュリティツールは、コンテナ技術アーキテクチャ以外の部分に構成証明（attestation）を活用して、コンテナがセキュアな環境で実行されていることを確実とする。

現在、トラステッドコンピューティングを提供するために利用可能な方法は、以下の通りである。

1. ファームウェア、ソフトウェア、および構成データを、計測のための信頼の基点（RTM：Root of Trust of Measurement）を使用して、実行前に計測する。
2. それらの計測値を trusted platform module（TPM）などのハードウェアの信頼の基点に保存する。
3. 現在の計測値が、予想される計測値と一致していることを検証する。一致していれば、プラットフォームが期待通りに動作することを信頼できることが証明される。

TPM 対応デバイスは、ブートプロセス中にマシンの完全性をチェックすることができ、ハードウェア、プリブート時、セキュアブートプロセスで、保護と検知のメカニズムを使用することができる。これと同じ信頼性と完全性の保証は、OS やブートローダを超えて、コンテナのランタイムやアプリにも拡張することができる。ここで留意すべきことは、クラウドサービスのユーザによるハードウェアの信頼の検証を可能にする標準は開発中であり、すべてのクラウドがこの機能を顧客に公開しているわけではないということである。技術的な検証が提供されていない場合、組織は、クラウドプロバイダとのサービス契約の一部として、ハードウェアの信頼性要件に対処することが望ましい。

システムが複雑化し、今日の脅威の深く埋め込まれるという特質から、セキュリティは、ハードウェアとファームウェアから始まるコンテナ技術のすべてのコンポーネントに広げることが望ましい。これによって、分散型のトラステッドコンピューティングモデルが形成され、コンテナの構築、実行、オーケストレーション、管理を行うための最も信頼されたセキュアな方法が提供される。

トラステッドコンピューティングモデルは、検証されたシステムプラットフォームを提供する計測／セキュアブートから始まり、ハードウェアの信頼の基点の連鎖を構築し、ブートローダ、OS カーネル、OS コンポーネントに拡張して、ブートメカニズム、システムイメージ、コンテナランタイム、コンテナイメージの暗号を使った検証を可能とすることが望ましい。コンテナ技術の場合、これらの手法は現在、ハードウェア、ハイパーバイザ、およびホスト

OS レイヤで適用可能であり、コンテナ専用のコンポーネントにこれらを適用するための初期の作業が進行中である。

本文書執筆時点で、NIST は業界のパートナーと協力して、コンテナ環境のためのトラステッドコンピューティングモデルを実証する市販の製品をベースとしたリファレンスアーキテクチャを構築している。[8]

[8] この分野における過去の NIST の取り組みについては、NIST IR 7904, Trusted Geolocation in the Cloud: Proof of Concept Implementation [27]. を参照。

5 コンテナの脅威シナリオの例

4 節で推奨した対策の有効性を説明するために、以下のコンテナの脅威シナリオの例を考えてみる。

5.1 イメージ内の脆弱性の悪用

コンテナ化された環境に対する最も一般的な脅威の一つは、コンテナ内のソフトウェアにおけるアプリケーションレベルの脆弱性である。例えば、ある組織が一般的なウェブアプリをベースにイメージを構築することがある。そのアプリに脆弱性がある場合、コンテナ内のアプリを破壊するために使用される可能性がある。いったん侵害されると、攻撃者は環境内のシステムをマッピングしたり、侵害されたコンテナ内で特権を昇格させたり、コンテナを悪用して他のシステムへの攻撃に利用したりすることができる（ファイルのドロッパー、またはコマンドアンドコントロールのエンドポイントとして機能する、など）。

推奨される対策を採用している組織は、上記の脅威に対して、多層防御の複数のレイヤを持つことになる。

1. デプロイのプロセスの早い段階で脆弱なイメージを検知し、脆弱なイメージのデプロイを防止するための制御を実施することで、本番環境に脆弱性が持ち込まれるのを防ぐことができる。
2. コンテナに対応したネットワークモニタリングとフィルタリングにより、他のシステムをマッピングしようとする際に、他のコンテナへの異常な接続を検知する。
3. コンテナに対応したプロセスモニタリングとマルウェア検知により、無効または予期しない悪意のあるプロセスの実行と、それらが環境に取り込むデータを検知する。

5.2 コンテナランタイムの悪用

まれなことではあるが、もしコンテナランタイムが侵害された場合、攻撃者はこのアクセスを利用してホスト上のすべてのコンテナ、さらにホスト自体を攻撃することができる。

この脅威のシナリオに関連した緩和策としては、以下のようなものがある。

1. 強制アクセス制御の機能を使用すると、プロセスとファイルシステムのアクティビティが、定義された境界内で、引き続きセグメント化されることを確実にするための追加の防壁を提供することができる。
2. ワークロードのセグメンテーションは、障害の範囲が、ホストを共有している共通の機微性レベルのアプリに限定されることを確実にする。例えば、ウェブアプリのみを実行しているホスト上の侵害されたランタイムは、財務アプリのコンテナを実行している他のソフトのランタイムには影響しない。
3. ランタイムの脆弱性の状態を報告し、脆弱なランタイムへのイメージのデプロイを防ぐことができるセキュリティツールは、そこでのワークロードの実行を防ぐことができる。

5.3 侵害されたイメージの実行

イメージは公開されている場所から容易に入手でき、出所が不明な場合が多いため、攻撃者は標的が使用することがわかっているイメージに、悪意のあるソフトウェアを埋め込むことができる。例えば、攻撃者が、特定のプロジェク

トに関する掲示板で標的が活動しており、そのプロジェクトのウェブサイトで提供されたイメージを使用していると判断した場合、これらのイメージの悪意のあるバージョンを作成して攻撃に使用しようとする可能性がある。

関連する緩和策は以下のとおりである。

1. 入念に検査、テスト、検証され、デジタル署名されたイメージのみが、組織のレジストリへのアップロードを許可されることを確実とする。
2. 信頼されたイメージのみを実行できるようにすることを確実とし、外部の未検証のソースからのイメージの使用を防止する。
3. イメージの脆弱性やマルウェアを自動的にスキャンし、イメージ内に埋め込まれたルートキットなどの悪意のあるコードを検知する。
4. リソースの悪用、特権の昇格、および実行ファイルの実行といったコンテナの機能を制限するランタイムコントロールを実装する。
5. 侵害されたイメージが影響を及ぼす恐れがある「影響の到達範囲」を制限するために、コンテナレベルのネットワークセグメンテーションを使用する。
6. コンテナランタイムが、最小特権および最小アクセスの原則に従って動作することを確認する。
7. コンテナランタイムの脅威プロファイルを作成する。これには、プロセス、ネットワークコール、ファイルシステムの変更が含まれるが、これらに限定されない。
8. ハッシュやデジタル署名を利用して、実行前にイメージの完全性を確認する。
9. 許容できる脆弱性の深刻度レベルを定めた基準に基づいて、イメージの実行を制限する。例えば、中程度以上の CVSS 評価を持つ脆弱性があるイメージを実行できないようにする。

