Engineer

Podman

デーモンを必要としないLinux向けコンテナエンジン

どのようなものか

Podmanは、Dockerと同じOCIイメージをビルド、取得、実行するLinux向けコンテナエンジンです。間に中央サーバーを置きません。各 `podman` コマンドは、自身のプロセス内で処理を行って終了します。起動したコンテナは、conmonという小さな監視プロセスの下で動き続けます。この監視プロセスは、起動元となったシェル、スクリプト、systemdなどの子プロセスです。プロジェクトは自身を、デーモンを使わない、オープンソースでLinuxネイティブなエンジンと説明し、大半のユーザーは `docker` で `podman` を呼び出すエイリアスを設定して、そのまま作業を続けられると述べています。

名前はpod managerの略です。Kubernetesから取り入れたpodは、ネットワーク名前空間を共有するコンテナの集まりで、localhostで互いに通信できます。Podmanではpodを手作業で作成でき、 `podman kube play` なら、KubernetesのYAMLファイルからコンテナとpodを起動できます。そのため、クラスターへ渡す前にマニフェストを試す場所としても適しています。

日常の運用で特徴になる機能は二つあります。一つは、root権限を使わない実行です。カーネルのユーザー名前空間を使い、通常のユーザーをコンテナ内のUID 0へ、 `/etc/subuid` と `/etc/subgid` にあるそのユーザーの従属ID範囲を、ほかのすべてのUIDへ対応付けることで、一般ユーザーがコンテナを動かせます。root権限を使わないコンテナの通信は、ユーザー空間のネットワークスタックpastaを通ります。非特権ユーザーには、ホストのパケットフィルターを書き換える権利がないためです。

もう一つはQuadletです。Quadletファイルは、コンテナ、ボリューム、ネットワーク、pod、イメージのビルドを、短く宣言的に記述します。systemdユニットと同じini構文を使い、システム用なら `/etc/containers/systemd/` 、一人のユーザー用なら `~/.config/containers/systemd/` へ保存します。起動時と、 `systemctl`

`daemon-reload` の実行時に、systemdのジェネレーターが `web.container` を通常の `web.service` へ変換します。従来の `podman generate systemd` も動きますが、プロジェクトは非推奨とし、Quadletへ移るよう勧めています。

概念を判断に生かす

「Podmanとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 ノートパソコン、テストサーバー、本番環境で同じ動作になるようにまとめ、忙しい週には処理能力を増やせるようにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Google Cloud・AWSでDockerアプリ20個を運用 — 20 デプロイし監視したコンテナアプリケーション

Google Cloud・AWSでDockerアプリ20個を運用 お問い合わせ →

なぜ重要なのか

Podmanが問い直すのは、プロセスツリーを誰が管理するかです。Dockerは `dockerd` を介してコンテナを管理し、独自の再起動方針、ログ記録、コンテナの状態チェックを持ちます。すでにsystemdを動かしているサーバーでは、監督役がもう一つ増え、二つは同じ言葉で管理できません。

PodmanとQuadletの組み合わせでは、監督役はsystemdだけです。コンテナは、ほかと同じサービスになります。 `systemctl status` で状態を見て、journaldに出力を残し、 `After=` と `Requires=` で、必要なデータベースとの起動順と依存関係を定めます。systemdがすでに扱う `Restart=` とcgroupの制限も、そのまま適用します。ユニットファイルを読める運用者なら、コンテナの管理層も読めます。

セキュリティーの面もあります。Dockerデーモンは、rootlessモードを選ばない限りrootで動きます。Docker自身のセキュリティーガイドは、ホストの任意のディレクトリーをコンテナへ渡せるため、デーモンを操作できるのは信頼するユーザーだけにすべきだと説明しています。Podmanは、コマンドの間も特権デーモンを動かし続けることはありません。root権限を使わない構成なら、さらに影響を狭められます。コンテナから抜け出したプロセスは、非特権ユーザーの権限になります。

したがって、DockerかPodmanかは、機能の多さより、環境に合うかという選択です。Docker Desktopを使う開発者のノートパソコン、Composeを中心に作業するチーム、Dockerソケットを前提とするツールには、Dockerの方が使いやすい選択です。ただし、Podmanも、要求に応じて互換APIを提供できます。すでに

systemdをサービス管理の基準としているサーバーや、ファイアウォールを書き換えるデーモンが、便利さより負担になる環境では、Podmanが向くことが多くなります。

