1L-06

クラウド基盤におけるファイル権限による ロール管理可能なコンテナエンジンの提案

飯國 隆志 [†1]　富永 浩之 [†2]
香川大学 [†1]　香川大学 [†2]

1. はじめに

クラウドコンピューティングの発達により，様々なサービスモデルが提案されている．その中には，FaaS(Function as a Service) や PaaS(Platform as a Service) のような実行環境を提供する形態がある．これらのサービスの実行環境としては，Docker[1] などのコンテナ型仮想環境が代表的である．ハイパーバイザ型の仮想環境と比べ，コンテナ型仮想環境は，比較的起動が高速である．そのため，イベント駆動で関数やコードを実行する，サーバレスコンピューティング [2] におけるサーバレスアーキテクチャに適している．しかし，コンテナ型仮想環境は，ホストとカーネルを共有することや，運用に特権をいくつも必要とする．そのため，いくつもセキュリティ的課題を抱えている [3]．本研究では，ホストに対して権限昇格が行われないようにすることや，他のコンテナへ意図しない影響を与えないことを目的とする．本論では，ホスト・コンテナ間のセキュリティ的課題を解決するための，コンテナエンジンを提案する．

2. ホスト・コンテナ間のセキュリティ的課題

ホスト・コンテナ間，複数コンテナ間でのセキュリティ的課題について述べる．コンテナの運用には，特権がいくつも必要である．そのため，通常は特権ユーザが行う必要がある．よって，不正に特権ユーザにログインされた場合や，コンテナからホストに対して攻撃が行われた場合，その他のユーザが管理するコンテナを停止することや，ホスト自体に攻撃されることが考えられる．

既存のコンテナ型仮想環境では，コンテナを実行する際の権限を制限することは可能である．しかし，特定のコンテナの起動，停止をすべてのユーザが行える．例えば, ユーザ A が作成したコンテナを，ユーザ B は停止することができる．この課題を解決する方法として，ユーザごとのコンテナを隔離するために，ユーザごとに仮想マシン 1 台を割り当て，コンテナのホストを分離することが考えられている [4].

A Proposal of a Role-manageable Container Engine with File Permissions on Cloud Infrastructure
[†1]Takashi IIGUNI, Kagawa University
[†2]Hiroyuki TOMINAGA, Kagawa University

3. 提案手法

本研究では，ホストや他のコンテナを保護する機構をもつコンテナ型仮想環境 Cromwell を提案する．本システムは，コンテナ型仮想環境として，コンテナのネットワークや，リソースの分離を行うものである．既存のシステムに Docker や LXC が存在する．これらは，標準の設定ではコンテナの管理に特権が必要になる．本システムでは，コンテナに最小限の Linux Capability を与え，非特権ユーザによるコンテナの運用を前提として設計している．Capability とは，通常は特権ユーザに与えられる特権をいくつかに分類し，それぞれ独立して有効化，無効化できるものである．

図 1 に，本システムのアクセス制御機構を示す．ここで，1 つのアプリケーションに必要なコンテナ群をロール，ロールを管理することができるユーザ，グループをそれぞれ，オーナー，オーナーグループと定義する．ユーザは，オーナーに登録されていないコンテナの起動，停止を行うことができない．また，ロールの違うコンテナ間の通信は，サブネットを分けているため，他のロールのコンテナに対し，影響を及ぼさない．既存のコンテナ型仮想環境と比較すると，オーナー以外のユーザはファイル権限が異なるため，ホストからコンテナに対し，影響を与えにくい．

4. システム構成

本システムの構成について述べる．図 2 に構成を示す．本システムは，クライアントとサーバに分かれている．Unix ドメインソケット上で HTTP による通信を行う．

サーバ側は Go 言語で開発した．最低限，コンテナを作成する際に特権が必要な操作を担う．クライアントに対し，特に，3 つの機能を提供する．コンテナ作成機能は，コンテナの起動に最低限必要なパッケージのインストールを行う．ネットワーク管理機能は，ブリッジネットワークの構築や，コンテナへの IP アドレスの振り分けを行う．特権管理機能は，Capability をプロセスに対して有効または無効にする．サーバ側は，特権ユーザで実行する必要がある．

クライアント側は Rust で開発した．非特権ユーザでの起動を想定し，コンテナの管理を行う．コンテナの起

動，停止や，プロセス，ネットワークの分離，使用するリソースを制限する機能を有する．プロセスやネットワークの分離には，Linux Namespace を用いる．リソースの制限には Cgroup の機能を用いる．ルートディレクトリの変更，Namespace の分離，ファイルシステムのマウントを行うため，特権が必要である．そのために，Capability を付与するようにサーバ側に対してリクエストを行う．