6 コンテナ技術のライフサイクルにおけるセキュリティに関する考慮事項

コンテナ技術のインストール、設定、およびデプロイを行う前に、慎重に計画を立てることが非常に重要である。これにより、コンテナ環境が可能な限りセキュアであり、関連するすべての組織のポリシー、外部規制、およびその他の要件に準拠していることを確実にすることができる。

コンテナ技術と仮想化ソリューションの計画と実装に関する推奨事項には、多くの類似点がある。NIST SP 800-125 [1]の 5 節には、すでに仮想化ソリューションに関する推奨事項の一式が含まれている。ここでは、これらのすべての推奨事項を繰り返す代わりに、読者にその文書を示し、以下に記載する例外を除いて、組織は NIST SP 800-125 の 5 節の推奨事項をすべてコンテナ技術のコンテキストに適用することが望ましいことを示す。例えば、仮想化セキュリティポリシーを作成する代わりに、コンテナ技術セキュリティポリシーを作成する。

本節では、NIST SP 800-125 の 5 節 の推奨事項に対する例外と追加事項を、計画と実装のライフサイクルの対応するフェーズごとにグループ化して記載する。

6.1 開始フェーズ

組織は、他のセキュリティポリシーがコンテナによってどのような影響を受けるかを検討し、コンテナを考慮に入れるために、必要に応じてこれらのポリシーを調整することが望ましい。例えば、インシデントレスポンス（特にフォレンジック）や脆弱性管理のポリシーは、コンテナの特殊な要件を考慮するための調整が必要となる場合がある。

コンテナ技術の導入は、組織内の既存の文化とソフトウェア開発方法論を混乱させる可能性がある。コンテナが提供するメリットを最大限に活用するためには、この新しいアプリの開発、実行、サポート方法をサポートするように組織のプロセスを調整することが望ましい。従来の開発手法、パッチ適用技術、およびシステムのアップグレードプロセスは、コンテナ化された環境に直接適用されない場合があり、組織内の従業員が新しいモデルに積極的に適応することが重要である。新しいプロセスでは、技術の変化によってもたらされる潜在的なカルチャーショックを考慮することができ、また、対処できる。ソフトウェア開発ライフサイクルに関わるすべての人に教育とトレーニングを提供することによって、人々がアプリを構築、配布、実行するための新しい方法に慣れることができる。

6.2 計画・設計フェーズ

計画と設計フェーズでの最初のコンテナ特有の考慮事項は、フォレンジックである。コンテナはほとんどが OS にすでに存在するコンポーネントの上に構築されるため、コンテナ化された環境でフォレンジックを実行するためのツールや技術は、ほとんどが既存のプラクティスを進化させたものである。コンテナとイメージのイミュータブルは、イメージが実行すべきこととインシデント中に実際に発生したこととの境界がより明確になるため、実質的にフォレンジック能力を向上させることができる。例えば、ウェブサーバを実行するために起動されたコンテナが突然メールリレーを開始した場合、その新しいプロセスがコンテナの作成に使用された元のイメージの一部ではないことは明らかである。OS とアプリの分離が少ない従来のプラットフォームでは、このような区別を行うことは非常に困難である。

プロセス、メモリ、およびディスクのインシデントレスポンスアクティビティに精通している組織では、コンテナを使用して作業する際も、ほぼ同様のアクティビティとなることが理解できるだろう。しかし、注意すべき違いもいくつかある。

コンテナは通常、ホスト OS から仮想化されたレイヤ化されたファイルシステムを使用している。ホスト上のパスを調べても、一般的にはこれらのレイヤの外側の境界しか見えず、その中のファイルやデータは見えない。したがって、コンテナ化された環境でインシデントに対応する際には、ユーザは使用中の特定のストレージプロバイダを識別し、そのコンテンツをオフラインで適切に調べる方法を理解しておくことが望ましい。

通常、複数のコンテナは、仮想化されたオーバーレイネットワークを使用して互いに接続されている。これらのオーバーレイネットワークでは、トラフィックが既存のネットワーク上をセキュアにルーティングできるよう、カプセル化と暗号化が頻繁に使用されている。しかし、これは、コンテナネットワーク上のインシデントを調査する場合、特にライブパケット解析を行う場合、使用するツールが、これらの仮想化されたネットワークを認識し、かつ、そのネットワークから、埋め込まれた IP フレームを既存のツールで解析するために抽出する方法を理解していなければならないことを意味している。

コンテナ内でのプロセスとメモリのアクティビティは、従来のアプリ内で見られていたものとほぼ同様であるが、親プロセスが異なる。例えば、コンテナランタイムは、コンテナ内のすべてのプロセスを入れ子形式で生成する場合がある。この場合、ランタイムは最上位のプロセスであり、コンテナごとの第 1 レベルの子を持ち、コンテナ内の各プロセスは第 2 レベルの子を持つ。 例えば、以下のようになる。

```
├─containerd─┬─┬─[container1─┬─bash]
│           │ │             └─8*[{thread}]]
│           │ ├─container2─┬─start.sh─┬─mongod───22*[{mongod}]
│           │ │            │          └─node─┬─4*[{V8 WorkerThread}]
│           │ │            │                └─5*[{node}]
│           │ │            └─8*[{thread}]
│           │ ├─container3─┬─mysqld───28*[{mysqld}]
│           │ │            └─8*[{thread}]
```

6.3 実装フェーズ

コンテナ技術が設計されたら、次のステップは、ソリューションを本番に投入する前に、設計のプロトタイプを実装してテストすることである。コンテナ技術は、VM 技術が提供しているイントロスペクション機能を提供していないことに注意する。

NIST SP 800-125 [1]では、認証、接続性とネットワーキング、アプリの機能、管理、パフォーマンス、技術自体のセキュリティなど、本番環境にデプロイする前に評価することが望ましい、仮想化技術のいくつかの側面を挙げている。これらに加えて、コンテナ技術の分離機能も評価することが重要である。コンテナ内のプロセスが、許可されているすべてのリソースにアクセスでき、他のリソースを表示したり、他のリソースにアクセスしたりできないことを確実とする。

実装では、コンテナとクラウドネイティブアプリに特化し、従来のツールでは不足していた運用を可視化する新しいセキュリティツールが必要となる場合がある。最後に、デプロイには、セキュリティイベントの記録、ネットワーク

管理、コードリポジトリ、認証サーバなどの既存のセキュリティ管理や技術の構成を変更して、コンテナと直接連携したり、これらの新しいコンテナセキュリティツールと統合したりすることが含まれることもある。

