CSAJC cloud security alliance℠ Japan Chapter

CSA cloud security alliance®

安全なアプリケーションコンテナアーキテクチャ実装のためのベストプラクティス

アプリケーションコンテナのセキュリティに関する考慮事項を信頼できるセキュアシステムのエンジニアリングに統合

ABSTRACT

アプリケーションコンテナとマイクロサービス・アーキテクチャは DevOps のようなアジャイルソフトウェア開発アプローチを活用する設計・開発・配置で利用されています。セキュリティはそれらのソフトウェア開発アプローチに組み込まれる事が必要です。この文書は開発者、運用者、アーキテクトの視点から、信頼できる安全なシステムのエンジニアリングにおけるアプリケーションコンテナをセキュアにするという課題を解決するための推奨事項とベストプラクティスを明らかにします。

ACKNOWLEDGEMENT

Application Containers and Microservices Working Group Co-chairs:

Anil Karmel Andrew
Wild

Lead Authors:

John Kinsella
Cem Gurkok
Frank Geck

Contributors:

Jeff Barnes
Randall Brooks
Ramaswamy Chandramouli
Madhav Chablani
Atul Chaturvedi
Aradhna Chetal
Joshua Cuellar
Joshua Daniel
Shyamkant Dhamke
Michele Drgon
Michaela Iorga Frank
Geck
Michael Green
Cem Gurkok
Amir Jerbi
John Kinsella
Juanita Koilpillai
Yin Lee
Aaron Lippold
Vishwas Manral
James
McCloskey Ki-
Hong Min Lloyd
Osafo Mark
Potter
Alex Rebo
Michael Roza
Ed Santiago
Kina Shah
Shankar
Ken Stavinoha
Shanthi Thomas
David Wayland
Shawn Wells
John Wrobel
Mark Yanalitis
John Osborne
Ashish Kurmi
James Yaple
Vrettos Moulos

CSA Staff:

Hillary Baron Marina
Bregu

TABLE OF CONTENTS

日本語版提供に際しての告知及び注意事項

本書「安全なアプリケーションコンテナアーキテクチャ実装のためのベストプラクティス」は、Cloud Security Alliance (CSA)が公開している「Best Practices for Implementing a Secure Application Container Architecture」の日本語訳です。本書は、CSA ジャパンが、CSA の許可を得て翻訳し、公開するものです。原文と日本語版の内容に相違があった場合には、原文が優先されます。
翻訳に際しては、原文の意味および意図するところを、極力正確に日本語で表すことを心がけていますが、翻訳の正確性および原文への忠実性について、CSA ジャパンは何らの保証をするものではありません。
この翻訳版は予告なく変更される場合があります。以下の変更履歴（日付、バージョン、変更内容）をご確認ください。

変更履歴

日付	バージョン	変更内容
2020 年 02 月 20 日	日本語版 1.0	初版発行

CSAジャパン成果物の提供に際しての制限事項

日本クラウドセキュリティアライアンス（CSA ジャパン）は、本書の提供に際し、以下のことをお断りし、またお願いします。以下の内容に同意いただけない場合、本書の閲覧および利用をお断りします。

１．　責任の限定

CSA ジャパンおよび本書の執筆・作成・講義その他による提供に関わった主体は、本書に関して、以下のことに対する責任を負いません。また、以下のことに起因するいかなる直接・間接の損害に対しても、一切の対応、是正、支払、賠償の責めを負いません。

(1) 本書の内容の真正性、正確性、無誤謬性
(2) 本書の内容が第三者の権利に抵触しもしくは権利を侵害していないこと
(3) 本書の内容に基づいて行われた判断や行為がもたらす結果
(4) 本書で引用、参照、紹介された第三者の文献等の適切性、真正性、正確性、無誤謬性および他者権利の侵害の可能性

２．　二次譲渡の制限

本書は、利用者がもっぱら自らの用のために利用するものとし、第三者へのいかなる方法による提供も、行わないものとします。他者との共有が可能な場所に本書やそのコピーを置くこと、利用者以外のものに送付・送信・提供を行うことは禁止されます。また本書を、営利・非営利を問わず、事業活動の材料または資料として、そのまま直接利用することはお断りします。

ただし、以下の場合は本項の例外とします。

(1) 本書の一部を、著作物の利用における「引用」の形で引用すること。この場合、出典を明記してください。
(2) 本書を、企業、団体その他の組織が利用する場合は、その利用に必要な範囲内で、自組織内に限定して利用すること。
(3) CSA ジャパンの書面による許可を得て、事業活動に使用すること。この許可は、文書単位で得るものとします。
(4) 転載、再掲、複製の作成と配布等について、CSA ジャパンの書面による許可・承認を得た場合。この許可・承認は、原則として文書単位で得るものとします。

３．　本書の適切な管理

(1) 本書を入手した者は、それを適切に管理し、第三者による不正アクセス、不正利用から保護するために必要かつ適切な措置を講じるものとします。
(2) 本書を入手し利用する企業、団体その他の組織は、本書の管理責任者を定め、この確認事項を順守させるものとします。また、当該責任者は、本書の電子ファイルを適切に管理し、その複製の散逸を防ぎ、指定された利用条件を遵守する（組織内の利用者に順守させることを含む）ようにしなければなりません。
(3) 本書をダウンロードした者は、CSA ジャパンからの文書（電子メールを含む）による要求があった場合には、そのダウンロードしまたは複製した本書のファイルのすべてを消去し、削除し、再生や復元ができない状態にするものとします。この要求は理由によりまたは理由なく行われることがあり、この要求を受けた者は、それを拒否できないものとします。
(4) 本書を印刷した者は、CSA ジャパンからの文書（電子メールを含む）による要求があった場合には、その印刷物のすべてについて、シュレッダーその他の方法により、再利用不可能な形で処分するものとします。

４．　原典がある場合の制限事項等

本書が Cloud Security Alliance, Inc.の著作物等の翻訳である場合には、原典に明記された制限事項、免責事項は、英語その他の言語で表記されている場合も含め、すべてここに記載の制限事項に優先して適用されます。

５．　その他

その他、本書の利用等について本書の他の場所に記載された条件、制限事項および免責事項は、すべてここに記載の制限事項と並行して順守されるべきものとします。本書およびこの制限事項に記載のないことで、本書の利用に関して疑義が生じた場合は、CSA ジャパンと利用者は誠意をもって話し合いの上、解決を図るものとします。

その他本件に関するお問合せは、info@cloudsecurityalliance.jp までお願いします。

日本語版作成に際しての謝辞

「安全なアプリケーションコンテナアーキテクチャ実装のためのベストプラクティス」の日本語訳は、CSA ジャパン会員の有志により行われました。
作業は全て、個人の無償の貢献としての私的労力提供により行われました。なお、企業会員からの参加者の貢献には、会員企業としての貢献も与っていることを付記いたします。
以下に、翻訳に参加された方々の氏名を記します。（氏名あいうえお順・敬称略）

伊賀　誠
勝見　勉
高瀬　一彰
谷　　徹也
富杉　麻衣絵
成川　達也
野原　峰彦
満田　淳
諸角　昌宏

NIST SP 800-180 で定義されている アプリケーションコンテナとマイクロサービスアーキテクチャは、DevOps のようなアジャイルソフトウェア開発アプローチを活用する設計・開発・デプロイで利用されています。アプリケーション・コンポーネントのセキュリティは、ソフトウェア開発ライフサイクル（SDLC）全体にわたって考慮されることが必要です。NIST 800-160 システムセキュリティエンジニアリングは様々なステークホルダーのニーズに基づいた信頼できるセキュアなシステムの必要性を定義しています。

この文書は、別の文書 "Challenges in Securing Application Containers and Microservices" と**対になることを意図しています**。"Challenges in Securing Application Containers and Microservices" は、開発者・運用者・アーキテクトの視点から**アプリケーションコンテナとマイクロサービスを安全にするための課題を明らかにしています。**

推奨事項とベストプラクティスは、情報セキュリティ、運用、アプリケーションコンテナ、およびマイクロサービスに関する深い知識と実践経験を持つ様々なグループ間での広範な連携を通じて作成されました。ここに含まれる推奨事項とベストプラクティスは、開発者、運用者、アーキテクトを対象としています。

1 イントロダクション

1.1 目的と範囲

この文書の目的は、アプリケーションコンテナをセキュアなアーキテクチャで実装するための推奨事項とベストプラクティスを提示することです。ガイダンスは、開発者、運用者、アーキテクトを含む様々な役割に適用されることを意図しています。信頼できる安全なシステムを開発するためにセキュリティ、開発者、運用者がともに協力することがセキュアな DevOps には必要です。

この文書の範囲は、アプリケーションコンテナに限定します。

1.2 文書構造

この後の構成は、以下の通りです。:

- 2章 はアプリケーションコンテナの概要説明です。
- 3章 はアプリケーションコンテナを安全にするための課題を解決するための推奨事項およびベストプラクティスのサマリです。
- Appendix A はこの文書で利用されている主要な略語や頭字語を記述しています。（英語）
- Appendix B はこの文書から選択された用語の索引を記載しています。（英語）
- Appendix C はこの文書で引用した参考文献のリストです。（英語）

1.3 対象読者

この文書が想定する読者は、アプリケーション開発者、アプリケーションアーキテクト、システム管理者、セキュリティ管理者、セキュリティプログラムマネージャ、情報システムセキュリティ責任者、その他のソフトウェア開発ライフサイクル（SLDC）におけるアプリケーションコンテナのセキュリティに責任を持つ、または興味を持つ方々です。

この文書は、読者がオペレーティングシステム、ネットワーキング、セキュリティなどの専門知識、アプリケーションコンテナ、マイクロサービスおよび DevOps のようなアプリケーション開発アプローチの専門知識を有していることを想定しています。アプリケーションコンテナテクノロジは日々変化するため、より詳細な最新の情報を得るために、この文書にリストされた文書を含む他のリソースを利用することを推奨します。

2 アプリケーションコンテナ

“Securing Application Containers and Microservices” （訳注：これがどの文書に当たるのかは不明）に詳述されたように、アプリーケーションコンテナとマイクロサービスはユニークな特徴を持っています。同じシステムを守るために、関係者がその役割の責任に応じたセキュリティ責任があります。そのため、アプリケーションコンテナとマイクロサービスの機能に対する課題は開発者、運用者、アーキテクトに向けて定義しています。これらのベストプラクティスに対して適用できる追加の課題は「追加の課題」カラムにリストされています。

Table 1 – Application Container Best Practices

ベストプラクティス	視点
3.1.1 **環境全体のコードプロモーション（開発、品質保証、テスト、プロダクション）**	
開発者は信頼の基点（root of trust）を確立しなければなりません。	開発者
運用者は、コンテナ管理プラットフォームを活用して、環境間で安全にイメージを移動しなければなりません。	運用者
3.1.2 **ホストをセキュアにします**	
マルチテナンシーやデータの機密性の要件は、コンテナの配置と監査の自動化を可能にするために、コンテナテクノロジー（ラベルなど）を通じて文書化しなければなりません。	運用者
コンテナランタイムは、リモートサービス（レジストリ、オーケストレーション）との間で暗号化通信をしなければなりません。	運用者
どうしても必要な場合を除き、コンテナは特権モードで実行してはいけません。	運用者
コンテナランタイムは、絶対に必要な場合にのみコンテナ内にデータボリュームをマウントすべきです（/proc、/etc、ランタイムソケットなどはマウントしないでください）。	運用者
ユーザは、プロセスにとって明示的に必要な機能以外のすべての機能を削除しなければなりません。ブラックリストのアプローチの代わりにホワイトリストを使用します。	運用者
3.1.3 **プラットフォーム/ホストからのコンテナ継続性監視**	
開発者は、明確なバージョン管理スキームを使用して、コンテナで実行されるアプリケーションのバージョンを識別する必要があります。	開発者
運用者は、コンテナランタイムはログを集中ログシステムに送信するように構成する必要があります。	運用者
運用者は、監視サービスコンテナーを、特権モードで実行させたりホストにアクセスさせてはならない。	運用者

3.1.4 コンテナネットワーク – ホストとコンテナ間の通信	
アプリケーションが安全なネットワーク通信プロトコルを使用することを推奨します。	開発者/ 運用者
運用者は、認証と認可の基盤を提供する必要があります。	運用者
運用者は、ネットワーク監視機能も提供する必要があります。	運用者
3.1.5 コンテナ・ネットワーク - コンテナ間通信	
アプリケーションは、安全なネットワーク通信プロトコルを使用する必要があります。	開発者/運用者
運用者は、認証と認可の基盤を提供する必要があります。	運用者
運用者は、ネットワーク監視機能も提供する必要があります。	運用者
運用者は、公開サービスのきめ細かなアクセス制御をサポートするコンテナ化ソリューションを使用する必要があります。	運用者
3.1.6 イメージの完全性とセキュリティレベルの検証	
開発者はイメージのビルドプロセスの一環としてイメージに署名する必要があります。またイメージの使用前にテストする必要があります。	開発者
開発者は開発プロセスの一部として脆弱性スキャンツールを使用する必要があります。	開発者/ 運用者
開発者は、イメージの中に必要なコンポーネントだけを入れるようにします。	開発者
運用者は、基盤の設計に際し、署名され承認されたイメージだけを使用許可する機能を組み込むようにすべきです。	運用者
運用者は、基盤の設計に際し、新たに公表された脆弱性を常に特定するようにすべきです。	運用者
3.1.7 コンテナのフォレンジック	
新しいアプリケーションについては、開発者は企画設計段階で、ログ機能を含める形のコーディングポリシーを作成し順守する必要があります。	開発者
既存のアプリケーションの場合、開発者は認証ログから始まるアプリケーションログをキャプチャする計画を実装するべきです。	開発者

運用者はコンテナ環境にフォレンジック機能に関する計画を組み入れる必要があります。	運用者
新しい基盤の場合、運用者は計画に基づきフォレンジック機能を実装すべきです。	運用者
既存の基盤に対しては、運用者は、まずデータ収集（ネットワークトラフィック、ディスクとメモリの 状態 ）から始めて、フォレンジック機能の実装を段階的に行うよう管理すべきです。	運用者
3.1.8 コンテナによるトラストチェーン	
開発者はイメージのビルドプロセスの一部としてイメージに署名し、イメージを使用開始前に検証すべきです。	開発者/ 運用者
開発者は開発プロセスの一環として脆弱性スキャンツールを使用すべきです。	開発者/ 運用者
開発者は、必要なコンポーネントだけがイメージに含まれるようにすべきです。	開発者
基盤の設計に際してのイメージの完全性検証、脆弱性検査、パッチ適用に関する運用者への推奨事項は3.1.6（イメージの完全性とセキュリティレベルの検証）によります。	運用者
運用者はイメージを保存しコンテナを実行する場合、セキュリティが強化されコンテナエンジンがセキュアに構成されたトラステッドホストを用いるべきです。	運用者
運用者は、認証認可メカニズ ム（つまり、TLSベースの相互認証）を使用して、コンテナ相互間および管理プラットフォームとコンテナ間のトラストを確保する必要があります。	運用者
運用者は、ロールに基づいてコンテナとイメージへのアクセスを制限すべきです。	運用者
運用者は、コンテナの構成パラメータとランタイムの完全性を監視すべきです。	運用者
3.1.9 コンテナのボリューム管理	
開発者はアプリケーション開発を安全に行うために、共有コンテナボリュームの必要性を最低限にすることや、ホストのディレクトリへのアクセスを必要としないことといった、十分なトレーニングを受けるべきです。	開発者

運用者はリソース管理、アクセス管理、コンテナのボリューム監視を支援するような基盤を提供すべきです。	運用者
運用者は、ユーザやアプリケーションごとの通信トラフィックを分離した基盤を提供すべきです。	運用者
運用者は、通信トラフィックをエンティティ(ワークロードなど)が所有するコンテナ間で通信できるようなインターフェイスを提供すべきです。	運用者
3.1.10 コンテナのシークレット管理	
開発者に対してトレーニングとベストプラクティスのガイダンスを提供します。	開発者
機密データとシークレットをバックエンドのアプリケーションコード内で取り扱うため、開発者用の共通ライブラリを作ります。	開発者
運用者は配備ツールまたはスクリプトおよびコンテナオーケストレーションツールを導入し、シークレット管理のアーキテクチャを設計すべきです。	運用者
3.1.11 プラットフォーム管理-ライフサイクルイベント通知	
コンテナのライフサイクル(start/stop/scaled)はCMPによって管理されています。開発者の観点では、コンテナ化されたアプリケーションがコンテナの移行を検知しない場合は、アプリケーションは不明な状態のままになる可能性があります。	開発者
3.1.12 プラットフォーム管理 - リソース要求	
開発者は、システムリソースのバランスをとることによって、持続可能なシステムパフォーマンスを達成できるように支援する必要があります。	開発者
マルチテナント環境では、運用者はバランスのとれたリソース消費であることを確認する必要があります。	開発者
マルチテナント環境では、運用者はCMP がクラスタリソースの消費を最適化することを確認する必要があります。	運用者
3.1.13 プラットフォーム管理 - コンテナリソース管理	
運用者は、CMP と個々のアプリケーションリソースの管理目標の間で許容できるバランスを見つけ出して達成する必要があります。	運用者

