Engineer

systemd

Linuxの起動と各サービスの開始を担う仕組み

どのようなものか

主要なLinuxディストリビューションのほぼすべてで、プロセスID 1はsystemdです。起動時にマシンを立ち上げ、稼働中はサービスを開始、監視し、終了時にはシステム全体を順序立てて停止する、システムとサービスのマネージャーです。プロジェクト名は小文字で書き、SystemDやSystem Dとはしません。末尾のdは、daemonを示す昔からのUnixの表記です。多くのディストリビューションで置き換えたSysV initスクリプトやUpstart、小規模なディストリビューションの一部が今も選ぶOpenRCやrunit、macOSで同じ役割を果たすlaunchdと並ぶ、initシステムの仲間です。

systemdが管理するものはすべてユニットで、セクションとキー・値の組からなる短いテキストファイルで記述します。マニュアルには、service、socket、target、device、mount、automount、timer、swap、path、slice、scopeの11種類があります。サービスはプログラムを動かし、タイマーはスケジュールに従って別のユニットを開始します。ソケットは待ち受けポートを保持し、最初の接続が来るとサービスを起動します。ターゲットはユニットをまとめ、ネットワークが使えるマルチユーザーシステムなど、マシンが到達できる状態を表します。依存関係と順序は別設定なので、「これにはあれが必要」と「これはあれの後に開始する」は別の二行です。systemdはそれらを、実行前に確認できる開始・停止ジョブの一つのトランザクションへまとめます。

以前のシェルスクリプト式initシステムと異なる設計上の選択が二つあります。各サービスは独自のLinuxコントロールグループ内で動くため、マネージャーはグループのプロセスを追跡し、まとめて止め、メモリーを制限し、CPU使用量を集計できます。また、プロジェクトはマネージャーだけでなく、ツール群でもあります。公式サイトには、ログ用デーモンであるjournalに加え、ログイン、ネットワーク設定、時刻同期、名前解決のコンポーネントが挙げられています。追加機能のどれを有効にするかはディストリビューションが選びます。ほぼ共通して使われるのは、PID 1のマネージャー部分です。

概念を判断に生かす

「systemd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。 サーバー、メール、アカウント、更新を日々管理し、誰もが頼る仕組みを平穏に動かし続けます。

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。 UbuntuでPostgreSQL・MS SQLサーバー10台を運用 — 10 導入し正常に保ったDBサーバー

UbuntuでPostgreSQL・MS SQLサーバー10台を運用 お問い合わせ →

なぜ重要なのか

Linuxサーバーでsystemdを使うかどうかが問題になることはまれです。ディストリビューションがすでに選んでいるためです。実際の判断は、監視とスケジュール管理にsystemdを使うか、その上に別のプロセスマネージャー、コンテナーデーモンの再起動ポリシー、cronを重ねるかです。

直接使えば、起動順序、再起動ポリシー、ログ、リソース制限、サンドボックス設定をサービスごとに一つのファイルへ集め、すでに動いているマネージャーが読みます。二つ目の監視プロセスは不要です。ユニットは `MemoryMax=` でメモリーを制限し、 `ProtectSystem=` でOS自身のディレクトリをサービスから読み取り専用にし、 `DynamicUser=` で一時的なユーザーとして実行できます。 `systemd-analyze security` は、各サービスの露出度を0.0から10.0で評価します。タイマーで起動したジョブは、crontabの一行とは異なり、journalへログを記録し、通常のサービスと同じサンドボックス設定を使い、 `systemctl list-timers` で前回と次回の実行を表示します。この知識は、ほかの環境でも役立ちます。ユニットファイルを読めるエンジニアなら、ディストリビューションを問わず、多くのLinuxサーバーを運用できます。

負担もあります。ユニットはLinux専用なので、macOSにも提供するチームはlaunchdのプロパティリストを、BSDにも提供するチームはrcスクリプトを別に書きます。大半のコンテナー内にはsystemd自体がなく、アプリケーションがPID 1となり、再起動ポリシーはコンテナーを実行する側が担います。コンテナーと通常のサービスがホストを共有するとき、PodmanのQuadletは、コンテナーの記述を普通のサービスユニットへ変換して差を縮めます。両方のワークロードを一つの監視機構の下に置けます。