プロトタイプの評価が完了し、コンテナ技術が本番で使用できる状態になったら、最初は少数のアプリでコンテナを使用することが望ましい。問題が発生した場合、複数のアプリに影響を与える可能性が高いため、早期に問題を識別しておくと、さらなるデプロイの前に対処することができる。また、段階的なデプロイは、開発者や IT スタッフ (システム管理者、ヘルプデスクなど) にコンテナの使用法やサポートについてのトレーニングを行う時間も提供する。

6.4 運用・保守フェーズ

コンテナ技術のセキュリティを維持するために特に重要であるため、定常的に実施する必要がある運用プロセスには、すべてのイメージをアップデートすること、および、アップデートしたイメージを古いイメージの代わりとしてコンテナに配布することが含まれる。ホストやオーケストレータのような他のサポートレイヤの脆弱性管理やアップデートの実施などの、その他のセキュリ ティのベストプラクティスも、重要な継続的な運用上のタスクである。コンテナのセキュリティおよびモニタリングのツールも同様に、既存のセキュリティ情報およびイベント管理（SIEM）ツールと統合し、コンテナ関連のイベントが、残りの環境全体にセキュリティを提供するために使用されるのと同じツールとプロセスに流れることを確実とすることが望ましい。

コンテナ化された環境でセキュリティインシデントが発生した場合、組織は、コンテナ専用の側面に最適化されたプロセ スとツールで対応できるように準備しておくことが望ましい。NIST SP 800-61「*Computer Security Incident Handling Guide* 」[28]で概説されているコアガイダンスは、コンテナ化された環境にも十分適用できる。しかし、コンテナを採用する組織は、コンテナのセキュリティの固有の側面のいくつかに対応するための強化を確実とすることが望ましい。

- コンテナ化されたアプリは、従来の運用チームとは異なるチームによって実行される可能性がある。したがって組織は、コンテナ運用を担当するチームがどのようなチームであっても、インシデントレスポンス計画に組み込み、そのチームが自身の役割を理解していることを確実にする。
- 本文書全体で説明されているように、コンテナ管理の一時的で自動化された性質は、組織が従来使用してきた資産管理ポリシーやツールと一致しない場合がある。インシデントレスポンスチームは、コンテナのロール、所有者、および機微性レベルを把握し、この把握したデータを自身のプロセスに統合できなければならない。
- コンテナ関連のインシデントに対応するための明確な手順を定義することが望ましい。例えば、特定のイメージが悪用されていて、そのイメージが何百ものコンテナで使用されている場合、レスポンスチームは攻撃を阻止するために、これらのコンテナをすべてシャットダウンする必要がある場合がある。一つの脆弱性が多くのシステムで問題を引き起こすことは以前からあったが、コンテナでは、既存のシステムにパッチをインストールするのではなく、新しいイメージを再構築して広く再デプロイする必要がある場合がある。この対応の変化は、様々なチームや承認が関係する可能性があるため、事前に理解を得て実施することが望ましい。
- 前述したように、記録やその他のフォレンジックデータは、コンテナ化された環境では保存方法が異なる場合がある。インシデントレスポンスチームは、データ収集に必要なさまざまなツールや技術に精通し、これらの環境に特化したプロセスを文書化しておくことが望ましい。

6.5 廃棄フェーズ

コンテナがアプリのニーズに基づいて自動的にデプロイされ、破棄される機能は、非常に効率的なシステムを可能にするが、記録保持、フォレンジック、イベントデータの要件にいくつかの課題が生じる可能性がある。組織は、データ保持ポリシーを満たすために適切なメカニズムが整備されていることを確認することが望ましい。取り組むべき課題の例としては、コンテナやイメージをどのように破棄するか、廃棄前にコンテナからどのようなデータを抽出するか、そのデータ抽出をどのように行うのか、コンテナが使用する暗号鍵をどのように無効化または削除するか、などが挙げられる。

コンテナ化された環境をサポートするデータストア及びメディアは、組織が策定する廃棄計画に含めることが望ましい。

7 結論

コンテナは、アプリの構築方法と実行方法の変革を象徴しているが、劇的に新しいセキュリティのベストプラクティスを必要とするものではない。むしろコンテナのセキュリティの最も重要な側面は、確立された技術と原則の改良である。本文書では、コンテナ技術に固有のリスクを考慮するために、一般的なセキュリティの推奨事項を更新・拡張している。

本文書では、コンテナをセキュアにする方法と、VM 内の同じアプリをセキュアにする方法とのいくつかの違いについて、すでに説明してきた。これらの点についての本文書のガイダンスをまとめておくことは有益である。

コンテナ環境では、さらに多くのエンティティが存在するため、セキュリティプロセスやツールは、それに応じてスケーリングできる必要がある。スケーリングは、データベースでサポートされるオブジェクトの総数だけでなく、ポリシーをいかに効果的かつ自律的に管理できるかを意味する。多くの組織では、何百もの VM にまたがるセキュリティ管理の負担に苦労している。コンテナ中心のアーキテクチャが一般的になり、これらの組織が何千、何万ものコンテナを担当するようになると、そのセキュリティ対策は自動化と効率性を維持することを重視することが望ましい。

コンテナを使用すると、アプリの更新は、年に数回から週に数回、あるいは一日に数回の更新へと、変更の頻度が格段に高くなる。以前は手動で行うことが許容されていたことは、もはや許容されない。自動化は、エンティティの正味の数に対応するためだけでなく、それらのエンティティの変更の頻度に対応するためにも重要である。ポリシーを一元的に表現し、環境全体にわたってポリシーの実施をソフトウェアで強制できるようにすることが極めて重要である。コンテナを採用する組織は、この変更の頻度を管理するように準備しておくことが望ましい。そのためには、根本的に新しい運用プラクティスと組織の進化が必要となる場合がある。

コンテナの使用により、セキュリティの責任の多くが開発者に移るため、組織は、開発者が適切な判断を下すために必要なすべての情報、スキル、ツールを備えることを確実とすることが望ましい。また、セキュリティチームは、開発サイクル全体を通して品質を積極的に強化できるようにすることが望ましい。この移行を成功させた組織は、これまでよりも迅速かつ少ない運用負荷で脆弱性に対応できるという、セキュリティ上のメリットを享受できる。

セキュリティはコンテナ自体と同様に移動可能でなければならないため、組織は、オープンで、プラットフォームや環境を越えて動作する技術やツールを採用することが望ましい。多くの組織では、開発者はある環境でビルドし、別の環境でテストし、さらに別の環境でデプロイすることになるため、これらすべての環境においてアセスメントと強制に一貫性を持たせることが鍵となる。また、移植性は環境だけでなく、時間的な問題でもある。継続的インテグレーションと継続的デプロイのプラクティスは、開発とデプロイのフェーズ間にあった従来の壁を取り払うため、組織は、イメージの作成、レジストリへのイメージの保存、コンテナ内でのイメージの実行において、一貫性のある自動化されたセキュリティプラクティスを確実とする必要がある。

