Haconiwa: プログラムによる、組み立て可能性と拡張性を持つLinuxコンテナ

近藤 宇智朗[1,a)]　松本 亮介[2,b)]　栗林 健太郎[2]

1. はじめに

インターネットの広まりに伴い，ウェブサイトの運用者は突発的なアクセス増に遭遇しやすくなった．その理由としては，例えばソーシャルネットワークのような情報が急速に拡散するプラットフォームの出現，急激な負荷増加を引き起こす DoS 攻撃などが挙げられる．

従って特にネットワークサービスにおいて，負荷状況に応じシステムを構成するサーバ数を効率的に増減するオートスケールの需要が高まっている．この場合，物理的なサーバのリソースをさらに細かい粒度で制御できる仮想化されたサーバを利用することが多いが，サーバ数増加の際の起動速度向上のため，ハイパーバイザー型仮想化に比べてさらにリソース制御粒度が細かいコンテナ型仮想化に注目が集まっている．

一方で現状では，コンテナ型仮想化を用いても不要な機能が有効になったり，要件に応じた開発が困難な場合がある．この課題は特に高集積性を要求するホスティング用途において大きく問題となる．本論文ではこれらの課題を解決する手法として，機能の取捨選択ができ，フック機構や言語の組み込みにより拡張性を高めて対応した Haconiwa というコンテナランタイムを提案する．

本論文での用語を定義する．仮想化形式にはホスト型，ハイパーバイザー型があるが，これらは CPU などの物理的なデバイスを仮想化し，任意の OS を走らせることができるが，ホスト OS とカーネルを共有しデバイスの仮想化は行わない形式をコンテナ型仮想化と呼ぶ [1]．コンテナ型仮想化ではプロセスに OS の機能で独立性を持たせるという特徴があり，その一つ一つの独立性のあるプロセスをコンテナと呼ぶ．このコンテナの起動，停止などライフサイクルを管理するソフトウェアを本論文ではコンテナランタイムと呼ぶ．

[1] GMO ペパボ株式会社 技術部 技術基盤チーム
Developer Productivity Team, Engineering Department, GMO Pepabo, Inc.
[2] GMO ペパボ株式会社 ペパボ研究所
Pepabo Research and Development Institute, GMO Pepabo, Inc., Tenjin, Chuo ku, Fukuoka 810-0001 Japan
[a)] udzura@pepabo.com
[b)] matsumotory@pepabo.com

2. コンテナを利用したオートスケールの課題

1 章で述べたオートスケールについて，松本らにより以下の課題が提示されている [2].

(1) インスタンスの追加処理が低速である
(2) ハードウェアリソースの利用効率が低い
(3) 空きリソース確認のためのスケジューリングの遅延
(4) スケーリングすべき状況検知のリアルタイム性が低い

この課題についてコンテナを利用した場合，ハイパーバイザー型などと比べ (1)，(3) については起動・停止の高速さ，(2) に関してはリソース制御粒度をより細かくできる点により有利であろう．そのためコンテナ型仮想化の採用が進んでいる．既存のコンテナランタイムには Docker[3] や LXC[4] などがあり，Docker の利用がオートスケールを要求するシステムの管理に有用であるという提案がある [5].

しかし，既存のコンテナランタイムにおいては共通で次のような課題がある．

まず不要な機能を簡単に除外できない点である．コンテナ仮想化では 3 章で述べる通り多数の機能を利用するが，パフォーマンスや運用の簡便化などの要件から，特定の機能を無効化する需要が存在する．しかし既存のコンテナランタイムではすべての機能が有効になり，一部を除外することは用途として想定されておらず，例えば Docker のドキュメントは個々の機能の選択・無効化への誘導がない [6] ため，利用者が無効にすることは難しい．

次にシステム要件を満たすような拡張が簡単にできない点も挙げられる．文献 [5] によると複数のコンテナを利用したシステムの構築には Kubernetes[7] を利用できるが，例えばホスティング用途のような高集積の環境を実現するには，コンテナ自身の状態によりリソース割り当てを変更するなど細かい自己の制御が必要となる．このような拡張は既存のコンテナランタイムや構築ツールでは対応できない．

3. 提案手法: Haconiwa

筆者らの開発した Haconiwa は 2 章の課題を解決する．課題を解決可能なコンテナランタイムには，次の性質が求められるが，Haconiwa はこれらを満たすためである．

まず先述の通り既存のコンテナランタイムは，コンテナ作成時に後述するコンテナ仮想化機能を全て有効にする実装となっている．その組み合わせは簡単には変更できず，無理に実現した場合設定や構成の保守性が下がる．課題の解決のために，必要・不要な機能の組み合わせを利用者が簡潔に記述できる性質が求められる．