3.1.14 コンテナ管理 - コンテナリソースのスケーリング	
コンテナ配備の構成を作成する場合は、開発者または運用者は、コンテナ内のリソース使用率、優先順位、および割り当て閾値をまとめて調整するために、CMPのリソース管理機能を利用します。	開発者
運用者は開発者と協力して、コンテナ化されたアプリケーションのリソース要求を理解し、適切な制約を設定し、パフォーマンスの問題を監視する必要があります。	運用者
3.1.15 コンテナ管理 - データのバックアップとレプリケーション	
運用者は、コンテナを破棄する前に、データバックアップシステムがコンテナに含まれている全ての新しいデータをバックアップできるかどうか、確認しておく必要があります。	運用者
3.1.16 コンテナ管理 - CMP間のコンテナのホスト変更	
機密データを扱うレガシーなアプリケーションを担当する運用者は、ホストを変更するコンテナに関連する機密データのインポート、エクスポートを自動化するツールの起動、管理などの作業を行う必要があります。	運用者
3.1.17 コンテナの暗号化	
開発者と運用者は標準的な、一般的に利用されている認証システムを採用するべきです。	開発者/ 運用者
開発者は、公平な第3者によって検証されているかどうか、また意図した運用環境で実行可能かどうか考慮したうえで、保存(DAR)(暗号化)のソリューションを決める必要があります。	開発者
機密性の高いアプリケーションの場合、開発者はそのアプリケーションを暗号化し、メインのアプリケーションを復号化し実行するコンテナのエントリポイントアプリケーションを作成する必要があります。	開発者

3 アプリケーションコンテナの課題に対するリスク緩和策

3.1.1 環境全体のコードプロモーション

環境全体（開発、品質保証、テスト、本番）のコードプロモーションに固有の課題に対して提案されている緩和策は、次の推奨事項から構成されます:

1. **開発者は信頼の基点（root of trust）を確立しなければなりません。**
PKI(公開鍵暗号基盤)を使用すると、暗号化とデジタル署名用のデジタル証明書の階層を作成、管理することができます。PKIは組織内に既に存在する場合があります。この新しいシステムまたは既存のシステムの証明書を使用して、コンテナイメージに署名することで、後でその出所と完全性を検証できます。

ビルドチェーンの完全性を確保するために、開発者は電子署名する能力を持ち、環境間または次のプロモーションフェーズにプロモートされるコードとバイナリ成果物（両方またはどちらか）に電子署名を提供しなければなりません。

PKIの一部である証明書を使用してコンテナイメージに署名することにより、CMP（コンテナ管理プラットフォーム）ユーザはイメージのソースを簡単に確認できます。これは重要です。イメージのソースが決定されると、ユーザは本番用にプロモートする前にイメージが適切な精査を受けたことを確認できるためです。

2. **運用者は、コンテナ管理プラットフォームを活用して、環境間で安全にイメージを移動しなければなりません。**
実稼働コンテナ環境では、アプリケーションコードとコンテナイメージの管理に加えて、管理する多くのコンポーネントがあります。1つ以上の環境や場所にまたがる組織の規模をサポートできるCMPには、コンテナイメージを安全に同期する機能を持たなければなりません。

ビルドチェーンの完全性を確保するために、運用者は開発者やオーケストレーションエンジンによって提供されるデジタル証明書とデジタル署名（両方またはどちらか）を使用して、環境とプロモーションフェーズ間でコードとバイナリ成果物（両方またはどちらか）を検証および検証する機能を持たなければなりません。

ビルドチェーンによってデプロイされたアプリケーションの一貫性を確保するには、運用者は、アプリケーションが実行または利用されているすべての場所でアプリケーションコードを同期する機能も持たなければなりません。

CMPを使用してコンテナイメージを管理することで、プラットフォームのユーザーは、自信を持ってコンテナイメージの適切なバージョンが環境全体で実行できることを期待できます。
これにより、異なる環境で実行されているアプリケーションのバージョンに関する混乱を最小限に抑えます。プラットフォームのユーザーは、信頼の基点を使用するベストプラクティスと組み合わせると、環境全体でコンテナイメージの出所と完全性を確保できます。

3.1.2 ホストをセキュアにする

ホストをセキュアにするためのリスク緩和策には、次の運用者の視点が含まれます：

1. マルチテナンシーやデータの機密性の要件は、コンテナの配置と監査の自動化を可能にするために、コンテナテクノロジー（ラベルなど）を通じて文書化しなければなりません。
ラベルは、コンテナやコンテナの集合内に格納または処理されるデータのタイプを分類するために重要です。これらのラベルはコンテナに格納されているデータのセキュリティプロファイルに一致する環境へのコンテナの配置を自動化するためにCMPに使用されます。ラベルは、監査エンジンがコンテナ内のデータを検証できるようにする機能も果たします。

ラベルをコンテナに適用することにより、CMPはコンテナに格納されたデータを保護する適切なセキュリティを備えたホストのクラスタにコンテナを確実に配置できます。

ラベルは、監査エンジンにコンテキストを提供し、同一ホスト上でホストされているコンテナによって処理されるデータを保護するために、ホストのセキュリティ状態を確認し通知することができます。

2. コンテナランタイムは、リモートサービス（レジストリ、オーケストレーション）との間で暗号化通信をしなければなりません。
複数のホストで構成されるコンテナエコシステムでは、コンテナとリソースを管理するためにノード間の通信が必要になります。ネットワークトラフィックの暗号化により、このトラフィックの機密性が確保されます。一部のコンテナランタイムでは、デフォルトで暗号化が有効になっています。有効になっていない場合もありますが、暗号化されたトラフィックのサポートは引き続き提供されます。

アプリケーションのネットワークトラフィックのセキュリティは開発者の課題として残りますが、安全なコンテナホスティングプラットフォームを提供するには、運用者がCMP内の通信が悪意のあるエンティティにより操作されないようにしなければなりません。管理通信の暗号化は、この課題に対処します。

3. どうしても必要な場合を除き、コンテナは特権モードで実行してはいけません。
どうしても必要な場合を除き、コンテナは特権モードで実行してはいけません。
コンテナに管理者レベルの権限を付与すると、基本的にコンテナ管理者レベルのコンテナホストへのアクセス権が付与されます。これには、特権コンテナがホストおよびそのホストで実行される可能性のある他のすべてのコンテナによって完全に信頼される必要があります。この許可は、コンテナの起動時にコンテナランタイムに要求する必要があります。一部のCMPは現在、デフォルトで特権コンテナを許可していません。

このような特権の付与を禁止することにより、管理者は特権コンテナが危険にさらされるリスクと、ホストとゲストコンテナの残りの部分が危険にさらされるリスクを軽減できます。デフォルトで許可しないことで、組織は特定のコンテナを特権モードで実行するリスク/恩恵を考慮しなければならなくなります。

4. コンテナランタイムは、絶対に必要な場合にのみコンテナ内にデータボリュームをマウントすべきです。（/proc、/etc、ランタイムソケットなどはマウントしないでください）。
特権アクセスを必要とするコンテナに対する部分的なリスク緩和策として、一部のアプリケーションコンテナは、ホストシステムからコンテナにマウントされたファイルやディレクトリへの読み取りと読み取り/書き込みアクセス（両方またはどちらか）を代わりに要求するように設計されています。これは、コンテナに管理者権限を付与するよりもいくらか優れていますが、この方法には、悪意のあるユーザがホストシステムへの意図しないアクセスができるという抜け穴があることを示しています。

ホストファイルシステムとサービスのコンポーネントへのアクセスを許可しないことで、ホスト/コンテナリソースが分離され、ホストのセキュリティが確保されます。

5. ユーザは、プロセスにとって明示的に必要なCapability設定以外のすべてを削除しなければなりません。

ブラックリストのアプローチの代わりにホワイトリストを使用します。
デフォルトでは、コンテナはLinuxカーネルの限定されたCapability設定で起動されます。これらのCapability設定を使用すると、ルート権限が通常必要とされるほとんどすべての特定の領域で、プロセスをルートとして実行する必要がなくなります。コンテナはCapability設定の追加や削除によるデフォルトのオーバーライドをサポートしますが、Capability設定を追加することでコンテナの安全性を低下させる可能性があります。

通常、ルート権限が必要なLinuxカーネル機能（ルートとしてプロセスを実行する必要はありません）を制限すると、攻撃対象領域が減少し、ホストの安全が保たれます。

“legacy” host hardeningガイドでは、運用者はどのアプリケーションがそのホストで実行されているかと、強化された状態に到達するために再構成されるパラメータとを理解しています。アプリケーションコンテナーホストの場合、運用者は必ずしもそのホストで実行されるアプリケーションが事前に分からないため、強化プロセスでは、そのホストで実行できるアプリケーションコンテナと、正当に使うリソースを考慮する必要があります。

3.1.3 プラットフォーム/ホストからのコンテナ継続性監視

プラットフォームからホストまでのコンテナの継続的監視は、以下に提案する緩和策を推奨します:

1. 開発者は、明確なバージョン管理スキームを使用して、コンテナで実行されるアプリケーションのバージョンを識別する必要があります。.