5. 実装の状況

サーバ側の実装について述べる．Capability の付与を行うための機構を実装した．これは，クライアント側で unshare システムコールや，chroot システムコールを行うためである．ホストと通信するためのネットワークインタフェースの作成なども行う．また，これらの機能をクライアント側からのリクエストにフックして使用するため，Unix ドメインソケット上で HTTP による API を提供する．

クライアント側の実装について述べる．Cgroup により CPU やメモリの使用量に制限を設ける機構を実装した．これにより，コンテナの使用できるリソースを制限することができる．また，PID やネットワークの Namespace を分離する機構を実装した．これにより，ホストのプロセスをコンテナから見ることはできず，ネットワークもホストから切り離すことができる．よって，ロールにオーナー，オーナーグループを設定し，他のユーザからファイル権限によって隔離されたコンテナを起動することができる．そのため，他のユーザがコンテナの起動，停止を行おうとすると，アクセスが拒否される (図 3)．なお，ロール毎のサブネットの分離は未実装である．

6. おわりに

本研究では，コンテナのホストに対しての権限昇格や，他のコンテナへの意図しない影響を与えないことを目的とした．本システムは，ホスト・コンテナ間，複数コンテナ間におけるセキュリティ的課題を解決することを目的としたコンテナ型仮想環境である．コンテナの起動・停止は，オーナーとなったユーザのみが可能とし，それ以外の非特権ユーザからの命令を受け付けない．また，コンテナ間のネットワークを保護するため，ロールごとにサブネットを分離する．現状は，オーナ以外がコンテナの起動，停止を行おうとすると，アクセスが拒否される．

今後は，コンテナオーケストレーションのための機能追加と，Open Container Initiative[5] におけるコンテナランタイムの標準化規格に沿った実装を目指す．

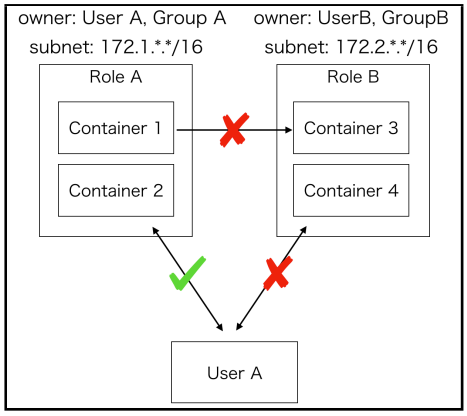

図 1 コンテナ間, ホスト・コンテナ間のアクセス制御

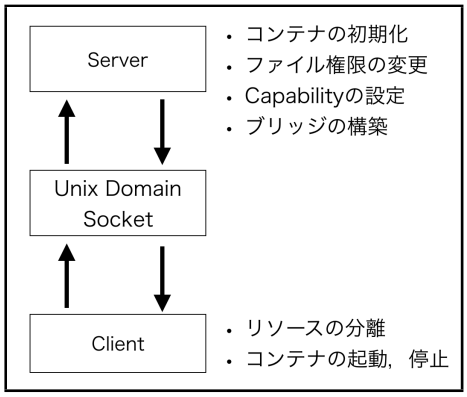

図 2 本システムの構成図

```
[userA@machineA ~]$ cromwell run --name archlinux-01
pid: 3315
Started initialize Container!
Mount rootfs ...
Mount container path ...
fork(2) start!
container pid: 3316
Mount procfs ...
[nobody@archlinux-01 /]$
```

```
[userB@machineA ~]$ cromwell run --name archlinux-01
pid: 3343
Started initialize Container!
thread 'main' panicked at 'Failed copy file: : Os
{ code: 13, kind: PermissionDenied, message:
"Permission denied" }', src/libcore/result.rs:1009:5
note: Run with `RUST_BACKTRACE=1` for a backtrace.
[userA@mathineA ~]$
```

図 3 本システムの実行ログ

参考文献

1) Docker Inc, Docker, https://www.docker.com/
2) Microsoft Azure, サーバーレス コンピューティングとは, https://azure.microsoft.com/ja-jp/overview/what-is-serverless-computing/
3) Daniel J Walsh (Red Hat), Are Docker containers really secure?, https://opensource.com/business/14/7/docker-security-selinux
4) OpenStack Foundation, Kata Containers, https://katacontainers.io/
5) The Linux Foundation, OCI, https://www.opencontainers.org/