また，自身の状態による細かい自己の制御に関して，コンテナの起動・停止などのライフサイクルイベントや，コンテナ起動後一定期間後を契機として実行する処理の記述（フック機構）が重要となる．加えて，より特殊かつ前例のない要件にも対応するためには，外部の HTTP API との連携，コンテナ機能以外の OS 機能の活用などが可能になる高い拡張性が必要である．既存のコンテナランタイムではフック機構はライフサイクルイベントのサポートのみにとどまり，さらに前例のない要件に対しては自分自身を拡張することによる対応ができない．

3.1 コンテナ仮想化機能への対応

Haconiwa は，Linux でコンテナ仮想化を実現する次の機能にアクセス可能である．これらは他のコンテナランタイムでも利用されるが，Haconiwa を用いると要件やリソースの状況などに応じ自由に組み合わせ可能である．

(1) Linux Namespace[8]，chroot システムコールといった OS リソースの分離機能．
(2) cgroup[8]，rlimit システムコールといった OS リソースの制限機能．
(3) コンテナ内部でのスーパユーザの一部の権限の制限ができる Linux Capability．
(4) seccomp によるシステムコールのフィルタリング．

3.2 フック機構への対応

Haconiwa は，4 種類のフック機構をサポートし．自己の制御などの拡張が開発できる．

(1) 起動，停止後などライフサイクルイベントでのフック．
(2) 起動後，任意の時間経過後に非同期で行うフック．
(3) 起動後，一定間隔で定期実行する処理のフック．
(4) シグナルによる任意タイミングのフック．

3.3 スクリプト言語 mruby の組み込み

Haconiwa は mruby[9] と呼ばれる，軽量な Ruby の実装を組み込んでいる．

mruby のプログラムにより機能の組み合わせの条件やフック処理の詳細，あるいは外部サービスとの連携，OS 機能との連携などを記述することができ，コンテナに動的な性質を与える．また，それらの機能は外部のライブラリにまとめ再利用することができる．このように mruby は拡張性の向上に寄与している．

4. 本手法の利用例

Haconiwa を活用したアーキテクチャとして，松本らにより FastContainer と呼ばれる手法が提案されている [10]．

FastContainer は，コンテナの状態遷移を効率化して高速に循環させ，システム全体としての恒常性を実現する Web アプリケーションの基盤である．先述したような突発アクセスへの対応であったり，ライブラリ等のバージョンアップや入れ替えといった運用上の問題を解決できる．松本らが提案したアーキテクチャの具体的実装において Haconiwa が利用されている．組み合わせ可能性により起動の高速化を実現している点，フック処理など拡張性によりコンテナの一定時間後の停止などの FastContainer の要件を満たす点から採用した．

5. まとめ

Haconiwa は，コンテナ機能，各種フックを自由に組み合わせ，なおかつ mruby によりプログラムでコンテナの拡張を記述可能であり，高い組み合わせ可能性と拡張性を実現している．Haconiwa の特徴を生かした FastContainer のような具体的なアーキテクチャが存在する．

今後 Haconiwa では，多くのコンテナ機能のサポートや，Haconiwa を利用するエコシステムの拡充を実現していく必要があろう．

参考文献

[1] 榎 美紀, 堀井 洋, 小野寺 民也, 様々な負荷の Web アプリケーション Docker コンテナが混在するクラウド環境性能評価, DEIM Forum 2017, 2017 年 3 月.
[2] 松本 亮介, FastContainer:実行環境の変化に素早く適応できる恒常性を持つシステムアーキテクチャ（発表資料）, インターネットと運用技術シンポジウム 2017, https://www.iot.ipsj.or.jp/iots2017%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%A0/, 2017 年 12 月.
[3] Docker Inc, Docker, https://www.docker.com/.
[4] IBM Corporation, LXC, https://linuxcontainers.org/lxc/.
[5] David Bernstein, Containers and Cloud: From LXC to Docker to Kubernetes, IEEE Cloud Computing (Volume: 1, Issue: 3, Sept. 2014), pp.81-84, 2014 年 9 月.
[6] Docker Inc, Docker overview, https://docs.docker.com/engine/docker-overview/#the-underlying-technology.
[7] Cloud Native Computing Foundation, Kubernetes, https://kubernetes.io/.
[8] Rosen R, Resource Management: Linux Kernel Namespaces and cgroups, Haifux, 2013 年 5 月.
[9] Yukihiro Matsumoto, mruby, https://mruby.org/.
[10] 松本 亮介, 近藤 宇智朗, 三宅 悠介, 力武 健次, 栗林 健太郎, FastContainer: 実行環境の変化に素早く適応できる恒常性を持つシステムアーキテクチャ, インターネットと運用技術シンポジウム 2017 論文集, pp.89-97, 2017 年 12 月.