受け入れられているバージョン管理スキームを定義して使用することにより、開発チームはビルドプロセスの一部としてバージョン番号を自動的に付与します。このバージョンをコンテナ名、タグ、ラベルのいずれかに適用することにより、開発者はコンテナで実行されているアプリケーションのバージョンを簡単に識別できます。

2. 運用者は、コンテナランタイムがログを集中ログシステムに送信するように構成する必要があります。

コンテナは一時的であるという性質を持つため、ログの集約の必要性が高まります;コンテナが終了（成功又は失敗のいずれか）すると、CMPはホストからコンテナのアーティファクトをすばやくクリーンアップします。コンテナの実行中と終了後の両方でログを表示するには、ログを送信し、安全で集約した場所に保存する必要があります。

コンテナからの監査ログとアプリケーションログの両方は、コンテナの実行中および終了後のトラブルシューティングとフォレンジックに有効です。これらのログは、コンテナの終了後に残る唯一の指標である可能性があるため、信頼できる必要があります。このため、ログは安全で信頼できる方法で保存する必要があります。

アプリケーションはより粒度の高いコンポーネントで構成されているため、運用者や開発者にとってトラブルシューティングがより困難になる可能性があります。また、運用者は、一貫したタイムスタンプを使用して、すべてのログを利用可能な方法で提供する必要があります。

中央ログシステムに安全なストレージ手段を提供することにより、運用者はセキュアなログストレージとアプリケーションあるいは監査ログの取得とレビューを行う手段を提供し、監視、トラブルシューティング、およびフォレンジックを可能にします。

3. 運用者は、監視サービスコンテナを、特権モードで実行させたりホストにアクセスさせてはならない。

デフォルトでは、コンテナランタイムは実行中のコンテナにホストおよびその他のコンテナへのフルアクセスを許可しません。場合によっては、ユーザまたはサードパーティは、コンテナを「特権」モードで実行し、システムリソースに簡単にアクセスできるようにすることを要求します。それによるリスクは、コンテナがリソースへの意図しないアクセス権を取得する可能性があること、またはコンテナが危険にさらされた場合、悪意のある者がホストや他のコンテナおよびリソースへのアクセスを取得することです。

たとえば、CSP上の2つのテナントは、組織内の異なる部門、あるいはパブリッククラウドの2つの異なる利用者である場合があります。どちらのシナリオにおいても、テナントが互いのデータまたはメタデータにアクセスできないことが不可欠です。このことは、テナント間でのアクセスができないように、特権コンテナがマルチテナント環境で実行できないようにするという課題を運用者に課します。
さらに、ルート権限で実行されているコンテナは、シングルテナントのシナリオでもセキュリティ上の懸念をもたらす場合もあります。

特権の付与を禁止することにより、管理者は特権コンテナがリソースまたはデータに不正にアクセスするリスクを軽減できます。デフォルトで許可しない場合、組織は特定のコンテナを特権モードで実行することのリスクと恩恵を考慮する必要があります。

3.1.4 コンテナネットワーク – ホストとコンテナ間の通信

ホストとコンテナ間の通信の課題に対する緩和策には、次の推奨事項が含まれます:

1. アプリケーションが安全なネットワーク通信プロトコルを使用することを推奨します。
コンテナによって生成されたネットワークトラフィックは、適切な暗号化プロトコル（SSL/TLS、IPSecなど）を利用して、アプリケーションと管理に関する機密性を維持する必要があります。

暗号化を実施し、機密性を維持するために安全な通信をサポートするライブラリを選択することは、情報の流出とネットワークトラフィックの改ざんを防ぎます。ネットワークトラフィックが改ざんされると、システムが侵害される可能性があります。

2. 運用者は、認証と認可の基盤を提供する必要があります。
運用者は、トラフィックを暗号化するだけでなく、アクターの身元とアクターが実行できるアクションを特定する手段を提供する必要があります。

双方向認証を行うTLSなどのソリューションを使用すると、トラフィックの機密性とアクターのIDの両方を確認します。認可メカニズムは、認証メカニズムによって提供されるIDを用いたLDAPなどのロールベースアクセス制御（RBAC）に基づいて提供されます。

3. 運用者は、ネットワーク監視機能も提供する必要があります。
ネットワークトラフィックと構成を監視すると、潜在的に悪意のあるトラフィックに加えて、ネットワークトラフィックと構成の改ざんを検出する手段が提供されます。

ネットワークトラフィック（つまり、ネットフロー）と構成の変更の記録は、特定の環境内で予期される振る舞いと活動のみが行われるようにします。これは、インシデント対応および検出機能の維持に役立ちます。

3.1.5 コンテナ・ネットワーク – コンテナ間通信

コンテナ間の通信に存在する課題に対する緩和策には、次の推奨事項が含まれます:

1. アプリケーションは、安全なネットワーク通信プロトコルを使用する必要があります。
コンテナによって生成されるネットワークトラフィックは、適切な暗号化プロトコル（SSL/TLS、IPSec）を利用して、アプリケーションと管理に関する機密性を維持する必要があります。

暗号化を実施し、機密性を維持するために安全な通信をサポートするライブラリを選択することは、情報の流出とネットワークトラフィックの改ざんを防ぎます。ネットワークトラフィックが改ざんされると、システムが侵害される可能性があります。

2. 運用者は、認証と認可の基盤を提供する必要があります。
運用者は、トラフィックを暗号化するだけでなく、アクターの身元とアクターが実行できるアクションを特定する手段を提供する必要があります。

双方向認証を行うTLSなどのソリューションを使用すると、トラフィックの機密性とアクターのIDの両方を確認します。認可メカニズムは、認証メカニズムによって提供されるIDを用いたLDAPなどのロールベースアクセス制御（RBAC）に基づいて提供されます。

3. 運用者は、ネットワーク監視機能も提供する必要があります。
ネットワークトラフィックと構成を監視すると、潜在的に悪意のあるトラフィックに加えて、ネットワークトラフィックと構成の改ざんを検出する手段が提供されます。

ネットワーク監視の課題を軽減するには、コンテナのネットワーク監視の標準的なセマンティクスとシンタックスを定義し、コンプライアンスの「必須レポート」イベントを指定し、コンテナのネットワークヘルス/ネットワークアクティビティ/イベントの正確性または関連性に対処することです。

ネットワークトラフィック（つまり、ネットフロー）と構成の変更の記録は、特定の環境内で予期される振る舞いと活動のみが行われるようにします。これは、インシデント対応および検出機能の維持に役立ちます。

4. 運用者は、公開サービスのきめ細かなアクセス制御をサポートするコンテナ化ソリューションを使用する必要があります。

運用者は、公開されたサービスのきめ細かなアクセス制御をサポートする手段を提供する必要があります。コンテナへのきめ細かなアクセス制御などのソリューションを使用すると、セキュリティリスクが軽減されます。

3.1.6 イメージの完全性とセキュリティレベルの検証

イメージの完全性とセキュリティレベルの検証における課題に対する推奨緩和策は以下のものです：

1. 開発者はイメージのビルドプロセスの一環としてイメージに署名する必要があります。またイメージの使用前にテストする必要があります。
イメージはビルドプロセスの一環として署名を行い、使用前に検証する必要があります。署名と検証は、イメージのコンテンツに実施するGNU Privacy Guard (GPG)もしくは類似の方法により実施できます。

ビルド時に行うイメージのコンテンツへのデジタル署名と使用前に署名のデータを検証することは、イメージのデータがビルドからランタイムに至る間に改ざんされていないことを保証します。

2. 開発者は開発プロセスの一部として脆弱性スキャンツールを使用する必要があります。

開発者は開発プロセスおよびCIパイプラインの一部として脆弱性スキャンツールを使用すべきで、またビルドプロセスにも脆弱性評価を組み入れるべきです。また、脆弱性評価に不合格の場合はビルドの失敗になるようにすることを考慮することが望まれます。脆弱性が検出されたら、脆弱性のあるコンポーネントにパッチを当て、イメージのビルドをやり直します。

脆弱性スキャナは、既知のセキュリティ脆弱性を含むサードパーティのコンポーネントの使用を識別し、警告します。開発サイクルの一部として脆弱性スキャンを適用すると、ソフトウェアがランタイム環境に入る前に既知の脆弱性が特定され、パッチが適用されるため、イメージのセキュリティ品質が向上します。