これらの変化に対応する組織は、コンテナを活用してセキュリティ全体を実際に改善することができる。コンテナのイミュータブルと宣言的な性質により、組織は、手作業の関与が最小限で済み、アプリの変更に応じて自身を更新する、より自動化されたアプリ中心のセキュリティ構想の実現を開始できる。コンテナは、受動的で手動の高コストなセキュリティモデルから、より優れたスケールと効率性を実現し、リスクを軽減するモデルへと移行する組織にとって、有効な機能である。

付録 A – 非コアコンポーネントを保護するための NIST リソース

この付録は、開発者システム、テストおよび認定システム、管理者システム、ホストのハードウェアおよび仮想マシンのマネージャなど、非コアなコンテナ技術コンポーネントをセキュアにするための NIST のリソースを記載している。さらに多くのリソースが他の組織から提供されている。

表 1：非コアコンポーネントをセキュアにするための NIST リソース

リソース名と URI	適用可能性
SP 800-40 Revision 3, *Guide to Enterprise Patch Management Technologies* https://doi.org/10.6028/NIST.SP.800-40r3	すべての IT 製品とシステム
SP 800-46 Revision 2, *Guide to Enterprise Telework, Remote Access, and Bring Your Own Device (BYOD) Security* https://doi.org/10.6028/NIST.SP.800-46r2	クライアント OS、クライアントアプリ
SP 800-53 Revision 4, *Security and Privacy Controls for Federal Information Systems and Organizations* https://doi.org/10.6028/NIST.SP.800-53r4	すべての IT 製品とシステム
SP 800-70 Revision 3, *National Checklist Program for IT Products: Guidelines for Checklist Users and Developers* https://doi.org/10.6028/NIST.SP.800-70r3	サーバ OS、クライアント OS、サーバアプリ、クライアントアプリ
SP 800-83 Revision 1, *Guide to Malware Incident Prevention and Handling for Desktops and Laptops* https://doi.org/10.6028/NIST.SP.800-83r1	クライアント OS、クライアントアプリ
SP 800-123, *Guide to General Server Security* https://doi.org/10.6028/NIST.SP.800-123	サーバ
SP 800-124 Revision 1, *Guidelines for Managing the Security of Mobile Devices in the Enterprise* https://doi.org/10.6028/NIST.SP.800-124r1	モバイル機器
SP 800-125, *Guide to Security for Full Virtualization Technologies* https://doi.org/10.6028/NIST.SP.800-125	ハイパーバイザと仮想マシン
SP 800-125A, *Security Recommendations for Hypervisor Deployment (Second Draft)* https://csrc.nist.gov/publications/detail/sp/800-125A/draft	ハイパーバイザと仮想マシン
SP 800-125B, *Secure Virtual Network Configuration for Virtual Machine (VM) Protection* https://doi.org/10.6028/NIST.SP.800-125B	ハイパーバイザと仮想マシン
SP 800-147, *BIOS Protection Guidelines* https://doi.org/10.6028/NIST.SP.800-147	クライアントハードウェア
SP 800-155, *BIOS Integrity Measurement Guidelines* https://csrc.nist.gov/publications/detail/sp/800-155/draft	クライアントハードウェア
SP 800-164, *Guidelines on Hardware-Rooted Security in Mobile Devices* https://csrc.nist.gov/publications/detail/sp/800-164/draft	モバイル機器

付録 B – コンテナ技術に関連する NIST SP 800-53 と NIST サイバーセキュリティフレームワークのセキュリティ管理策

NIST SP 800-53 Revision 4 [29] のセキュリティ管理のうち、コンテナ技術にとって最も重要なものを表 2 に示す。

表 2：NIST SP 800-53 によるコンテナ技術のセキュリティ管理策

NIST SP 800-53 の管理策	関連する管理策	参考文献
AC-2, Account Management	AC-3, AC-4, AC-5, AC-6, AC-10, AC-17, AC-19, AC-20, AU-9, IA-2, IA-4, IA-5, IA-8, CM-5, CM-6, CM-11, MA-3, MA-4, MA-5, PL-4, SC-13	
AC-3, Access Enforcement	AC-2, AC-4, AC-5, AC-6, AC-16, AC-17, AC-18, AC-19, AC-20, AC-21, AC- 22, AU-9, CM-5, CM-6, CM-11, MA-3, MA-4, MA-5, PE-3	
AC-4, Information Flow Enforcement	AC-3, AC-17, AC-19, AC-21, CM-6, CM-7, SA-8, SC-2, SC-5, SC-7, SC-18	
AC-6, Least Privilege	AC-2, AC-3, AC-5, CM-6, CM-7, PL-2	
AC-17, Remote Access	AC-2, AC-3, AC-18, AC-19, AC-20, CA-3, CA-7, CM-8, IA-2, IA-3, IA-8, MA-4, PE-17, PL-4, SC-10, SI-4	NIST SPs 800-46, 800-77, 800-113, 800-114, 800-121
AT-3, Role-Based Security Training	AT-2, AT-4, PL-4, PS-7, SA-3, SA-12, SA-16	C.F.R. Part 5 Subpart C (5C.F.R.930.301); NIST SPs 800-16, 800-50
AU-2, Audit Events	AC-6, AC-17, AU-3, AU-12, MA-4, MP-2, MP-4, SI-4	NIST SP 800-92; https://idmanagement.gov/
AU-5, Response to Audit Processing Failures	AU-4, SI-12	
AU-6, Audit Review, Analysis, and Reporting	AC-2, AC-3, AC-6, AC-17, AT-3, AU-7, AU-16, CA-7, CM-5, CM-10, CM-11, IA-3, IA-5, IR-5, IR-6, MA-4, MP-4, PE-3, PE-6, PE-14, PE-16, RA-5, SC-7, SC-18, SC-19, SI-3, SI-4, SI-7	
AU-8, Time Stamps	AU-3, AU-12	
AU-9, Protection of Audit Information	AC-3, AC-6, MP-2, MP-4, PE-2, PE-3, PE-6	
AU-12, Audit Generation	AC-3, AU-2, AU-3, AU-6, AU-7	
CA-9, Internal System Connections	AC-3, AC-4, AC-18, AC-19, AU-2, AU-12, CA- 7, CM-2, IA-3, SC-7, SI-4	
CM-2, Baseline Configuration	CM-3, CM-6, CM-8, CM-9, SA-10, PM-5, PM-7	NIST SP 800-128
CM-3, Configuration Change Control	CA-7, CM-2, CM-4, CM-5, CM-6, CM-9, SA-10, SI-2, SI-12	NIST SP 800-128
CM-4, Security Impact Analysis	CA-2, CA-7, CM-3, CM-9, SA-4, SA-5, SA-10, SI-2	NIST SP 800-128
CM-5, Access Restrictions for Change	AC-3, AC-6, PE-3	
CM-6, Configuration Settings	AC-19, CM-2, CM-3, CM-7, SI-4	OMB Memoranda 07-11, 07-18, 08-22; NIST SPs 800-70, 800-128; https://nvd.nist.gov; https://checklists.nist.gov; https://www.nsa.gov
CM-7, Least Functionality	AC-6, CM-2, RA-5, SA-5, SC-7	DoD Instruction 8551.01
CM-9, Configuration Management Plan	CM-2, CM-3, CM-4, CM-5, CM-8, SA-10	NIST SP 800-128

