1A-06

組込み機器向けコンテナランタイムの評価

堀井　圭祐†　　大木　英俊†　　上野　貴廣†　　山田　竜也†　　水口　武尚†

三菱電機株式会社　情報技術総合研究所†

1. はじめに

エンタープライズ系システムでは既に広く普及しているコンテナ型仮想化技術であるが，システムに対する要求拡大や搭載されているマシンスペックが向上していることから，近年は組込み機器への適用も検討され始めている[1].

コンテナ型仮想化を実現するためには，Docker[2]に代表されるコンテナランタイムを利用することが一般的である．コンテナランタイムはOSS での開発が盛んであり，複数の選択肢があるため，システムの特性に応じて適切なものを選定することが望ましい．そこで，組込み環境に適したコンテナランタイムを抽出するために，定性的側面・性能的側面で評価を実施する．

本稿では，組込み機器向けコンテナランタイムの評価内容と結果について述べる．

2. コンテナランタイムの概要

コンテナランタイムは，大きく次の 2 種類に分けられる．

- **高レベルランタイム**
  コンテナイメージを管理しており，コンテナのルートファイルシステムとなるディレクトリ作成などを行う．デーモンプロセスとして動作することが多い．
- **低レベルランタイム**
  実際にコンテナの生成・実行・停止・削除を行う．低レベルランタイムの仕様は OCI (Open Container Initiative) により定義されており，高レベルランタイムからは仕様に準拠した json ファイルを介して指示を受け取る．バイナリとして実装される．

上記以外にも Docker のように高レベルランタイムよりも上位で動作するソフトウェアを含んだものや，1 つのランタイムで高レベルと低レベルの両方の機能が提供されるものも存在する．組込み機器のようなリソースに制限がある環境では低レベルランタイムのみを使用するケースも考えられるが，今回は選択肢を制限せずに高レベル／低レベル／その他のランタイムをそれぞれ比較することとする．

3. 評価

3.1. 定性的な比較

高レベルランタイムとしては，cri-o，Podman，containerd の 3 つを比較する．各ランタイムの特徴と主な違いを次に示す．

- **特徴**：cri-o は Kubernetes の利用が前提であり，Pod 内でコンテナを管理．Podman は Docker と互換性があり，多機能．containerd は Docker の標準ランタイム．
- **CRI(Kubernetes との I/F)**：Podman 以外は準拠
- **デーモンの有無**：Podman 以外はデーモン有

低レベルランタイムとしては，runc，crun，runsc (gVisor) の 3 つを比較する．各ランタイムの特徴と主な違いは，次のとおりである．

- **特徴**：runc は Docker の標準ランタイム．crun は C 言語で開発されており，runc よりも省リソース化を実現．runsc はサンドボックス化することで，ホストからの隔離性を強化．
- **バイナリサイズ**：
  crun ＜ runc(約 1.9 倍) ＜ runsc(約 3.6 倍)
- **ホストとのカーネル共有**：runsc のみゲストカーネル上で動作するため，共有しない

最後に，その他のタイプとして Docker と balenaEngine も比較対象とする．Docker は豊富な機能を備え，事実上のデファクトスタンダードである．balenaEngine は組込みや IoT での利用を想定し，Docker をベースに不要な機能を削減することで，省リソース化を実現している．また，単一のバイナリで高／低レベルランタイムの機能を提供する．

3.2. 性能測定

3.2.1. 性能測定概要

前節で述べたコンテナランタイムを対象に性能測定を行う．性能測定時のコンテナランタイムの組合せを表 1に示す．同レベルのランタイム間で性能差を比較できるように，もう一方のランタイムは固定する．また，測定項目は下記の 3 項目とする．

- **コンテナライフサイクルコマンド性能**
  コンテナの操作（生成，開始，終了，削除，状態確認）コマンド実行に要する時間を測定
- **ストレージアクセス性能**
  不揮発領域への読み書き性能を bonnie++（ファイルシステムのベンチマーク）により測定
- **メモリ使用量**
  コンテナ実行時に消費されるメモリ量を測定

性能測定環境を表 2に示す．

表 1　性能測定対象のコンテナランタイム一覧

#	高レベルランタイム	低レベルランタイム
1	cri-o v1.21.3	runc v1.0.2
2	Podman v3.3.1	
3	containerd v1.5.2	
4		crun v0.21
5		runsc(gVisor) v20210830.0
6	Docker (dockerd+containerd+runc) v20.1.7	
7	balenaEngine (daemon+containerd+runc) v17.12.0	

表 2　性能測定時の動作環境

項目		仕様
組込みボード		Raspberry Pi4 Model B Rev 1.2
	CPU	Arm Cortex-A72, ARMv8 64bit, 1.5GHz, 4 コア
	メモリ	LPDDR4, 4GB
	ストレージ	microSD カード, 32GB
OS		Linux 5.4.0-1042-raspi, Ubuntu 20.04 LTS

An evaluation of container runtime for embedded systems
Keisuke Horii†, Hidetoshi Oki†, Takahiro Ueno†, Tatsuya Yamada†, Takehisa Mizuguchi†
† Information Technology R&D Center, Mitsubishi Electric

3.2.2. 測定結果

コンテナライフサイクルコマンド性能の測定結果を図 1に示す．図 1の縦軸は測定対象の組合せ（表1の 1 列目に対応），横軸は各コマンドの処理時間の合計を示す．また，キャッシュ無効時（上段）と有効時（下段）の結果をそれぞれ示す．図 1より，高レベルランタイムの中では Podman が最も低速であり，containerd と cri-o は同程度である．低レベルランタイムでは 3 種類とも同程度である．Docker と balenaEngine は，Podman+runc を除いて他の組合せよりも低速である．Podman+runc と Docker は特に start と stop に時間を要している．balenaEngine は stop 以外のコマンドは高速である．

