Container Runtime Meetup (2021/1/28)

DockerとPodmanの比較

日本電信電話株式会社
ソフトウェアイノベーションセンタ
須田 瑛大

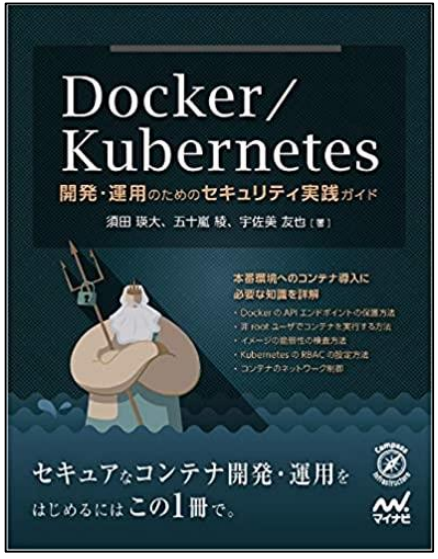
自己紹介

色々なコンテナ関連OSSのメンテナ (コミッタ)

- Moby (OSS版Docker), BuildKit, containerd, runc
- RootlessKit, slirp4netns (Docker・Podman共通のRootlessコンテナツールキット)
- Podmanのリポジトリにもwrite権限を持っている

「Docker/Kubernetes 開発・運用のための
セキュリティ実践ガイド」執筆

- Docker/Kubernetesだけでなく Podmanも網羅
- https://www.amazon.co.jp/dp/4839970505



Podmanとは

- Docker互換のコンテナエンジン

```
$ podman run -p 80:80 --name nginx docker.io/library/nginx
```

- RHEL, CentOS, Fedora などに 標準で付属している

- Kubernetesで動いている**Pod**を**man**ageできるわけでは**ない**
  ただしCRI-OのPodの一覧とdiffの表示くらいはできる (podman ps --external, podman diff)

主な比較観点

技術以外の観点も比較

アーキテクチャ	リリースサイクル	コミュニティ
CLI	API	Compose
ネットワーク	イメージビルド	セキュリティ

比較するバージョン

- Docker 20.10.2

- Podman 3.0.0-rc1
  - RHEL/CentOS 8.3 に付属するバージョンは 2.0
  - RHEL 8系列にPodman 3.xが入るのはRHEL 8.5以降と思われる

結論から先に

- デーモンを起動しなくても使えるのがPodmanの最大の特徴
  - ただし、OS起動時にコンテナも起動するには結局systemdの設定を触る必要があるので、好みがわかれる
- リリースサイクルの点ではPodmanが魅力的
- 機能面や性能面では、重要な違いは少ない
  - 2020年までは色々違っていた
  - イメージビルドに関してはPodmanはまだ遅れている

比較観点: アーキテクチャ

- Docker はデーモン(dockerd)とクライアント(docker CLI)に分かれており、REST APIで通信する

- **Podman は基本的にデーモン無しで動作する**
  - 厳密にはデーモン無しで動作するわけではなく、むしろコンテナ毎 (Pod毎) に小さいデーモン (conmon) をバックグラウンドで起動する
  - ホストOS起動時にコンテナを自動的に起動するには、systemd unitをいちいち作成する必要がある
    - `podman generate systemd <CONTAINER|POD>`
    - `podman system boot` (unitファイルを1つで済ませる仕組み。Podman 3.1前後で入る見込み #8828)

比較観点: アーキテクチャ

- バッチジョブ型のコンテナを実行する場合は、Podmanの方が使いやすいかもしれない

- サービス型のコンテナを実行する場合は、DockerでもPodmanでも、systemdを設定するから手間から逃れられない
  - **どちらの方が使いやすいかは微妙**
  - コンテナ毎に `podman generate systemd` しないといけないので、(Podman 3.1がリリースされるまでは)、むしろPodmanの方が不便かもしれない