NIST SP 800-53 の管理策	関連する管理策	参考文献
CP-2, Contingency Plan	AC-14, CP-6, CP-7, CP-8, CP-9, CP-10, IR-4, IR-8, MP2, MP-4, MP-5, PM-8, PM-11	Federal Continuity Directive 1; NIST SP 800-34
CP-9, Information System Backup	CP-2, CP- 6, MP-4, MP-5, SC-13	NIST SP 800-34
CP-10, Information System Recovery and Reconstitution	CA-2, CA-6, CA-7, CP-2, CP-6, CP-7, CP-9, SC-24	Federal Continuity Directive 1; NIST SP 800-34
IA-2, Identification and Authentication (Organizational Users)	AC-2, AC-3, AC-14, AC-17, AC-18, IA-4, IA-5, IA-8	HSPD-12; OMB Memoranda 04-04, 06-16, 11-11; FIPS 201; NIST SPs 800-63, 800-73, 800-76, 800-78; FICAM Roadmap and Implementation Guidance; https://idmanagement.gov/
IA-4, Identifier Management	AC-2, IA-2, IA-3, IA-5, IA-8, SC-37	FIPS 201; NIST SPs 800-73, 800-76, 800-78
IA-5, Authenticator Management	AC-2, AC-3, AC-6, CM-6, IA-2, IA-4, IA-8, PL-4, PS-5, PS-6, SC-12, SC-13, SC-17, SC-28	OMB Memoranda 04-04, 11-11; FIPS 201; NIST SPs 800-63, 800-73, 800-76, 800-78; FICAM Roadmap and Implementation Guidance; https://idmanagement.gov/
IR-1, Incident Response Policy and Procedures	PM-9	NIST SPs 800-12, 800-61, 800-83, 800-100
IR-4, Incident Handling	AU-6, CM-6, CP-2, CP-4, IR-2, IR-3, IR-8, PE-6, SC-5, SC-7, SI-3, SI-4, SI-7	EO 13587; NIST SP 800-61
MA-2, Controlled Maintenance	CM-3, CM-4, MA-4, MP-6, PE-16, SA-12, SI-2	
MA-4, Nonlocal Maintenance	AC- 2, AC-3, AC-6, AC-17, AU-2, AU-3, IA-2, IA-4, IA-5, IA-8, MA-2, MA-5, MP-6, PL-2, SC-7, SC-10, SC-17	FIPS 140-2, 197, 201; NIST SPs 800-63, 800-88; CNSS Policy 15
PL-2, System Security Plan	AC-2, AC-6, AC-14, AC-17, AC-20, CA-2, CA-3, CA-7, CM-9, CP-2, IR-8, MA-4, MA-5, MP-2, MP-4, MP-5, PL-7, PM-1, PM-7, PM-8, PM-9, PM-11, SA-5, SA-17	NIST SP 800-18
PL-4, Rules of Behavior	AC-2, AC-6, AC-8, AC-9, AC-17, AC-18, AC-19, AC-20, AT-2, AT-3, CM-11, IA-2, IA-4, IA-5, MP-7, PS-6, PS-8, SA-5	NIST SP 800-18
RA-2, Security Categorization	CM-8, MP-4, RA-3, SC-7	FIPS 199; NIST SPs 800-30, 800-39, 800-60
RA-3, Risk Assessment	RA-2, PM-9	OMB Memorandum 04-04; NIST SPs 800-30, 800-39; https://idmanagement.gov/
SA-10, Developer Configuration Management	CM-3, CM-4, CM-9, SA-12, SI-2	NIST SP 800-128
SA-11, Developer Security Testing and Evaluation	CA-2, CM-4, SA-3, SA-4, SA-5, SI-2	ISO/IEC 15408; NIST SP 800-53A; https://nvd.nist.gov; http://cwe.mitre.org; http://cve.mitre.org; http://capec.mitre.org
SA-15, Development Process, Standards, and Tools	SA-3, SA-8	

NIST SP 800-53 の管理策	関連する管理策	参考文献
SA-19, Component Authenticity	PE-3, SA-12, SI-7	
SC-2, Application Partitioning	SA-4, SA-8, SC-3	
SC-4, Information in Shared Resources	AC-3, AC-4, MP-6	
SC-6, Resource Availability		
SC-8, Transmission Confidentiality and Integrity	AC-17, PE-4	FIPS 140-2, 197; NIST SPs 800-52, 800-77, 800-81, 800-113; CNSS Policy 15; NSTISSI No. 7003
SI-2, Flaw Remediation	CA-2, CA-7, CM-3, CM-5, CM-8, MA-2, IR-4, RA-5, SA-10, SA-11, SI-11	NIST SPs 800-40, 800-128
SI-4, Information System Monitoring	AC-3, AC-4, AC-8, AC-17, AU-2, AU-6, AU-7, AU-9, AU12, CA-7, IR-4, PE-3, RA-5, SC-7, SC-26, SC-35, SI-3, SI-7	NIST SPs 800-61, 800-83, 800-92, 800-137
SI-7, Software, Firmware, and Information Integrity	SA-12, SC-8, SC-13, SI-3	NIST SPs 800-147, 800-155

以下のリストは、コンテナ技術のセキュリティにとって最も重要な NIST Cybersecurity Framework [30]のサブカテゴリの詳細を示している。