3. 開発者は、イメージの中に必要なコンポーネントだけを入れるようにします。
開発者は、必要なコンポーネントのみが含まれるようにイメージを絞り込む必要があります。 OSをフルに入れ込むより、ベースラインイメージと最小限のパッケージセットを使用することが望まれます。

イメージから不要なコンポーネントを削除すると、古いまたはパッチが適用されていない可能性があるパッケージの数が減るため、セキュリティの脆弱性の数を減らせます。

4. 運用者は、基盤の設計に際し、署名され承認されたイメージだけを使用許可する機能を組み込むようにすべきです。

運用者は、署名され承認されたイメージのみを受け入れる基盤を設計する必要があります。イメージの承認は、組織のソフトウェア検収ポリシー（品質、セキュリティ、安定性など）その他の検収基準に準拠します。そうすることで、運用者は、適切に署名されていても他の面で検収基準を満たさないイメージの実行を防ぐことが可能になると期待されます。

署名の検証を確認することで、ビルドからイメージ実行までの間にイメージデータが改ざんされていないことが確認できます。未承認または検収不合格のイメージ（署名されている場合でも）の使用を防止することで、実行前のイメージのさらなる検証が実現します。

5. 運用者は、基盤の設計に際し、新たに公表された脆弱性を常に特定するようにすべきです。

運用者は、以下のようなケースにおいて、イメージの脆弱性を継続的に識別する基盤を提供する必要があります：レジストリに保存されているイメージ、ホストにローカルに保存されているイメージ、コンテナとして実行されているイメージ。新しい脆弱性が発見されたら、脆弱性の深刻度とアプリケーションへの影響度に基づいて軽減対策を実行する必要があります。実行中のコンテナにパッチを適用するよりは、新しいバージョンのイメージにパッチを適用し、新しいイメージをコンテナ実行環境に投入することが望まれます。

時の経過とともに新しい脆弱性が発見され公開されるため、イメージを継続的に再スキャンする必要があります。スキャンには、イメージレジストリ内のすべてのイメージ、ローカルマシン上の（実行待ち）イメージ、およびコンテナとして現に実行中のイメージを含む必要があります。パッチ適用に際して、実行中のコンテナを更新するのではなく、新しいイメージバージョンを作成すると、プロセスの可視性が向上し、コンテナとそのイメージ間の一貫性が確保されます。

3.1.7 コンテナのフォレンジック

コンテナのフォレンジックにおける課題への対処として、推奨事項は以下のものです：

1. 新しいアプリケーションについては、開発者は企画設計段階で、ログ機能を含める形のコーディングポリシーを作成し順守する必要があります。
アプリケーションは、認証、認可、アクション、失敗に関するログを提供する必要があります。開発者は、企画設計段階の一部としてこの機能を組み入れる必要があります。

アプリケーションの認証、認可、アクション、失敗をログに記録することで、証跡が得られ、調査が入り根本原因を特定する必要がある場合に追跡が可能になります。

2. 既存のアプリケーションの場合、開発者は認証ログから始まるアプリケーションログをキャプチャする計画を実装するべきです。

既存のアプリケーションは、認証、認可、アクション、失敗に関するログを提供する必要があります。これらのログ項目のいずれかが実装されていない場合、開発者はメンテナンスフェーズの一部としてこれらの機能を提供するべきです。

アプリケーションの認証、認可、アクション、失敗をログに記録することで、証跡が得られ、調査が入り根本原因を特定する必要がある場合に追跡が可能になります。

3. 運用者はコンテナ環境にフォレンジック機能に関する計画を組み入れる必要があります。
フォレンジック機能は、インシデント対応および軽減措置の基礎をなします。**フォレンジック**は、インシデントの根本原因を特定するために必要な証拠を提供し、軽減措置を迅速化します。**フォレンジック**計画には、計装、人的資源、証拠のソース（ネットワーク、ディスク、メモリなど）、プロセス、手順を含むべきです。コンテナ環境はすぐに消滅する性質を持つので、証拠をキャプチャして分析するフレームワークは、より迅速である必要があります。

フォレンジック機能をインシデント対応計画と手順に統合すると、証拠を取得して処理する手段を提供し、根本原因を特定する時間を短縮し、侵害にさらされる度合いを最小限に抑えることができます。

4. 新しい基盤の場合、運用者は計画に基づきフォレンジック機能を実装すべきです。
計画に基づくフォレンジック機能の設計と実装は、その他のインフラと共に行い、企業のインシデントレスポンス、事業継続および災害復旧計画に組み込まれる必要があります。

フォレンジック機能の設計と実装は、変化への対応が必要な場合に、計画を見直し設計を修正するのに有効です。こうした機能は、証拠の収集と処理の手段となり、根本原因を特定する時間を短縮し、侵害の危険を最小限に抑制します。

5. 既存の基盤に対しては、運用者は、まずデータ収集（ネットワークトラフィック、ディスクとメモリの状態 ）から始めて、フォレンジック機能の実装を段階的に行うよう管理すべきです。

フォレンジック機能を既存の基盤に組み入れることは、運用者の保守および変更管理業務の一環であるべきです。このことは、デューデリジェンスとコンプライアンスの一環であり、また規則と規制の順守を実現する意味で実施されるべきです。

フォレンジック機能の実装により、運用者はデューデリジェンス要件、コンプライアンス対策、規則、規制を満たすことができます。こうした機能は、証拠の収集と処理の手段となり、根本原因を特定する時間を短縮し、侵害の危険を最小限に抑制します。

3.1.8 コンテナによるトラストチェーン

コンテナによりトラストチェーンを実現するために、以下の対策を推奨します:

1. 開発者はイメージのビルドプロセスの一部としてイメージに署名し、イメージを使用開始前に検証すべきです。
イメージは、ビルドプロセスの一部として署名し、使用する前に検証する必要があります。署名と検証はイメージの内容にGPG署名を施すか、類似の方法で実現できます。開発者は、ビルドプロセスの一環としてイメージに署名されていることを確実にしなければなりません。運用者は、配備する前にイメージの署名を検証する機能が利用できるようにしておく必要があります。

ビルド時のイメージコンテンツのデジタル署名と使用前の署名済みデータの検証により、ビルドとランタイムの間でイメージデータの改ざんがないことが確認できます。

2. 開発者は開発プロセスの一環として脆弱性スキャンツールを使用すべきです。

開発者は、開発プロセスおよびCIパイプラインの一環として脆弱性スキャンツールを使用し、脆弱性評価をビルドプロセスに取り入れる必要があります。また、脆弱性評価で問題が生じたら、ビルドを取りやめることを考えるべきです。 脆弱性が発見されたら、脆弱なコンポーネントにセキュリティパッチを適用し、イメージを再構築すべきです。

脆弱性スキャナは、既知のセキュリティ脆弱性を持つサードパーティコンポーネントの使用を検出し、警告を出します。開発サイクルの一部として脆弱性スキャンを適用することで、ソフトウェアがランタイム環境に入る前に既知の脆弱性が検出され、パッチが適用されるため、イメージのセキュリティ品質を向上させます。

3. 開発者は、必要なコンポーネントだけがイメージに含まれるようにすべきです。

開発者は、必要なコンポーネントのみが含まれるようにイメージを整理するべきです。ベースラインイメージを使用してパッケージのセットは最小限にすることを推奨します。

イメージから不要なコンポーネントを削除すると、古いまたはパッチが適用されていない可能性があるパッケージを減らすことができ、セキュリティ上の脆弱性を減少させます。

4. 基盤の設計に際してのイメージの完全性検証、脆弱性検査、パッチ適用に関する運用者への推奨事項は3.1.6（イメージの完全性とセキュリティレベルの検証）によります。

5. 運用者はイメージを保存しコンテナを実行する場合、セキュリティが強化されコンテナエンジンがセキュアに構成されたトラステッドホストを用いるべきです。
運用者は、OSとコンテナエンジンを強化した上でイメージを保存してコンテナを実行すべきです。ホストには必要なパッケージのみを配備し、セキュリティアップデートとカーネルのパッチを常に適用することが重要です。運用者は、ベンダーのセキュリティのベストプラクティスに従ってコンテナエンジンを強化する必要があります。

ホストOSとコンテナエンジンに適切にパッチを適用し、安全に構成することにより、コンテナの攻撃対象面と既知の脆弱性にさらされるリスクを減少させます。

6. 運用者は、認証認可メカニズム（つまり、TLSベースの相互認証）を使用して、コンテナ相互間および管理プラットフォームとコンテナ間のトラストを確保する必要があります。
エンドポイント間の相互認証（相互TLS認証など）を使用して、コンテナ管理プラットフォームとコンテナのエンドポイントを構成します。相互認証がサポートされていない場合は、コンテナのアンバサダーパターン（訳注：Ambassadorはマイクロサービス間のAPI Gatewayとして機能し、認証やルーティング、タップなどを提供します）を使用してセキュリティ保護されたプロキシ通信を実現することを検討します。