実際の活用例

最近の二つのプロジェクトで、Podmanがコンテナの管理層を担います。それ以前には、Dockerで構築した複数のシステムの問題を調べるツールとして使っていました。

あるホストでは、Dockerの代わりにPodmanを使い、コンテナをsystemd配下のQuadletユニットとして動かしています。理由はファイアウォールです。Dockerの既定のポート公開は、強化ロールが設ける規則より前に働きますが、これらのコンテナはホストのネットワークを使い、ポートを一つも公開しません。ユニットはrootlessではなく、システム単位で動かします。ページは、その選択を、自動化の単純さと、rootlessで得られるより強い分離との交換条件として記録し、コンテナからの脱出リスクの方が重要になった場合の移行先も示しています。デプロイとプロビジョニングは分けています。rootがデプロイ経路を一度整え、その後のリリースでは、非特権ユーザーが範囲を限定した規則で実行を許された、固定のスクリプトだけを動かします。スクリプトが行うのは、イメージのビルドとサービスの再起動の二つだけです。

同じ構成は、小さなハードウェアでも動きます。単一コア、メモリー512 MBのホストが、自社の公開TLSエンドポイント、Gitフォージ、onionサービスを動かし、バックアップの自動化は別に実装しています。コンテナの管理層は、独自の監督機能を持つデーモンではなく、ほかのすべてと同じ監督役の下にあるQuadletユニットです。記録された一日分のサンプルでは、OSが使える464 MBのメモリーのうち、利用可能と報告された量は約310 MBを下回りませんでした。

それ以前には、Google CloudとAWSに分散した二十個のDocker化アプリケーションで、Podmanを使って問題を調査していました。オーケストレーションにはKubernetesを使っていました。イメージは同じで、その周りのツールは選べました。

問題が起きるところ

Podmanの問題の多くは、Dockerと同じ動き、または通常のsystemdユニットと同じ動きを期待することから起きます。実際には、両方の性質を少しずつ持っています。

Quadletファイルを編集しても、変わりません。 ジェネレーターが動くのは、起動時と `systemctl daemon-reload` の実行時なので、どちらかが起きるまで、ディスク上の変更は反映されません。その後も、systemdが生成したサービスを一時的なものとして扱うため、 `systemctl enable` では有効にできません。起動時の開始は、ファイル自身の `[Install]` セクションに書きます。

```
# /etc/containers/systemd/web.container
[Unit]
Description=Web front end

[Container]
Image=docker.io/library/nginx:1.27
Network=host

[Service]
Restart=always

[Install]
WantedBy=multi-user.target
```

保存後に、 `systemctl daemon-reload` と `systemctl start web.service` を実行します。

rootlessサービスが、所有者のログアウトで止まります。 ユーザーユニットは、そのユーザー専用のsystemdマネージャー内にあり、最後のセッションが終わるとマネージャーも終了します。そのユーザーに `loginctl enable-linger` を実行すると、起動時からマネージャーを動かし続けられます。再起動後も動かしたいrootlessサービスには、これが必要です。

rootlessのウェブサーバーが、ポート80を要求して拒否されます。 1024未満のポートには、rootか `CAP_NET_BIND_SERVICE` ケーパビリティーが必要で、この規則はPodmanを動かすユーザーにも適用されます。高い番号のポートを公開して前にプロキシを置くか、ホスト上のすべてのユーザーに影響することを理解した上で、カーネルの `ip_unprivileged_port_start` を意図して下げます。

バインドマウント内のファイルが、見知らぬ所有者になります。 rootlessモードでは、ユーザー自身のUIDがコンテナ内のrootになり、ほかのコンテナUIDは、従属ID範囲へ対応付けられます。内部でUID 999として動くプロセスが書いたファイルは、ホストからは100998のような所有者に見えます。 `podman unshare` は、同じ対応付けの中でシェルを開くので、 `chown` がコンテナの期待どおりに動きます。マウントの `U` オプションは、コンテナに合うよう、自動的に所有者を変更します。

参考資料

- Podmanとは — ツールの定義から始まる、プロジェクト自身のドキュメントです。
- podman-systemd.unit — ファイルの種類、検索経路、すべてのキーを扱う、Quadletのリファレンスです。
- rootless環境でのPodmanの基本設定と利用 — 従属ID、ユーザー名前空間、pastaによる通信です。
- podman-unshare — rootlessのユーザー名前空間に入り、所有者を調べたり直したりする方法です。
- podman-generate-systemd — ユニットを作る従来の方法と、それを非推奨としてQuadletを勧める説明です。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「Podmanとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/podman/