- **識別：資産管理**
  - ID.AM-3: 組織内の通信とデータフロー図が、作成されている。
  - ID.AM-5: リソース（例：ハードウェア、デバイス、データ、ソフトウェア）が、それらの分類、重要度、ビジネス上の価値に基づいて優先順位付けられている。
- **識別：リスクアセスメント**
  - ID.RA-1: 資産の脆弱性が識別され、文書化されている。
  - ID.RA-3: 内部および外部からの脅威が、識別され、文書化されている。
  - ID.RA-4: ビジネスに対する潜在的な影響とその発生可能性が、識別されている。
  - ID.RA-5: 脅威、脆弱性、発生可能性、影響が、リスクを判断する際に使用されている。
  - ID.RA-6: リスク対応が、識別され、優先順位付けされている。
- **防御：アクセス制御**
  - PR.AC-1: 認可されたデバイス、ユーザのアイデンティティと証明書が管理されている。
  - PR.AC-2: 資産に対する物理アクセスが、管理され、保護されている。
  - PR.AC-3: リモートアクセスが、管理されている。
  - PR.AC-4: アクセスの許可が、最小権限の原則および役割の分離の原則を組み入れて、管理されている。
- **防御：意識向上およびトレーニング**
  - PR.AT-2: 権限を持つユーザが、自身の役割と責任を理解している。
  - PR.AT-5: 物理セキュリティおよび情報セキュリティの担当者が、自身の役割と責任を理解している。
- **防御：データセキュリティ**
  - PR.DS-2: 伝送中のデータが、保護されている。
  - PR.DS-4: 可用性を確保するのに十分な容量が、維持されている。
  - PR.DS-5: データ漏えいに対する防御対策が、実装されている。

- PR.DS-6: 完全性チェックメカニズムが、ソフトウェア、ファームウェア、および情報の完全性を検証するために使用されている。

- **防御：情報を保護するためのプロセス及び手順**
  - PR.IP-1: 情報技術/産業用制御システムのベースラインとなる構成が定められ、維持されている。
  - PR.IP-3: 構成変更管理プロセスが、策定されている。
  - PR.IP-6: データは、ポリシーに従って破壊されている。
  - PR.IP-9: （インシデント対応および事業継続）対応計画と（インシデントからの復旧および災害復旧）復旧計画が、策定され、管理されている。
  - PR.IP-12: 脆弱性管理計画が、作成され、実装されている。
- **防御：保守**
  - PR.MA-1: 組織の資産の保守と修理は、承認・管理されたツールを用いて実施され、ログが記録されている。
  - PR.MA-2: 組織の資産に対する遠隔保守は、承認を得て、ログが記録され、不正アクセスを防止した形式で実施されている。
- **防御：保護技術**
  - PR.PT-1: 監査記録/ログ記録の対象が、ポリシーに従って決定され、文書化され、実装され、その記録がレビューされている。
  - PR.PT-3: システムや資産へのアクセスが制御され、最低限の機能性の原則が組み入れられている。
- **検知：異常とイベント**
  - DE.AE-2: 検知したイベントは、攻撃の標的と手法を理解するために分析されている。
- **検知：セキュリティの継続的なモニタリング**
  - DE.CM-1: ネットワークは、サイバーセキュリティの潜在的なイベントを検知できるようにモニタリングされている。
  - DE.CM-7: 権限のない人員、接続、デバイス、ソフトウェアのモニタリングが、実施されている。
- **対応：対応計画の作成**
  - RS.RP-1: 対応計画が、インシデントの発生中または発生後に実施されている。
- **対応：分析**
  - RS.AN-1: 検知システムからの通知は、調査されている。
  - RS.AN-3: フォレンジックが、実施されている。
- **対応：低減**
  - RS.MI-1: インシデントは、封じ込められている。
  - RS.MI-2: インシデントは、緩和されている。
  - RS.MI-3: 新たに識別された脆弱性は、許容できるリスクである場合にはその旨を文書化され、そうでない場合にはリスクが緩和されている。
- **復旧：復旧計画の作成**
  - RC.RP-1: 復旧計画がインシデントの発生中または発生後に実施されている。

表 3 に、コンテナ技術を使用することで部分的または完全に実現できる NIST SP 800-53 Revision 4 [29] のセキュリティ管理策を示す。一番右の列は、各 NIST SP 800-53 管理策に対応する本文書の節を示している。

表 3：コンテナ技術でサポートされている NIST SP 800-53 の管理策

NIST SP 800-53 の管理策	コンテナ技術の関連性	本文書の関連節
CM-3, Configuration Change Control	イメージを使用して、アプリの変更管理ができる。	2.1, 2.2, 2.3, 2.4, 4.1
SC-2, Application Partitioning	ユーザ機能を管理者機能から分離するには、コンテナまたは他の仮想化技術を使用して、機能を別のコンテナで実行することで、部分的に実現できる。	2 (introduction), 2.3, 4.5.2
SC-3, Security Function Isolation	セキュリティ機能と非セキュリティ機能との分離は、機能を別のコンテナで実行されるように、コンテナや他の仮想化技術を使用することによって部分的に実現できる。	2 (introduction), 2.3, 4.5.2
SC-4, Information in Shared Resources	コンテナ技術は、各コンテナの共有リソースへのアクセスを制限するように設計されているため、あるコンテナから別のコンテナに情報が誤って漏えいすることはない。	2 (introduction), 2.2, 2.3, 4.4
SC-6, Resource Availability	各コンテナで利用可能な最大リソースを指定することができるため、どのコンテナでも過剰なリソースを消費しないようにして、リソースの可用性を保護する。	2.2, 2.3
SC-7, Boundary Protection	コンテナ間で境界を設定して適用し、コンテナ同士の通信を制限できる。	2 (introduction), 2.2, 2.3, 4.4
SC-39, Process Isolation	複数のコンテナが同じホスト上で同時にプロセスを実行できるが、それらのプロセスは互いに分離されている。	2 (introduction), 2.1, 2.2, 2.3, 4.4
SI-7, Software, Firmware, and Information Integrity	イメージのコンテンツに対する権限のない変更は容易に検知でき、変更されたイメージは既知の正常なコピーに置き換えられる。	2.3, 4.1, 4.2
SI-14, Non-Persistence	コンテナ内で実行されているイメージは、必要に応じて新しいイメージのバージョンに置き換えられるため、実行中のイメージ内に保存されているデータ、ファイル、実行可能ファイル、その他の情報は永続的ではない。	2.1, 2.3, 4.1

表 3 と同様に、表 4 は、コンテナ技術を使用することで部分的または完全に達成できる NIST Cybersecurity Framework [30]のサブカテゴリの一覧を示している。右端の列には、各 Cybersecurity Framework のサブカテゴリに対応する本文書の節を示している。

表 4：コンテナ技術がサポートする NIST サイバーセキュリティフレームワークのサブカテゴリ

サイバーセキュリティフレームワークのサブカテゴリ	コンテナ技術の関連性	本文書の関連節
PR.DS-4: Adequate capacity to ensure availability is maintained	各コンテナで利用可能な最大リソースを指定することができるため、どのコンテナでも過剰なリソースを消費しないようにして、リソースの可用性を保護する。	2.2, 2.3
PR.DS-5: Protections against data leaks are implemented	コンテナ技術は、各コンテナの共有リソースへのアクセスを制限するように設計されているため、あるコンテナから別のコンテナに情報が誤って漏えいすることはない。	2 (introduction), 2.2, 2.3, 4.4
PR.DS-6: Integrity checking mechanisms are used to verify software, firmware, and information integrity	イメージのコンテンツに対する権限のない変更は容易に検知でき、変更されたイメージは既知の正常なコピーに置き換えられる。	2.3, 4.1, 4.2
PR.DS-7: The development and testing environment(s) are separate from the production environment	コンテナを使用すると、すべての環境で同じイメージを調整なしで使用できるため、開発環境、テスト環境、本番環境を別々に持つことが容易になる。	2.1, 2.3
PR.IP-3: Configuration change control processes are in place	イメージは、アプリの変更管理に役立つ。	2.1, 2.2, 2.3, 2.4, 4.1