比較観点: アーキテクチャ

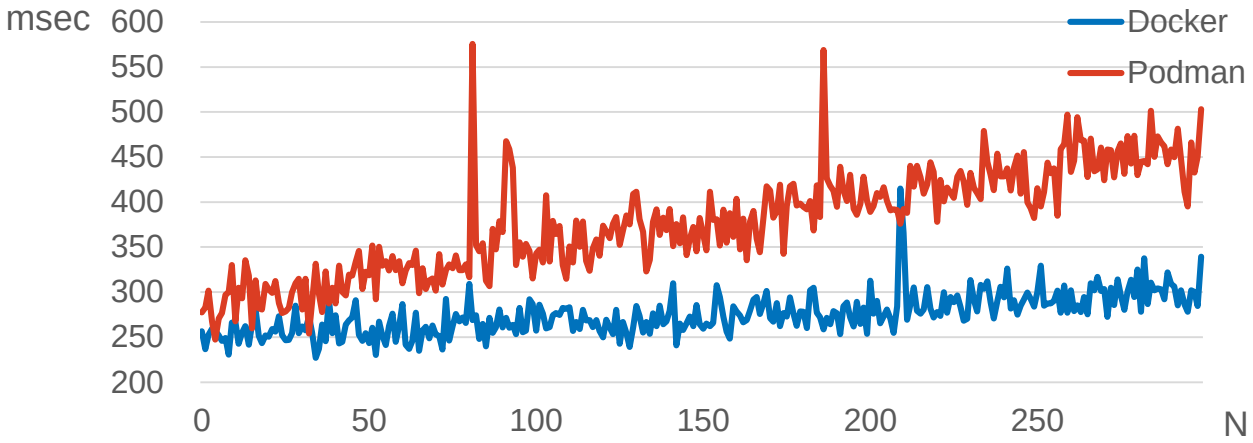
- Podmanの方が、ランタイムのバグが顕在化しにくい
  - あるコンテナを担当するPodmanプロセスやconmonプロセスが、クラッシュしたりメモリリークしたりしていても、他のコンテナには影響しない

- Dockerでは live-restore モードを有効化すれば、デーモンのクラッシュによるコンテナへの影響は軽減できるが、完全ではない

比較観点: アーキテクチャ

ベンチマーク

```
# docker pull nginx:alpine
# ntimes -n 300 docker run -d nginx:alpine
```



Podmanのほうが若干遅いが、大した差ではない

Docker: overlay2, runc / Podman: overlay, crun / ホスト: Ubuntu 20.10、CPU 4コア、RAM 8GB

比較観点: アーキテクチャ

NTT

ベンチマーク

```
# docker pull nginx:alpine
# ntimes -n 300 docker run -d nginx:alpine
```

	Docker	Podman
平均	275 ms	377 ms
最大	415 ms	576 ms
最小	227 ms	247 ms
標準偏差	23 ms	58 ms
99th パーセンタイル	337 ms	500 ms
95th パーセンタイル	310 ms	465 ms
50th パーセンタイル	273 ms	381 ms

