B-015

動作に必要な計算機資源とユーザ権限を抑えたコンテナ管理ツール
A Container Management Tool with Less Memory and User Privilege

三宅 貴義[1)]　　乃村 能成[1)]
Takayoshi Miyake　Yoshinari Nomura

1. はじめに

近年，コンテナによるアプリケーションの環境構築が増えている．たとえば，コンテナを用いて，ユーザのアプリケーション導入作業を簡便化したり，Raspberry Pi 等の小型計算機をエッジサーバとしてアプライアンス化したりすることがある．

こうしたコンテナの一般利用が増す中で，Docker に代表されるコンテナ管理ツール自体の導入の難しさや計算機資源圧迫が問題となる．

そこで我々は，エンドユーザにとって導入しやすいコンテナ管理ツール orca[1] を開発している．orca は，単一のバイナリを設置するだけで導入可能で，root 権限不要で動作する．また，Rust 言語で実装されており，動作において，高速，省資源，メモリ安全性を実現している．

本稿では，既存のコンテナ管理ツールの問題点を挙げ，orca の実装を示し，既存のコンテナ管理ツールとの性能比較について報告する．

2. 既存コンテナ管理ツールの課題と解決手法

コンテナ管理ツールは，コンテナイメージをダウンロードおよび管理する部分と計算機資源を分離したプロセスを作成する部分から成る．既存コンテナ管理ツールの課題と，その解決手法を以下に提案する．

(1)　計算機資源の圧迫
既存コンテナ管理ツールでは，管理用のデーモンを用いることがあり，デーモンの常駐によってメモリ資源が圧迫される．しかし，コンテナ内のプロセスをターミナル上で操作できる状態のコンテナ (**対話的なコンテナ**) のみを使用するならば，このデーモンは不要となる．
　そこで，対話的なコンテナのみを作成するコンテナ管理ツールを実装する．これにより，デーモンが不要になり，メモリ資源の圧迫が抑えられる．
　また，実装において，少メモリおよび高速で動作させるための工夫として，プログラミング言語 Rust を用いて実装する．Rust の処理系は，C 言語と同様に特別なランタイムを必要とせず，ネイティブコードで動作するため，メモリ使用量の削減，および C 言語と同等の実行速度を期待できる．

(2)　導入と削除作業の複雑化
コンテナは，ホスト計算機の環境に依存しない動作を期待されているが，コンテナ管理ツール自身は，ホスト計算機の環境に依存している．具体的には，コンテナ管理ツールをインストールする際，依存するライブラリも同時にインストールしなければならない．これにより，インストール作業が複雑になる．例として，Docker の公式サイト [2] では，インストールには 7 つのコマンドを実行しなければならないことが示されている．同様にアンインストール作業も複雑になるため，一時的な利用のために導入して即削除という要求に応え辛い．
　そこで，静的リンクな単一バイナリのコンテナ管理ツールを実装する．これにより，ホストにインストールされているライブラリに依存することなく導入でき，削除も容易になるため，導入と削除作業の複雑性を解消できる．

(3)　過剰な機能による実装の複雑化
コンテナ管理ツールは，ネットワーク特権ポートのバインドや，コンテナの保存や復元といった通常のユーザ権限では行えない機能を実装している．これらの過剰な機能により，コンテナ管理ツールの実装が複雑化している．しかし，コンテナの保存や復元といった操作はコンテナの使用法によっては必要ない．また特権ポートのバインドについては，特権ポート以外のポートで代用できる．
　そこで，これらの過剰な機能を省いたコンテナ管理ツールを実装することで，実装の複雑性を解消できる．

(4)　過剰なユーザ権限の使用
コンテナの作成には計算機資源の名前空間を分離する必要がある．既存のコンテナ管理ツールは root 権限といった特別なユーザ権限を使用して名前空間を分離する．これは，Linux における名前空間の分離には，特別なユーザ権限が必要だったためである．しかし，他の名前空間より後に追加されたユーザ名前空間の機能により，通常のユーザ権限で名前空間を分離できるようになった．
　そこで，ユーザ名前空間の機能を用いることで，通常のユーザ権限でコンテナを作成するコンテナ管理ツールを実装する．これにより，コンテナ管理ツールの用いるユーザ権限を抑えられる．また，コンテナ内で作成したファイルの所有者が root になってしまう問題に対しても，この方法により解決できる．

3. orca の実装および使用例

3.1 実装

2 章で示した解決法に基づきコンテナ管理ツール orca を実装した．orca は Rust 言語を用いて実装され，以下の特徴を持つ．

(1)　既存のコンテナ管理ツールと比べ，必要とする計算機資源を抑えている．
(2)　root 権限を必要とせず，ユーザ権限で動作する．

また，orca は静的リンクな単一のバイナリとして提供され，以下のコマンドによりインストールできる．

1)　岡山大学大学院自然科学研究科，Graduate School of Natural Science and Technology, Okayama University