これらの管理策に関する情報と、可能な実装に関するガイドラインは以下の NIST の出版物で見ることができる。

- *FIPS 140-2, Security Requirements for Cryptographic Modules*
- *FIPS 197, Advanced Encryption Standard (AES)*
- *FIPS 199, Standards for Security Categorization of Federal Information and Information Systems*
- *FIPS 201-2, Personal Identity Verification (PIV) of Federal Employees and Contractors*
- *SP 800-12 Rev. 1, An Introduction to Information Security*
- *Draft SP 800-16 Rev. 1, A Role-Based Model for Federal Information Technology/Cybersecurity Training*
- *SP 800-18 Rev. 1, Guide for Developing Security Plans for Federal Information Systems*
- *SP 800-30 Rev. 1, Guide for Conducting Risk Assessments*
- *SP 800-34 Rev. 1, Contingency Planning Guide for Federal Information Systems*
- *SP 800-39, Managing Information Security Risk: Organization, Mission, and Information System View*
- *SP 800-40 Rev. 3, Guide to Enterprise Patch Management Technologies*

- *SP 800-46 Rev. 2, Guide to Enterprise Telework, Remote Access, and Bring Your Own Device (BYOD) Security*
- *SP 800-50, Building an Information Technology Security Awareness and Training Program*
- *SP 800-52 Rev. 1, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*
- *SP 800-53 Rev. 4, Security and Privacy Controls for Federal Information Systems and Organizations*
- *SP 800-53A Rev. 4, Assessing Security and Privacy Controls in Federal Information Systems and Organizations: Building Effective Assessment Plans*
- *SP 800-60 Rev. 1 Vol. 1, Guide for Mapping Types of Information and Information Systems to Security Categories*
- *SP 800-61 Rev. 2, Computer Security Incident Handling Guide*
- *SP 800-63 Rev. 3, Digital Identity Guidelines*
- *SP 800-70 Rev. 3, National Checklist Program for IT Products: Guidelines for Checklist Users and Developers*
- *SP 800-73-4, Interfaces for Personal Identity Verification*
- *SP 800-76-2, Biometric Specifications for Personal Identity Verification*
- *SP 800-77, Guide to IPsec VPNs*
- *SP 800-78-4, Cryptographic Algorithms and Key Sizes for Personal Identification Verification (PIV)*
- *SP 800-81-2, Secure Domain Name System (DNS) Deployment Guide*
- *SP 800-83 Rev. 1, Guide to Malware Incident Prevention and Handling for Desktops and Laptops*
- *SP 800-88 Rev. 1, Guidelines for Media Sanitization*
- *SP 800-92, Guide to Computer Security Log Management*
- *SP 800-100, Information Security Handbook: A Guide for Managers*
- *SP 800-113, Guide to SSL VPNs*
- *SP 800-114 Rev. 1, User's Guide to Telework and Bring Your Own Device (BYOD) Security*
- *SP 800-121 Rev. 2, Guide to Bluetooth Security*
- *SP 800-128, Guide for Security-Focused Configuration Management of Information Systems*
- *SP 800-137, Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations*
- *SP 800-147, BIOS Protection Guidelines*
- *Draft SP 800-155, BIOS Integrity Measurement Guidelines*

付録 C – 頭字語および略語

本文書で使用されている一部の頭字語と略語の定義を以下に示す。

AES	Advanced Encryption Standard
API	Application Programming Interface （アプリケーションプログラミングインタフェース）
AUFS	Advanced Multi-Layered Unification Filesystem
BIOS	Basic Input/Output System
BYOD	Bring Your Own Device
cgroup	Control Group
CIS	Center for Internet Security
CMVP	Cryptographic Module Validation Program
CVE	Common Vulnerabilities and Exposures （共通脆弱性識別子）
CVSS	Common Vulnerability Scoring System （共通脆弱性評価システム）
DevOps	Development and Operations
DNS	Domain Name System
FIPS	Federal Information Processing Standards
FIRST	Forum for Incident Response and Security Teams
FISMA	Federal Information Security Modernization Act （米国連邦情報セキュリティ近代化法）
FOIA	Freedom of Information Act （米国情報自由法）
GB	Gigabyte （ギガバイト）
I/O	Input/Output （入力/出力）
IP	Internet Protocol （インターネットプロトコル）
IPS	Intrusion Prevention System （侵入防止システム）
IT	Information Technology （情報技術）
ITL	Information Technology Laboratory
LXC	Linux Container （Linux コンテナ）
MAC	Mandatory Access Control （強制アクセス制御）
NIST	National Institute of Standards and Technology （米国標準技術研究所）
NTFS	NT File System
OMB	Office of Management and Budget （行政管理予算局）
OS	Operating System （オペレーティングシステム）
PIV	Personal Identity Verification （個人識別情報の検証）
RTM	Root of Trust for Measurement （測定の信頼性基点）

SDN	Software-Defined Networking
seccomp	Secure Computing （セキュアコンピューティング）
SIEM	Security Information and Event Management （セキュリティ情報イベント管理）
SP	Special Publication （特別出版物）
SQL	Structured Query Language
SSH	Secure Shell （セキュアシェル）
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TPM	Trusted Platform Module
URI	Uniform Resource Identifier
US	United States （アメリカ合衆国／米国）
USCIS	United States Citizenship and Immigration Services
VM	Virtual Machine（仮想マシン）
VPN	Virtual Private Network（仮想プライベートネットワーク）
WAF	Web Application Firewall （Web アプリケーションファイアウォール）