データの改ざん、中間者攻撃、および機微情報の漏えいを防ぐために、すべての通信に相互認証を適用することを推奨します。

7. 運用者は、ロールに基づいてコンテナとイメージへのアクセスを制限すべきです。
運用者は、ユーザアクセス制御メカニズムを適用して、認証されたユーザのみがコンテナとイメージにアクセスできるようにする必要があります。ユーザアクセス制御メカニズムを適用すると、権限のないユーザがイメージやコンテナにアクセスすることを防止できます。

8. 運用者は、コンテナの構成パラメータとランタイムの完全性を監視すべきです。
運用者は、コンテナの構成をモニターし、セキュアでない設定を検出し、実行中のコンテナの構成ファイルとプロセスの完全性を確認する必要があります。

安全でない設定によるコンテナの実行は、ホストおよびネットワークリソースへの無制限のアクセスをもたらします。コンテナプロセスと構成ファイルの変更は、コンテナが改ざんされていることを示している場合があります。

3.1.9 コンテナのボリューム管理

コンテナのボリューム管理における課題に対する緩和策には、次の推奨事項が含まれます:

1. 開発者はアプリケーション開発を安全に行うために、共有コンテナボリュームの必要性を最低限にすることや、ホストのディレクトリへのアクセスを必要としないことといった、十分なトレーニングを受けるべきです。

2. 運用者はリソース管理、アクセス管理、コンテナのボリューム監視を支援するような基盤を提供すべきです。
ボリュームリソースの管理は、以下のボリューム管理サービスのアクティビティを含みます:

- ボリュームリソースを作成、プロビジョニングおよび更新
- ボリュームリソースへの登録、アクセスおよびアクセス拒否
- ボリュームリソースの利用停止や終了
- ボリュームリソースの問題解決
- ボリュームリソースの監視、レポート、監査、インベントリ、検証、および確認
- ボリュームリソースの移動
- ボリュームリソースの暗号化
- ボリュームリソースの復旧
- ボリュームリソースのパフォーマンスチューニング

アセット（人、グループ、コンテナ、ネットワーク、ホスト、ボリュームなど）間の関係性は明確に認識され、ボリューム管理サービスのアクティビティと紐づけられます:

- データボリュームに紐づけられるコンテナのアクセス
  - アセットコンテナは
    - データボリュームへの認可されたアクセス
    - ボリュームが同じVMホスト上に存在する場合の認可されたフルアクセス

- データおよびオペレーションボリュームに紐づけられるロールのアクセス
  - ボリュームロール
    - データボリュームは常に専用のストレージに紐づけられます
    - ランタイムオペレーションボリューム
  - グループロール
    - 開発者ロールは
      * データボリュームに対してアクセスが許可されます
      * オペレーションボリュームに対してアクセスが許可されます
    - 運用者ロールは
      * データボリュームに対してアクセスが許可されます
      * オペレーションボリュームに対してアクセスが許可されます
    - ユーザロールは、
      * データボリュームに対してアクセスが許可されます
      * オペレーションボリュームに対してアクセスが許可されます

「リソース管理」の細分化は、ボリュームリソース管理のアクティビティにおいて継続的な管理のための潜在的な実施基準と認識されています。

3. 運用者は、ユーザやアプリケーションごとの通信トラフィックを分離した基盤を提供すべきです。

ユーザやアプリケーションごとの通信トラフィックは、潜在的に侵害されたユーザやアプリケーションからのlateral movementを制限するために分離される必要があります。このネットワーク分離はlateral movementの脅威ベクターを制限します。

4. 運用者は、通信トラフィックをエンティティ(ワークロードなど)が所有するコンテナ間で通信できるようなインターフェイスを提供すべきです。

コンテナがエンティティ(ワークロードなど)で構成されて効果的に相互通信できることを確実にするインターフェイスが求められています。このインターフェイスは、改ざんされていないことを確認するために、外部のエンティティにより監視されます。

監視プラットフォームを伴ったインターフェイスは、エンティティ(ワークロードなど)が所有するコンテナが通信可能かつ、ユーザやアプリケーションごとに分離されていることを確実にします。

3.1.10 コンテナのシークレット管理

コンテナのシークレット管理では、課題に対して提案または採用された次の軽減策を推奨します。

1. 開発者に対してトレーニングとベストプラクティスのガイダンスを提供します。

バックエンドの開発者は、パブリッククラウドがマルチテナントであることにより引き起こされる脅威について、周知され訓練されるべきです。開発者がアプリケーションのコード内に静的なシークレットをハードコードしてしまう習慣を避けるため、アプリケーション内でシークレット管理機能をどのように使うのかといったテンプレートを教えられたり提供されたりする必要もあります。

バックエンドの開発者はサーバサイドが信頼できる環境だと思ったその日から、サーバのコード内に機密情報を書いてしまう習慣があります。現在のパブリッククラウド環境の微妙な差異や脅威への意識を向上させることは、そのような習慣を避けることに役立ちます。

2. 機密データとシークレットをバックエンドのアプリケーションコード内で取り扱うため、開発者用の共通ライブラリを作ります。

バックエンドのアプリケーションコード内での機密データとシークレットの取り扱いを処理する確実な方法を実現するために、共通のライブラリのセットが開発者に提供されるべきです。

アプリケーションの開発者はセキュリティの開発者である必要はなくて、セキュリティではなくアプリケーション機能を開発することに時間を捧げるべきです。アプリケーションの中で要求されるセキュリティは、使いやすくて明快であるべきです。それは、最小限の努力でアプリケーション間の一貫性を達成する必要があります。

3. 運用者は配備ツールまたはスクリプトおよびコンテナオーケストレーションツールを導入し、シークレット管理のアーキテクチャを設計すべきです。

人の手によって作られた配備シナリオに対応した配備スクリプトと同様に、配備時間とユーザ入力を利用したシークレット管理の配備と最初のシーディングは、特定のコンテナオーケストレーションスクリプト

を密に統合することに利用できます。完全に自動化された配備の場合、シークレット管理サーバが初期ブートのシークレットを要求するために、キーマネジメントサーバ(KMS)またはハードウェアセキュアモジュール(HSM)を利用すべきです。例えば、特定のロールに紐づけられた仮想マシン(VM)のメタデータは、VMの起動中にHSMからシークレットを取得するため、HSMの認証に利用することができます。

シークレットのブートはいつもニワトリが先か卵が先かの問題で、最初のシークレットをHSMやKMSから取得するために、VMのメタデータなどいくつかの自動的に作成されるデータを利用することは、この問題を解決する１つの方法になります。

一度シークレット管理システムが配備時または起動時のシークレットでブートされると、シークレットの動的生成を行うことが可能になります。例えば、アプリケーションにデータベースアカウントのクレデンシャルを動的に生成するため、配備時に取得されたデータベースのadminクレデンシャルを使って新規ユーザアカウント作成を許可しそれを利用するアプリケーションに受け渡す必要があります。

3.1.11 プラットフォーム管理 – ライフサイクルイベント通知

ライフサイクルイベント通知に適用されるプラットフォーム管理の課題に対して提案された緩和策には、次の推奨事項が含まれます。

1.コンテナのライフサイクル(start/stop/scaled)はCMPによって管理されています。開発者の観点では、コンテナ化されたアプリケーションがコンテナの移行を検知しない場合は、アプリケーションは不明な状態のままになる可能性があります。

CMPはカプセル化されたアプリケーションが既知の安全な状態に適切に移行できるようにすべきです。従来の解決策はアプリケーションにコンテナのライフサイクルイベントが通知されることを許可するものでした。リソースはコンテナの削除後に解放され、既知の状態でプールに戻されて、コンテナは状態変化のイベントをログ出力できるべきです。

コンテナのライフサイクルイベントを通知するのは不可欠で、これにより安全な起動と停止を確実にする適切なアクションをとることができます。

3.1.12 プラットフォーム管理 - リソース要求

課題に対する以下の緩和策を推奨します。

1.開発者は、システムリソースのバランスをとることによって、持続可能なシステムパフォーマンスを達成できるように支援する必要があります。

開発者は、運用者やアーキテクトと協力し、バイナリとライブラリがコンテナ化されたインフラストラクチャでの動作が最適化されていることを確認する必要があります。

2.マルチテナント環境では、運用者はバランスのとれたリソース消費であることを確認する必要があります。サービスの品質は常にマルチテナンシーを犠牲にして維持される必要があります。これにより、少なくとも1つのサービスレベル契約（SLA）を履行できます。開発者やベンダーと協力してリソースリクエストを調整することは有益です。