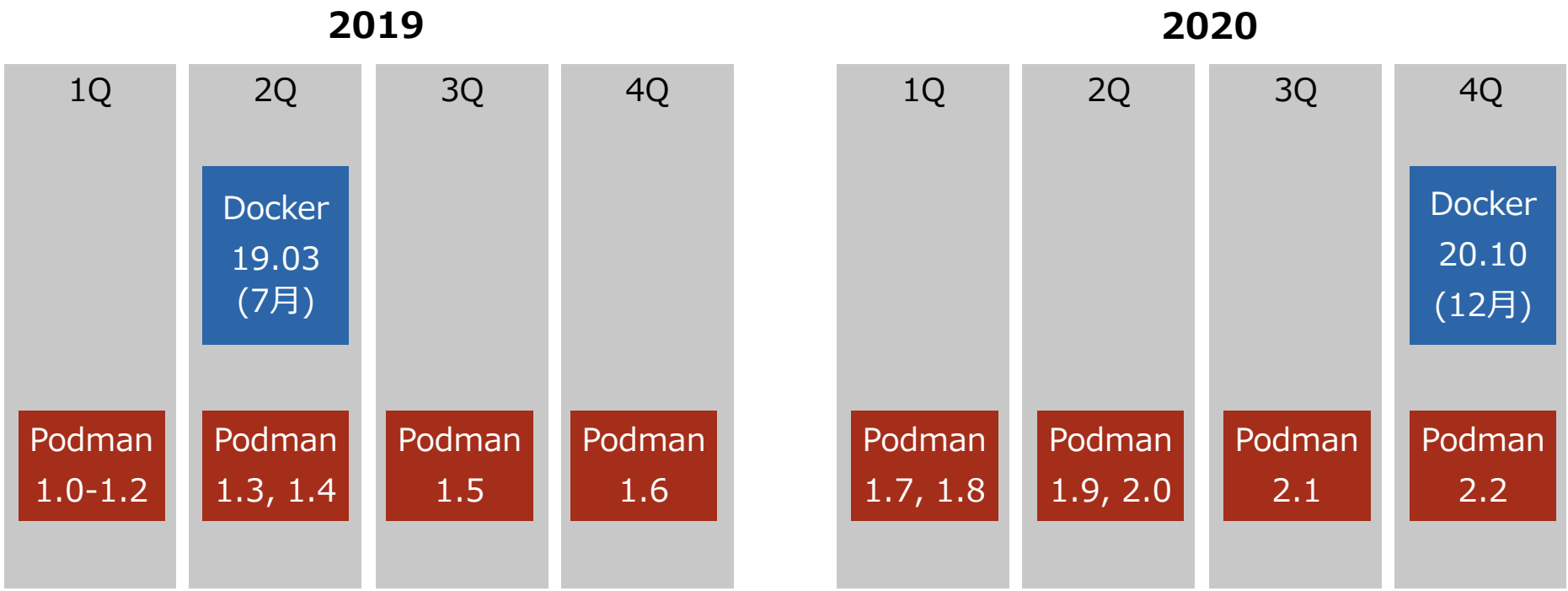
Docker: overlay2, runc / Podman: overlay, crun / ホスト: Ubuntu 20.10、CPU 4コア、RAM 8GB

比較観点: リリースサイクル

NTT

- Docker界隈は新機能のpull requestが投稿されても、マージされ、かつ、リリースが出るまでには時間がかかる
- **Podmanの方がリリースサイクルが格段に早い**
- コンテナ界隈の新機能を(野良ビルドせずに)試すにはPodmanの方が向いていることが多い

比較観点: リリースサイクル



※細かいリリース (x.y.zのz) については、Dockerも頻繁にリリースしている

比較観点: コミュニティ

- Docker (の大半) は Docker社を中心とするMoby コミュニティにより開発されている (github.com/moby)
- Podman はRed Hat社を中心とする github.com/containers コミュニティにより開発されている

- Docker社が単独でDocker (Moby) を作っているわけではないし、Red Hat社が単独でPodmanを作っているわけでもない
- いずれのコミュニティも、CNCFなどのfoundationには属していない
- いずれも大部分がオープンソース (Apache License 2.0)
  - Docker for Mac/Win はプロプラ部分が多い