付録 D – 用語集

アプリケーションの仮想化 Application virtualization	仮想化の一形態で、単一の共有オペレーティングシステムカーネルを、複数の個別のアプリケーションインスタンスに公開し、それぞれがホスト上の他のすべてのインスタンスから分離された状態に保たれる。
ベースレイヤ Base layer	他のすべてのコンポーネントが追加される、イメージの下層レイヤ。
コンテナ Container	アプリケーション仮想化環境内で、アプリケーションをパッケージングし、セキュアに実行するための方法。アプリケーションコンテナやサーバーアプリケーションコンテナとも呼ばれる。
コンテナランタイム Container runtime	各コンテナの環境。コンテナを実行するためのリソースとリソースの使用を分離する複数のオペレーティングシステムコンポーネントを調整するバイナリで構成される。
コンテナ専用のオペレーティングシステム Container-specific operating system	コンテナのみを実行するよう明示的に設計された最小限のホストオペレーティングシステム。
ファイルシステムの仮想化 Filesystem virtualization	仮想化の一形態で、他のコンテナのストレージにアクセスしたり変更したりせずに、複数のコンテナが同じ物理ストレージを共有できる。
汎用オペレーティングシステム General-purpose operating system	コンテナ内のアプリケーションだけでなく、様々な種類のアプリケーションを実行するために使用できるホストオペレーティングシステム。
ホストオペレーティングシステム Host operating system	アプリケーション仮想化アーキテクチャ内の複数のアプリケーションによって共有されるオペレーティングシステムカーネル。
イメージ Image	コンテナの実行に必要なすべてのファイルを含むパッケージ。
分離 Isolation	ソフトウェアの複数のインスタンスを分離して、各インスタンスが自分自身だけを認識し、影響を与えることができるようにする機能。
マイクロサービス Microservice	連携してアプリケーションを構成する一連のコンテナ。
名前空間の分離 Namespace isolation	コンテナが相互に作用するリソースを制限する分離の一形態。
オペレーティングシステムの仮想化 Operating system virtualization	同じオペレーティングシステム用に作成されたアプリケーションを実行するために使用できるオペレーティングシステムの仮想的な実装。[参考文献[1]より]

オーケストレータ Orchestrator	DevOps のペルソナや、ペルソナに代わって作業する自動化を可能にするツールで、レジストリからイメージを引き出し、それらのイメージをコンテナにデプロイし、実行中のコンテナを管理する。オーケストレータは、ホスト全体のコンテナリソースの消費、ジョブの実行、マシンの状態をモニタリングする役割も担っている。
オーバーレイネットワーク Overlay network	ほとんどのオーケストレータに含まれるソフトウェアで定義されたネットワークコンポーネントで、同じ物理ネットワークを共有するアプリケーション間の通信を分離するために使用することができる。
レジストリ Registry	開発者が、作成したイメージを簡単に保存したり、タグを付けてイメージをカタログ化して識別したり、バージョン管理をして発見や再利用に役立てたり、他の人が作成したイメージを見つけてダウンロードしたりできるサービス。
リソースの割り当て Resource allocation	与えられたコンテナが消費できるホストのリソースの量をコントロールするメカニズム。
仮想マシン Virtual machine	仮想化によって作られたシミュレーション環境。[参考文献[1]より]
仮想化 Virtualization	他のソフトウェアが実行されるソフトウェアおよび/またはハードウェアのシミュレーション。[参考文献[1]より]

付録 E – 参考文献

[1] NIST Special Publication (SP) 800-125, *Guide to Security for Full Virtualization Technologies*, National Institute of Standards and Technology, Gaithersburg, Maryland, January 2011, 35pp. https://doi.org/10.6028/NIST.SP.800-125.

[2] Docker, https://www.docker.com/

[3] rkt, https://coreos.com/rkt/

[4] CoreOS Container Linux, https://coreos.com/os/docs/latest

[5] Project Atomic, http://www.projectatomic.io

[6] Google Container-Optimized OS, https://cloud.google.com/container-optimizedos/docs/

[7] Open Container Initiative Daemon (OCID), https://github.com/kubernetesincubator/cri-o

[8] Jenkins, https://jenkins.io

[9] TeamCity, https://www.jetbrains.com/teamcity/

[10] Amazon EC2 Container Registry (ECR), https://aws.amazon.com/ecr/

[11] Docker Hub, https://hub.docker.com/

[12] Docker Trusted Registry, https://hub.docker.com/r/docker/dtr/

[13] Quay Container Registry, https://quay.io

[14] Kubernetes, https://kubernetes.io/

[15] Apache Mesos, http://mesos.apache.org/

[16] Docker Swarm, https://github.com/docker/swarm

[17] NIST Special Publication (SP) 800-154, *Guide to Data-Centric System Threat Modeling (Draft)*, National Institute of Standards and Technology, Gaithersburg, Maryland, March 2016, 25pp. https://csrc.nist.gov/publications/detail/sp/800154/draft.

[18] *Common Vulnerability Scoring System v3.0: Specification Document*, Forum for Incident Response and Security Teams (FIRST). https://www.first.org/cvss/specification-document.

[19] NIST Special Publication (SP) 800-111, Guide to Storage Encryption Technologies for End User Devices, National Institute of Standards and Technology, Gaithersburg, Maryland, November 2007, 40pp. https://doi.org/10.6028/NIST.SP.800-111.

[20] CIS Docker Benchmark, Center for Internet Security (CIS). https://www.cisecurity.org/benchmark/docker/.

[21] Security Enhanced Linux (SELinux), https://selinuxproject.org/page/Main_Page

[22] AppArmor, http://wiki.apparmor.net/index.php/Main_Page

[23] NIST Special Publication (SP) 800-123, *Guide to General Server Security*, National Institute of Standards and Technology, Gaithersburg, Maryland, July 2008, 53pp. https://doi.org/10.6028/NIST.SP.800-123

[24] NIST Special Publication (SP) 800-164, *Guidelines on Hardware-Rooted Security in Mobile Devices (Draft)*, National Institute of Standards and Technology, Gaithersburg, Maryland, October 2012, 33pp. https://csrc.nist.gov/publications/detail/sp/800-164/draft.

[25] NIST Special Publication (SP) 800-147, *BIOS Protection Guidelines*, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2011, 26pp. https://doi.org/10.6028/NIST.SP.800-147.

[26] NIST Special Publication (SP) 800-155, *BIOS Integrity Measurement Guidelines (Draft)*, National Institute of Standards and Technology, Gaithersburg, Maryland, December 2011, 47pp. https://csrc.nist.gov/publications/detail/sp/800-155/draft.

[27] NIST Internal Report (IR) 7904, *Trusted Geolocation in the Cloud: Proof of Concept Implementation*, National Institute of Standards and Technology, Gaithersburg, Maryland, December 2015, 59 pp. https://doi.org/10.6028/NIST.IR.7904.

[28] NIST Special Publication (SP) 800-61 Revision 2, *Computer Security Incident Handling Guide*, National Institute of Standards and Technology, Gaithersburg, Maryland, August 2012, 79 pp. https://doi.org/10.6028/NIST.SP.800-61r2.

[29] NIST Special Publication (SP) 800-53 Revision 4, *Security and Privacy Controls for Federal Information Systems and Organizations*, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2013 (including updates as of January 15, 2014), 460pp. https://doi.org/10.6028/NIST.SP.800-53r4.

[30] *Framework for Improving Critical Infrastructure Cybersecurity Version 1.0*, National Institute of Standards and Technology, Gaithersburg, Maryland, February 12, 2014. https://www.nist.gov/document-3766.