実際の活用例

自社のプラットフォームでは、三か所でsystemdを活用しています。

2026年9月の実装記録では、企業の公開サービスを512 MBのホスト一台で動かす実績の本番ホストは、利用可能なメモリーが約464 MBのシングルコアマシンでした。七つのドメインのTLSを担うウェブサーバー、Gitフォージ、Onionサービス、Gemini用とGopher用のサーバー各一つ、侵入元の遮断機能が常駐していました。バックアップの自動化は別に実装されており、暗号化バックアップが稼働中だと証明するものではありません。その規模では、常駐する各プロセスに、それだけのメモリーを使う理由が必要でした。そこでsystemdを監視機構とし、その上に別の監視プロセスを重ねませんでした。コンテナーも独自のデーモンではなく、同じマネージャー配下のQuadletユニットとして動かしました。さらにOnionサービスも載せられるかを検討した際、記録した一日分の測定サンプルでは、464 MB中の利用可能メモリーが約310 MBを下回ることはありませんでした。

バックアップと復元テストの実装では、設定がある場合のデータベースダンプ、別事業者のストレージへのSFTP経由の暗号化・重複排除resticスナップショット、保持方針に従う削除、整合性検査にsystemdユニットを使います。別の月次ワークフローが既知のファイルを復元してバイト列を比較し、バックアップと復元は別々の成功シグナルを持ちます。タイマーにはリポジトリ設定が必要です。2026-10-27付の実装記録は、リポジトリ未設定のためタイマーが無効だったと述べており、実際の復元を完了した記録もありません。この作業では、後述するsystemdの落とし穴にも遭遇しました。パスフレーズはユニットが読む保護されたファイルに置きます。ユニットの環境を通じて渡す値はエスケープ処理されるため、バックスラッシュで値が知らないうちに変わる可能性があったからです。

技術構成は、通常のsystemdユニットを既定とし、DockerやKubernetesは必要性に見合うワークロードに限るという、全体の方針を示しています。

問題が起きるところ

ユニットを編集しても、古い設定で動き続ける。 systemdはユニットファイルをメモリーへ読み込むため、ディスク上で変えても、`systemctl daemon-reload` でマネージャーの認識を更新するまでは反映されません。`systemctl cat` が表示するファイルも、動作中のマネージャーが使う内容と異なることがあります。自動で再読み込みする `systemctl edit` で変更するか、再読み込みの後にサービスを再起動します。

シークレットが、一文字欠けた状態でサービスへ届く。 `Environment=` の値はsystemd独自の引用規則を通り、C形式のエスケープが処理されるため、文字としてのバックスラッシュは二重にする必要があります。公式ドキュメントは、環

境変数がD-Bus経由で非特権クライアントに見えることも警告しています。シークレットはファイルに置き、`LoadCredential=`で渡します。サービスには、そのユーザーとrootだけが読める読み取り専用コピーが与えられます。

パッケージ更新が、ローカルの修正を黙って取り消す。 `/usr/lib/systemd/system/`内にある配布元のファイルを編集すると、次のパッケージ更新で上書きされます。変更は`/etc/systemd/system/<unit>.d/`内のdrop-inに置きます。元の設定に重ねて適用され、更新後も残ります。コマンドのリストは置き換わらず累積するため、開始コマンドを変えるdrop-inは、まず空の代入で消去します。

```
# /etc/systemd/system/app.service.d/override.conf
[Service]
ExecStart=
ExecStart=/usr/local/bin/app --config /etc/app/app.toml
Restart=on-failure
```

サーバーが停止していた夜の定期ジョブが実行されない。 マシン停止中に予定時刻を迎えたカレンダータイマーは、`Persistent=`の既定値がfalseなので、そのまま次回を待ちます。バックアップなど、実行が必要なジョブには`Persistent=true`を設定します。タイマーが再び有効になると、設定済みのランダムな遅延があればそれに従って、逃した実行を行います。

異常終了したサービスが、停止したままになる。 `Restart=`の既定値は`no`なので、終了したサービスは誰かが気付くまで止まったままです。長時間動くサービスには`Restart=on-failure`を設定します。ただし起動回数の制限に達すると、マネージャーは再試行をやめます。手動起動も、制限期間が過ぎるか、`systemctl reset-failed`を実行するまで待つ必要があります。

参考資料

- systemd(1) — ユニットの種類、起動順序、応答するシグナルなど、マネージャー自体を説明しています。
- systemd.unit(5) — ユニットファイル、依存関係、起動制限、設定を上書きするdrop-inディレクトリを説明しています。
- systemd.service(5) — サービスの種類、再起動ポリシー、コマンドリストのリセット方法を説明しています。
- systemd.exec(5) — 実行環境、サンドボックス設定、資格情報を説明しています。
- systemd.timer(5) — `Persistent=`を含む、カレンダータイマーと単調時計のタイマーを説明しています。

- systemctl(1) — ユニットの開始、編集、再読み込み、調査に使うコマンドラインを説明しています。

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

「systemd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/systemd/