比較観点: コミュニティ

- Red Hat社がDockerに投稿したパッチの一部がマージされなかったことが、Podman発足の背景として挙げられる

- 良くも悪くも、Podmanの方がパッチのマージには積極的

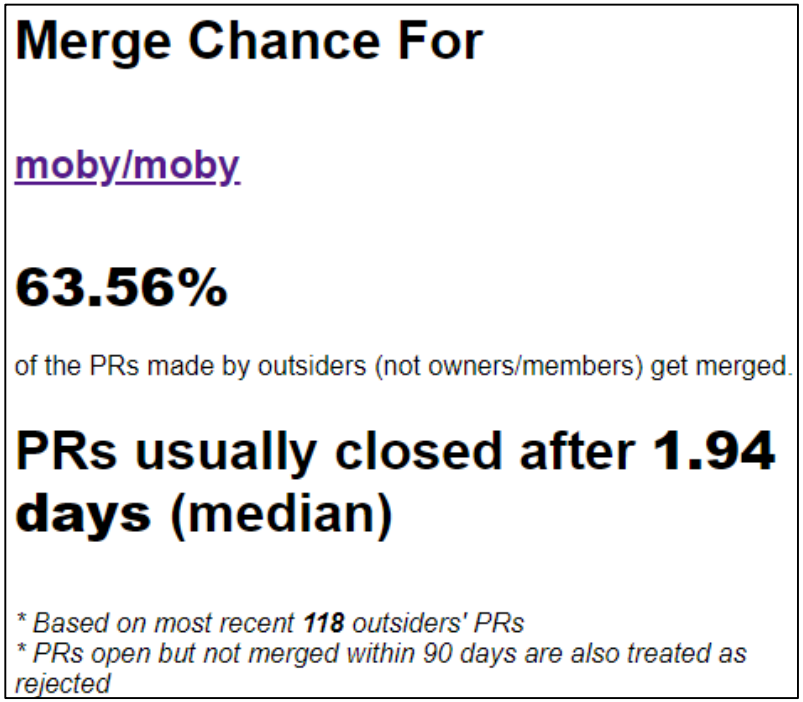
Table of Patches

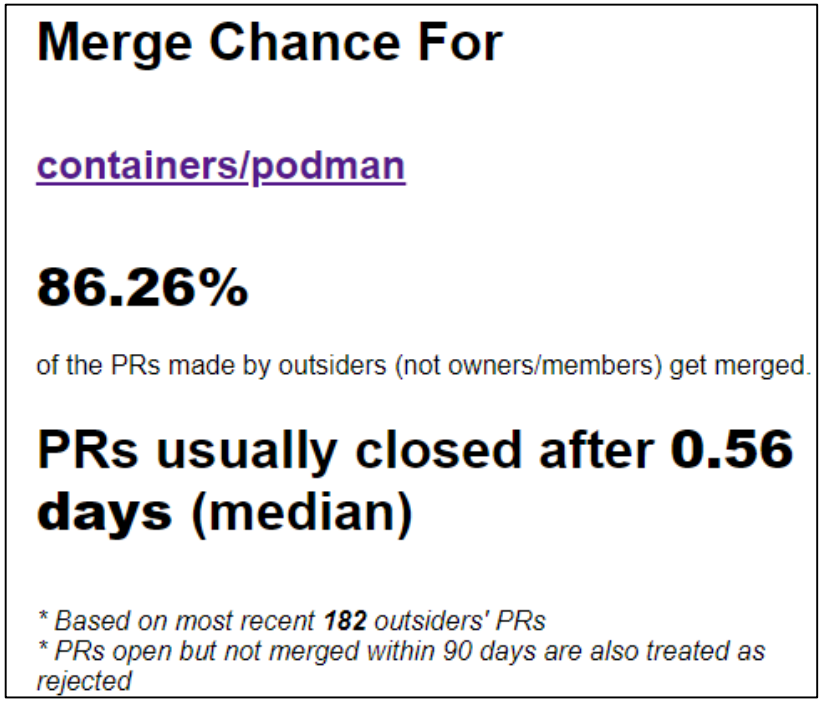
Name	Type	PRs	Status
Add-RHEL-super-secrets-patch	Red Hat	6075	Maintaining
Add-add-registry-and-block-registry-options-to-docker	Upstream	11991, 10411	Rejected, Maintaining
Improved-searching-experience	Upstream		Rejected, Maintaining
Add-dockerhooks-exec-custom-hooks-for-prestart/poststop-containers	Upstream	17021	Pending
Return-rpm-version-of-packages-in-docker-version	Red Hat	14591	Maintaining
rebase-distribution-specific-build	Upstream	15364	Pending
System-logging-for-docker-daemon-API-calls	Upstream	14446	Blocked
Audit-logging-support-for-daemon-API-calls	Upstream	109	Blocked
Add-volume-support-to-docker-build	Upstream	See Below	See Below

https://www.projectatomic.io/docs/docker_patches/

比較観点: コミュニティ

https://merge-chance.info/





2021/1/25 取得

大きい違いは以上

あとは細かい話

比較観点: CLI

- PodmanのCLIはDockerとほぼ同じ
- 基本的に `alias podman=docker` して良い
- `podman network inspect` など一部のコマンドは出力が非互換