3. マルチテナント環境では、運用者はCMP がクラスタリソースの消費を最適化することを確認する必要があります。
リソース消費のバランスをとることにより、持続可能な実行が容易になり、リソースの競合が最小限に抑えられます。

3.1.13 プラットフォーム管理 – コンテナリソース管理

課題に対する以下の緩和策を推奨します。

1. 運用者は、CMP と個々のアプリケーションリソースの管理目標の間で許容できるバランスを見つけ出して達成する必要があります。

CMPは、アプリケーションが策定するリソース管理ポリシーを保持する必要があります。この確認は、拘束力のあるSLA になります。

CMP は、設定された時間間隔で所定のリソースの割り振りを試みる必要があります。次に、アプリケーションは、その時間間隔中にこれらのリソースを事前に定義された方法で使用します。もし違反（無条件のアプリケーション終了を含む)が起これば、当事者が一連の所定の行動に取り組むことができる条項があります。

3.1.14 コンテナ管理 – コンテナリソースのスケーリング

コンテナ管理における、コンテナリソースのスケーリングの課題に対して以下の緩和策を推奨します。

1. コンテナ配備の構成を作成する場合は、開発者または運用者は、コンテナ内のリソース使用率、優先順位、および割り当て閾値をまとめて調整するために、CMPのリソース管理機能を利用します。

アプリケーション開発者は、開発中のアプリケーションが、本番環境で使用可能なコンピューティングリソース全体を用意できないことを受け入れる必要があります。自動スケールが存在する場合でも、その機能には限界があり、無限にスケールアウトしません。構成管理チームによって確立されたパラメータ内でアプリケーションが機能できることを保証することによって、開発者は、水平インフラストラクチャスケーリング機能の存在を想定し、垂直スケーリングを要求することを避けるべきです。

開発者は、リソースの使用とセキュリティ強化のためのパラメータを取得し、それらの構成設定をソフトウェア開発環境とテストスクリプトに組み込む必要があります。定義済みの本番環境パラメータでアプリケーション機能をテストしないと、アプリケーションのパフォーマンスが低下するリスクが下がるのではなく上がります。
使用中のCMPに対する適切なリソースとセキュリティ機能の適切な構成は、リソース消費による障害または暴走したコンテナプロセスが隣接するコンテナに影響を与えないことを保証します。

2. 運用者は開発者と協力して、コンテナ化されたアプリケーションのリソース要求を理解し、適切な制約を設定し、パフォーマンスの問題を監視する必要があります。

開発者と運用者は、協力することでアプリケーションのリソース要件を共通理解できます。リソース消費は実行時のイベントなので、コンテナが暴走し、環境内ですべてのCPU、メモリ、および IO リソースが消費されるのを防ぐために管理策をコンテナを起動する前に実施します。

運用者は、アプリケーションの現在のリソース使用状況、セキュリティパラメータ、必要なログについて開発者と情報交換する必要があります。それにより、開発中のコンテナ化されたアプリケーションが、本番環境で必要なリソースに基づいてコンテナに設定された制限内で動作できるように、開発者は、情報に基づいたテストとQAの決定を行います。

3.1.15 コンテナ管理 – データのバックアップとレプリケーション

コンテナ管理において、データのバックアップとレプリケーションを行う場合、以下の緩和策を推奨します。

1. 運用者は、コンテナを破棄する前に、データバックアップシステムがコンテナに含まれている全ての新しいデータをバックアップできるかどうか、確認しておく必要があります。

新しくコンテナ化されたアプリケーションの場合、永続的なデータをコンテナ内に保存してはなりません。アプリケーションコンテナ内でレガシーなアプリケーションが実行されている可能性がある場合、運用者は開発者と協力してデータの保存場所を特定し、バックアップが定期的に実行され、かつ、コンテナが破棄される前にバックアップされることを確認してください。一部のCMPにはコンテナの停止または破棄の前に通知を行う機能が搭載されています。この機能が使えるときは、バックアップがないままデータが破壊されないようにすることに利用すべきです。

開発者の視点：

1. **アプリケーションのバックアップおよびリストアに必要な要件を適切に文書化する必要があります。**また、この文書にはアプリケーションをバックアップまたはリストアするための状態を定義し記載しておかなければなりません。一部のアプリケーションでは、ストレージのスナップショットだけで十分な場合もありますが、別のアプリケーションでは、独自のツールを実行しなければならない場合もあります。アプリケーションをオフラインにしなければならない、あるいは、アクティブユーザ/セッションの数を減らさなければならない場合には、その要件を文書化する必要があります。

2. **バックアップまたはリストアの際にアプリケーションをオフラインにする場合、アプリケーションの上流または下流で影響を受けるサービスについてもドキュメントに記載しておくことが重要です。**要件を文書化することで、 運用者はブルー・グリーン・デプロイメントのような手法を用いて、一時的に別のコンテナを実行させ、連鎖的な障害を回避することができるでしょう。

3. **開発者がアプリケーションの上流または下流の依存関係の影響を考慮しなければならない場合、クラウドネイティブのベストプラクティスを用いることを強く推奨します。**例えば、アプリケーションへの依存性と冪等性（訳注： ある操作を1回行っても複数回行っても結果が同じであることをいう概念）を組み込んだリカバリを設計し、サービスへ実装しておくことが重要となります。このようなベストプラクティスを用いることで、バックアップまたはリストア中にアプリケーションが使用できない場合には、上流および下流の依存関係を回復できるでしょうし、システム障害を回避することもできるでしょう。

4. **開発者は、単一サービスに責任を持つ、インターフェースを分離する、サービスの依存を回避するといった、マイクロサービスにおける主な慣行に従う必要があります。**イミュータブル・インフラストラクチャを用いる場合、開発者はゲートウェイサービスまたはservice helper patternを用いて、コンテナ内で発生するコンピュータプロセスをバックエンドのプロセスへオフロードする必要があります。開発者はWebアプリケーションアーカイブ（WAR パッケージファイル）に依存しないようにしてください。また、ファイルシステムに配置されたJARファイルに依存する、クライアントランタイム環境を増強する、形成することがないようにしてください。

開発者は、可変レイテンシーに対応し、ワークロードが実行される場所が不明な動的なデプロイメントパターンにも対応したアプリケーションを構築する必要があります。

5. 開発者は**処理が実行されるファイルシステムに依存せず、データとメタデータを独立させて管理できるソフトウェア機能を提供しなければなりません。また、開発者はコンテナの実行を“プロセスのユニット”として扱う必要があります。これにより、使用されるデータはコンテナの外部に存在しますが、論理ボリュームを介してアクセスができます。**
開発者のコードリポジトリをリストアポイントとして使用する考えは推奨されません。適切に構成し実行されるバックアップAPIは、データやメタデータをリストアするだけでなく、コンピュータスタックから独立して実行され、指定した時点へのリストアが可能です。

運用者の視点：

1. 運用者は、開発者と組織のポリシーによって定められた基準を参照して、アプリケーションを適切にバックアップまたはシステム上へリストアする方法を実装しなければなりません。これらのポリシーには、バックアップの回数や頻度などが要件として含まれる場合があります。
2. **ストレージのスナップショットによる方法で十分に対応可能な単純なユースケースの場合、運用者はアプリケーションと組織の要件に従って、バックアップを定期的に実施する必要があります。**
3. 一部の環境には、ボリュームを作成した際に、ストレージデバイスへ動的にマッピングする機能を有する場合があります。これはダイナミックプロビジョニングと呼ばれる機能です。運用者がスナップショットポリシーを使用している場合、ストレージメディアではボリュームが作成されるまでスナップショットのスケジュールが許可されないため、ダイナミックプロビジョニングを利用できないことがあります。これが正しく実行されない場合、運用者はこのダイナミックプロビジョニングを無効にしなければなりません。
4. 単純ではない構成の場合、 運用者はアプリケーションがバックアップできる安全な状態を用意し、アプリケーションのバックアップに必要なアプリケーション固有のツールを実行する責任を負います。これはアプリケーションがオフラインになったときに、アプリケーションの依存関係に関連したワークロードを処理するために一時的に別のインスタンスを立ち上げることも含みます。これは簡単ではないかもしれません。例えば、運用者には以下の事項が求められます：
   a. ブルー・グリーン・デプロイメントを展開することで、アプリケーションがオフラインの間にトラフィックをオフロードします。
   b. 開発者またはアプリケーションベンダー、あるいはそれら双方によって示されているように、バックアップまたはリストアに適した安全な状態を作成します。これには途中となっているセッションや終了したトランザクションが含まれる場合があります。コンテナのシナリオに実装する場合、サイドカーパターン（データベースの隣りに配置されるコンテナを走らせるエージェント）を使用するか、他のcron ツールを使用する必要があります。
   c. アプリケーションをバックアップするためにアプリケーション固有のツールを実行しなければなりません。Webサーバの場合、すべてのイベントをセカンダリクラスタへ複製する必要があるかもしれません。データベースの場合、特定のツールを走らせることが含まれるかもしれません。ツールがデータのアーカイブを作成する場合、データを安全にストレージメディアへバックアップするようにしてください。データアーカイブを安全な場所へ移動するにあたって、カスタムスクリプトを用いることもあります。