表 1　評価を行った環境

項目	バージョン
OS	Linux (Ubuntu 20.04.2 LTS)
カーネル	5.8.0-50-generic
Docker	20.10.3
podman	3.1.2
orca	0.1

```
$ curl -L "https://github.com/nomlab/orca/
  releases/latest/download/orca" > orca
$ chmod +x orca
```

3.2 使用例

root 権限を自由に利用できない共用の計算機において，すでにインストールされている gcc に影響を与えずに上位のバージョンのものを使用してコンパイル作業を行いたい要求がある．

この要求に対して，Docker を用いて gcc をインストールしようとした場合，Docker 自体のインストールや実行に root 権限が必要となり，要求を満たすことができない．一方 orca は以下のコマンドにより，root 権限を必要とせず gcc のインストールができる．このため，orca を用いることでこの要求を満たすことができる．

```
$ ./orca -d alpine  bash
# apk update && apk add gcc git
# git clone https://example.com/example.git
# gcc example/code_for_new_gcc.c
```

4. orca の評価

4.1 評価する項目とその手法

orca の評価項目と評価手法を以下に示す．

(1)　起動時間の比較
コンテナの起動時間を 3 つのコンテナ管理ツール，Docker，podman[3]，orca を対象に比較する．コンテナの起動時間は以下の一連の動作が完了するまでの時間とする．なお，時間の計測には time コマンドを用いる．
(A)　コンテナ管理ツールの起動
(B)　コンテナイメージ (alpine) の展開
(C)　コンテナの作成
(D)　true コマンドの実行

(2)　メモリ使用量の比較
コンテナ管理ツールのメモリ使用量を 3 つのコンテナ管理ツール，Docker，podman，orca を対象に比較する．メモリ使用量は，コンテナが存在しない場合 (停止時) と対話的なコンテナが存在する場合 (起動時) のそれぞれについて計測する．また，コンテナ管理ツールが複数のプロセスから成る場合は，それらすべてのプロセスのメモリ使用量の合計をコンテナ管理ツールのメモリ使用量とする．なお，メモリ使用量の測定には ps コマンドを用いた．

また，評価を行った環境を**表 1** に示す．

4.2 評価結果

評価結果を**表 2** に示し，考察を加える．

表 2　起動時間とメモリ使用量の比較

ツール名	起動時間	メモリ使用量 (停止時 - 起動時)
Docker	1,027 ms	167 - 231 MB
podman	1,302 ms	24 - 139 MB
orca	55 ms	0 - 8.2 MB

(1)　起動時間の比較
表 2 より，Docker の起動時間は 1,027 ms である．一方，orca の起動時間は 55 ms であり，Docker の起動時間と比較すると 20 分の 1 程度となっている．これは，Docker と orca のコンテナイメージの管理方法が異なることが要因であると考えられる．Docker はコンテナイメージをレイヤと呼ばれる複数のファイルシステムで管理しており，コンテナの起動時にこれらのレイヤを統合し，コンテナイメージを作成する．一方，orca はコンテナイメージを一つのファイルシステムで管理しており，コンテナの起動時にレイヤの統合が必要ない．このようなコンテナイメージの管理方法の違いにより，orca の起動時間は Docker と比較して減少したと考えられる．

(2)　メモリ使用量の比較
表 2 より，orca は，コンテナ管理のためのデーモンが存在しないため，コンテナが存在しない場合はメモリ使用量がゼロとなる．一方，Docker は，コンテナが存在しない場合においても，167MB のメモリを使用する．

また，起動時については，Docker では 167 MB → 231 MB と，64 MB のメモリ使用量の増加が見られるが，orca は，8.2 MB である．これは，起動時を見ると Docker の 30 分の 1 程度，停止時から起動時の増加分だけで見ても 8 分の 1 程度の増加に抑えられていることが分かる．

5. おわりに

既存のコンテナ管理ツールの問題点への対処として，計算機資源とユーザ権限を抑えたコンテナ管理ツール orca を提案した．

orca は Docker と比較して，コンテナの起動時間が 20 分の 1 程度に抑えられていた．また，コンテナの起動時のメモリ使用量は 30 分の 1 程度，停止時からのメモリ使用量の増加も 8 分の 1 程度に抑えられていた．さらに，orca が Docker より有効に使用できる例を示した．

参考文献

[1] nomlab: orca, Okayama University (online), available from ⟨https://github.com/nomlab/orca⟩ (accessed 2021-06-14).
[2] Docker Inc.: Docker, Docker Inc. (online), available from ⟨https://www.docker.com/⟩ (accessed 2021-06-14).
[3] containers organization: podman, containers organization (online), available from ⟨https://podman.io/⟩ (accessed 2021-06-14).