次に，ストレージアクセス性能の測定結果を表 3に示す．表 3では bonnie++の測定項目の内，シーケンシャルな書込み（キャラクタ単位／ブロック単位／ブロック単位の再書込み）と読込み（キャラクタ単位／ブロック単位），ファイルのランダムシーク性能を示す．表 3より，高レベルランタイム間で性能差はほとんどなく，低レベルランタイムの中では全ての項目で runsc の性能が最も低い．全体では，containerd+runsc 以外の組合せを除いて概ね横並びの結果である．

最後に，メモリ使用量の測定結果を図 2に示す．図 2ではコンテナの起動から終了までのピーク使用量（free コマンドを 1 秒間隔で実行し，初期値との差分を測定）と高レベルランタイムにおけるデーモンプロセスのメモリ使用量（ps コマンドの RSS を測定）の合計値を示す．Podman はデーモンレスであるが，参考としてコンテナ起動時に実行されるプロセス（podman）のメモリ使用量を示す．図 2より，高レベルランタイムのピーク使用量は同程度であるが，デーモンプロセスを含めると containerd が最も多く，デーモンレス型の Podman が最も少ない．低レベルランタイムでは crun が最も少なく，runsc が最も多い．全体では，Docker が他よりも 2 倍近く多く，Podman+runc の組合せが最も少ない．

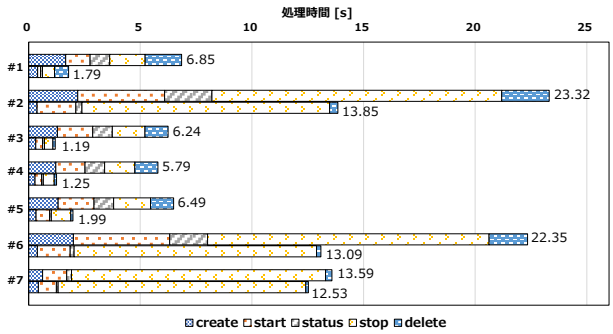

図 1　コンテナライフサイクルコマンド性能測定結果

表 3　ストレージアクセス性能測定結果［KiB/Sec］

	Write			Read		Seek
#	Char	Block	Rewrite	Char	Block	
1	88	13,840	10,477	269	41,810	2,563
2	89	13,744	10,394	293	41,832	2,273
3	88	13,971	10,482	270	41,782	2,518
4	91	14,075	10,571	280	41,605	2,471
5	88	13,505	7,615	6	39,121	2,380
6	89	13,937	10,474	293	41,788	2,359
7	91	13,360	10,113	267	41,429	2,350

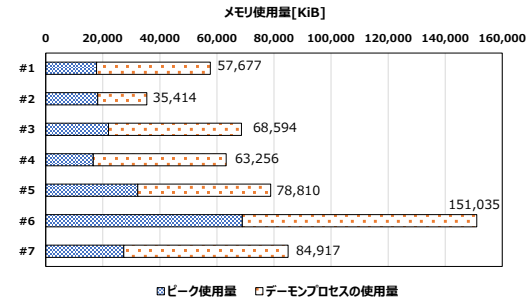

図 2　メモリ使用量測定結果

4. 考察

3章の評価結果より，組込み機器に適したコンテナランタイムを検討する．

高レベルランタイムは Podman と containerd が候補として考えられる．cri-o は Kubernetes による管理を前提としており，Kubernetes なしで利用するメリットはない．containerd は突出して優れた点があるわけではないものの，定性的側面と性能面のどちらにおいても標準的である．Podman はメモリ使用量が少なく，デーモンプロセスの異常がコンテナに波及しないというメリットもある．ただし，コマンド実行は他と比べて低速であるため，起動や終了に時間制約がある場合は注意が必要である．

低レベルランタイムについては runc と crun が候補として挙げられる．runsc はストレージアクセス性能が低く，消費リソースも多い．そのため，セキュリティ的側面での要求が厳しくない限り，使用するメリットはないと考えられる．crun はバイナリサイズなどのリソース面を除いて，定性的側面において runc との差異はなく，今回の測定結果から性能差もほとんどない結果であった．

その他には balena Engine も候補として考えらえる．Docker は組込み機器にはオーバスペックである可能性が高く，性能面でのデメリットも大きい．一方，balena Engine は Docker の機能を保ちつつ，ある程度の省リソース化を実現できていることから，機能面の要求が高い場合には適用の可能性がある．

5. おわりに

今回，組込み機器へのコンテナ適用に向けて，定性的側面と性能的側面からコンテナランタイムの評価を行った．その結果，組込み機器向けの balena Engine に加え，高レベルランタイムでは Podman や containerd，低レベルランタイムでは runc や crun のようなエンタープライズ系のコンテナランタイムも適用可能性があることを確認できた．

今後は，他のコンテナランタイムの追加やネットワーク性能，ベンチマークテストといった今回の測定対象からは除外した項目の評価などを実施する予定である．

参考文献

[1] 宮田幸太，菊田祐司，宮崎剛，“産業用組込機器におけるコンテナ管理技術の適用評価”，第 83 回情報処理学会全国大会講演論文集，Vol.1，No.1A-05，Mar. 2021.
[2] Docker. Inc，Docker，“https://www.docker.com/”