3.1.16 コンテナ管理 - CMP間のコンテナのホスト変更

コンテナ管理に関する課題に対して、いくつかの緩和策があります。CMP間でコンテナがホストを変更する際には、以下の推奨事項を参照してください：

1. **機密データを扱うレガシーなアプリケーションを担当する運用者は、ホストを変更するコンテナに関連する機密データのインポート、エクスポートを自動化するツールの起動、管理などの作業を行う必要があります。**

いまのところ、CMP間のプラットフォーム情報や設定情報をエクスポートまたはインポートするような、標準的な方法は存在していません。このアクティビティはCMPレベルまたはクラウドブローカーの権限で実行することができます。したがって、運用者は、ホスト変更するコンテナやそれに関連するセキュリティコンポーネントを移動する機能を持つユーティリティを利用する必要があります。

この緩和策は、ターゲットとなるCMP（訳注：原文はCSPだが文脈からCMPと理解）でセキュリティ構成が取られていないコンテナを作成し、コンテナ自体を移行し、ソースCMP（訳注：原文はCSPだが文脈からCMPと理解）でコンテナとそのセキュリティ構成を削除する手動プロセスに依存しています。この手動操作による手順は操作ミスが発生するリスクがあるため、このプロセスを自動化できるユーティリティのコレクションを持つことは、時間をかける価値があります。

3.1.17 コンテナの暗号化

コンテナの暗号化に関する緩和策については、以下の推奨事項を参照してください:

1. 開発者と運用者は標準的な、一般的に利用されている認証システムを採用するべきです。

共通の認証システムを使用することで、CMP間で同じ暗号鍵を用いることに対する認証と認可が行われると想定できるようになります。コンテナ内部でデータまたはアプリケーションのいずれかが暗号化される場合、暗号鍵の使用を許可する信頼できる仕組みが必要です。さもないと、アプリケーションは複数の認証方法をサポートするように書かれなければならず、無用な複雑さが生じます。

2. 開発者は、公平な第3者によって検証されているかどうか、また意図した運用環境で実行可能かどうか考慮したうえで、保存(DAR)(暗号化)のソリューションを決める必要があります。

もし可能であれば、開発者はFIPS 140-2検証済のDAR(暗号化)のソリューションを利用する必要があります。また、アプリケーション プログラミング インターフェース(API)と鍵管理システム(KMS)を調査することも必要です。FIPS検証済みソリューションおよび互換性あるKMSについては、それらのソリューションのセキュリティについて第三者評価が行われ、高いレベルの保証が得られます。

開発者は、最新のAES(advanced encryption standard)を可能な限り長い鍵長で用いるようにしてください。これによりデータの機密性が確保されます。

3. 機密性の高いアプリケーションの場合、開発者はそのアプリケーションを暗号化し、メインのアプリケーションを復号化し実行するコンテナのエントリポイントアプリケーションを作成する必要があります。

機密性の高いアプリケーションを暗号化することで、意図しない漏えいのリスクを緩和することができます。これによりコンテナイメージ(および機密性の高いアプリケーション)の保存、送信、共有が意図しない漏えいにさらされないようにできます。アプリケーションの暗号化と復号化は、他のデータの暗号化および復号化と同様のプロセスとして扱う必要があります。

Appendix A–Acronyms

Selected acronyms and abbreviations used in this paper are defined below.

ACM	Application Container and Microservices
CMP	Container Management Platform
CSP	Cloud Service Provider
HSM	Hardware Secure Module
IAM	Identity and Access Management
KMS	Key Management System
OS	Operating System
RBAC	Role-Based Access Control
SLA	Service-Level Agreement
SSL	Secure Sockets Layer
VM	Virtual Machine

Appendix B –Glossary	
Application container **[NIST SP800-180]**	An application container is a construct designed to package and run an application or its components running on a shared operating system. Application containers are isolated from other application containers and share the resources of the underlying operating system, allowing **for efficient restart, scale-up, or scale-out of applications across clouds.** Application containers typically contain microservices
Container management platform	A container management platform is an application designed to manage containers and their various operations, including but not limited to **deployment, configuration, scheduling, and destruction.**
Container lifecycle events	The main events in the life cycle of a container are create container, run container, pause container, unpause container, start container, stop container, restart container, kill container, destroy container.
Developer **[NIST SP 800-64 Rev 2/ ISO/IEC 17789:2014]**	The cloud service Developer is a sub-role of cloud service partner which is responsible for designing, developing, testing, and maintaining the implementation of a cloud service. This can involve composing the service implementation from existing service implementations. The cloud service Developer's cloud computing activities include: • **design, create and maintain service components (clause 8.4.2.1);** • **compose services (clause 8.4.2.2);** • **test services (clause 8.4.2.3).** **NOTE 1 – Cloud service integrator and cloud service component** Developer describe sub-roles of cloud service Developer, where the cloud service integrator deals with the composition of a service from other services and where cloud service component Developer deals with the design, creation, testing, and maintenance of individual service components. **NOTE 2 – This includes service implementations and service** components that involve interactions with peer cloud service providers.
Host	OS supporting the container environment
Lateral movement **[LIGHTCYBER 2016]**	Lateral action from a compromised internal host to strengthen the attacker foothold inside the organizational network, to control additional machines, and to eventually control strategic assets.

Microservices **[NIST SP800-180]**	A microservice is a basic element that results from the architectural decomposition of an application's components into loosely coupled patterns consisting of self-contained services that communicate with each other using a standard communications protocol and a set of **well-defined APIs, independent of any vendor, product, or technology.** Microservices are built around capabilities as opposed to services, build on SOA, and are implemented using Agile techniques. Microservices are typically deployed inside application containers.
Enterprise Operator	The individual or organization responsible for the set of processes to deploy and manage IT services. They ensure the smooth functioning of the infrastructure and operational environments that support application deployment to internal and external customers, including the network infrastructure, server and device management, computer operations, IT infrastructure library (ITIL) management, and help desk services. [1]
Enterprise Architect	The individual or organization responsible for strategic design recommendations. They determine, by applying their knowledge of cloud, container and microservices components to the problems of the business; the best architecture to meet the strategic needs of the business. Additionally, they develop and maintain solution roadmaps and oversee their adoption working with Developers and Operators to **ensure an efficient and effective solution implementation.**
Container resources	Four resources required for containers to operate are CPU, memory **(+swap), disk (space + speed), and Network.**
Container resource requests	The amount of CPU, memory (+swap), and disk (space + speed) that the system will allocate to the container considering the resource limit.
Container resource limit	The maximum amount of resources (CPU, memory (+swap), and disk (space + speed)) that the system will allow a container to use.

1 Wikipedia. Information technology operations. Retrieved from https://en.wikipedia.org/wiki/Information_technology_operations.

Appendix C–References

[SP 800-180]	**NIST Special Publication (SP) 800-180 (Draft),** *NIST Definition of Microservices, Application Containers and System Virtual Machines*, **National Institute of Standards and Technology, Gaithersburg, Maryland, February 2016, 12pp.** http://csrc.nist.gov/publications/drafts/800-180/sp800-180_draft.pdf
[SP 800-160]	**NIST Special Publication (SP) 800-160,** *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems,* **National Institute of Standards and Technology, Gaithersburg, Maryland, November 2016, 257pp.** http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf
[SP 800-123]	**NIST Special Publication (SP) 800-123,** *Guide to General Server Security*, **National Institute of Standards and Technology, Gaithersburg, Maryland, July 2008, 53pp.** https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf
[NIST SP 800-64 Rev 2]	**NIST Special Publication (SP) 800-64 Rev 2,** *Security Considerations in the System Development Life Cycle*, **National Institute of Standards and Technology, Gaithersburg, Maryland, October 2008, 67pp.** http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf
[ISO/IEC 17789:2014]	International Organization for Standardization/International Electrotechnical Commission, *Information Technology – Cloud Computing – Reference Architecture*, **ISO/IEC 17789:2014, 2014.** https://www.iso.org/standard/60545.html **[accessed 5/11/17].**
[LIGHTCYBER 2016]	*Cyber Weapons Report 2016,* **LightCyber, Ramat Gan, Israel, 2016, 14pp.** http://lightcyber.com/cyber-weapons-report-network-traffic-analytics-reveals-attacker-tools/ **[accessed 5/11/17].**