比較観点: API

- Podman 1.x の API (varlink) はDocker非互換

- Podman 2.0からはDocker REST APIに対応
  - `podman system service [--timeout=0]` コマンドでDocker互換デーモンが起動する (`/run/podman/podman.sock`)

- 2.x では互換性が低かったが、3.0 では ほぼ問題ない
  - 3.0 でも細かいところは非互換

比較観点: Compose

NTT

- かつてのPodmanは、Docker Compose との互換性を追求することに消極的だった
  - 複数のコンテナをPodに収めて、Pod内のlocalhostソケット経由で通信させることが推奨されていた

- Podman 3.0 では、Docker互換APIを通じて
  docker-compose コマンドをそのまま実行できる

```
# export DOCKER_HOST=unix:///run/podman/podman.sock
# docker-compose up
```

比較観点: Kubernetes マニフェスト

NTT

- Podman は名前の通り、Podをfirst classなオブジェクトとして扱える
  - Pod内のコンテナはnetwork namespace などを共有するので、localhostのソケットで通信できる (サイドカーに便利)
  - Dockerでも `docker run --net=container:<SANDBOX>` コマンドで実現できるが、やや面倒
- KubernetesのPodとは独立している。Kubernetesで動いている**Pod**の**man**agement とは**基本的に関係ない**。
  - **podman ps** 、 **podman diff** などごく一部のコマンドはKubernetes (CRI-O) のPodにも対応

比較観点: Kubernetes マニフェスト

- `podman play kube` コマンドで、KubernetesのPodマニフェストやDeploymentマニフェストを直接実行できる

- が、極めて基本的なマニフェストしか実行できない

- Kubernetesのマニフェストをローカルで動かしたい場合は、**kind** (Kubernetes-in-Docker) を使う方が良い
  - Docker、Podman 両対応
  - 軽量さでは `podman play kube` コマンドの方が魅力的ではある

比較観点: Kubernetes マニフェスト

- 結局、Podman で Pod を扱う場面は多くはない

- 個人的には “Podman” は いい名前ではないと思う

比較観点: ネットワーク

- Docker は libnetwork を使うのに対し、Podman は Kubernetes と同様に CNI (Container Network Interface) を使う

- PodmanでもDockerと同様に `podman network create` コマンドでカスタムネットワークを作成できる

比較観点: ネットワーク

NTT

- ただし、Podman ではカスタムネットワーク間の通信が隔離されないので、カスタムネットワークを作る意味があまりない (Podman 3.0rc1 現在)

- CNI isolation pluginを別途インストールすると、Podman でもネットワークを隔離できる
  - https://github.com/AkihiroSuda/cni-isolation
  - Podman 3.1 か 3.2 ころでデフォルトになることを目指している (#5805)

比較観点: イメージビルド

- Docker は BuildKit を使うのに対し、Podman は Buildah を使う

- **Buildah は BuildKit に比べるとかなり遅れている**印象
  - マルチステージDockerfileを並列ビルドできない
  - `RUN --mount=type=(cache|secret|ssh)` など最新のDockerfile syntaxに対応していない
  - マルチアーキテクチャイメージを簡単にビルドできない

比較観点: イメージビルド

- Podman向けに複雑なイメージをビルドする際は、BuildKitの buildctl コマンド を使うのが良さそう (Docker不要)

```
$ buildctl build ... --output type=oci | podman load foo
```

比較観点: セキュリティ [Rootless]

- Dockerも Podmanも Rootless モード に対応している
  - 非rootでコンテナランタイムを実行することで、root権限を奪われにくくする技術

- Podman の方がリリースが頻繁なこともあり、Podman固有の機能と勘違いされがち
  - 実装自体は連携とりながらほぼ同時期

- いずれもUser Namespaces、RootlessKit、slirp4netns など共通する技術基盤を使っている

比較観点: セキュリティ [Rootless]

- Docker 20.10、Podman 2.1以降では機能的にほぼ同等
  - Docker 19.03以前のRootlessでは `docker run` の `--memory` や `--cpus` が使えなかった
  - Podman 2.0以前のRootlessでは `podman network create` できなかった
- ただし、**Docker Compose は Podman の Rootless では十分に動かない** (Podman 3.0 rc1 現在)
  - `podman network connect` が未実装のため
  - 3.1か、3.2 頃までには動くようにできそう

比較観点: その他

- Docker Swarm に相当する機能は Podman には無い

- Docker for Mac/Win に相当するプロダクトは Podman には無い
  - ただし、Red Hat Code Ready Containers (ラップトップ用OpenShift) を入れると、Podman も一緒についてくる
  - Docker for Mac/Win、 Red Hat CRC いずれともプロプライエタリな部分が大きい

まとめ

- デーモンを起動しなくても使えるのがPodmanの最大の特徴
  - ただし、OS起動時にコンテナも起動するには結局systemdの設定を触る必要があるので、好みがわかれる
- リリースサイクルの点ではPodmanが魅力的
- 機能面や性能面では、重要な違いは少ない
  - 2020年までは色々違っていた
  - イメージビルドに関してはPodmanはまだ遅れている

補足資料

ありがちな誤解

以下は全て**誤解**

- 「PodmanはKubernetesが呼び出すランタイムである」
- 「RHEL/CentOS 8 や Fedora ではDockerは動かないのでPodmanを使う必要がある」
- 「RootlessコンテナはPodman固有の機能」
- 「PodmanではDocker Composeは使えない」

補足資料

誤解「PodmanはKubernetesが呼び出すランタイムである」

NTT

- PodmanがmanageするPodは、基本的にKubernetesのPodと関係ない
  - (Virtual Kubelet を使えばPodmanのPodをKubernetesでオーケストレートできるが、POC留まり)

- PodmanはKubernetesのランタイムであるとする言説を散見するが、おそらく CRI-O と混同している
  - Podman とコードや主要開発者が重なっている
  - CRI-O は Docker ではなく containerd と競合する

補足資料

誤解「RHEL/CentOS 8 ではDockerは動かないのでPodmanを使う必要がある」

- RHEL 8 や CentOS 8 には Docker のRPMが含まれないのは事実
  - すなわち、RHEL ユーザへのRed Hat社によるサポートもない

- CentOS 8 には Docker社公式のRPMが提供されている
  https://download.docker.com/linux/centos/8/

- CentOS 8 とバイナリ互換性がある他のOSでもおそらく動作する

補足資料

誤解「Fedora ではDockerは動かないのでPodmanを使う必要がある」

- Docker 19.03 は Fedora 31以降のデフォルトの構成では動かなかった
- Docker 20.10 は cgroup v2 に対応したので、Fedoraでもデフォルトで動く

- Fedoraが配布しているパッケージは19.03で止まっている
  - Docker社が配布しているパッケージは20.10対応 (https://get.docker.com)
  - Fedora 34リリース時には、Fedoraからも20.10が配布されるはず

補足資料

誤解「Rootlessコンテナは Podman固有の機能」

NTT

- Rootlessコンテナ自体は2013年から存在 (unprivileged LXC)
- 2016年-2017年にはruncのRootlessが実現
- 2018年にはcontainerd、BuildKit、imgのRootlessが実現

- 続いて、PodmanやDockerのRootlessも実現
  - Red Hatや弊社で連携とりながら、ほぼ同時期に実装
  - Docker関連リポジトリのpull requestのマージやリリースに時間がかかっている間に、PodmanがRootlessコンテナの代表として認知されるようになった

補足資料

誤解「PodmanではDocker Composeは使えない」

NTT

- Podman 3.0 ではAPIの互換性が十分に高いので、Docker Compose をそのまま実行できる
  - `export DOCKER_HOST=unix:///run/podman/podman.sock` してから `docker-compose` コマンドを実行するだけ

- Podman Compose という独自実装もある
  https://github.com/containers/podman-compose
  - 古いバージョンのPodmanにも対応