Red Hat Enterprise Linux 8

systemd ユニットファイルを使用したシステムのカスタマイズおよび最適化

systemd を使用したシステムパフォーマンスの最適化および設定の拡張

Last Updated: 2024-03-21

Red Hat Enterprise Linux 8 systemd ユニットファイルを使用したシステムのカスタマイズおよび最適化

systemd を使用したシステムパフォーマンスの最適化および設定の拡張

法律上の通知

概要

systemd ユニットファイルを変更してデフォルト設定を拡張し、システムの起動パフォーマンスを確認し、systemd を最適化して起動時間を短縮します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに関するご意見やご感想をお寄せください。また、改善点があればお知らせください。

Jira からのフィードバック送信 (アカウントが必要)

1. Jira の Web サイトにログインします。
2. 上部のナビゲーションバーで **Create** をクリックします。
3. **Summary** フィールドにわかりやすいタイトルを入力します。
4. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
5. ダイアログの下部にある **Create** をクリックします。

第1章 SYSTEMD ユニットファイルでの作業

systemd ユニットファイルはシステムリソースを表します。システム管理者は、次の高度なタスクを実行できます。

- カスタムユニットファイルの作成
- 既存のユニットファイルの変更
- インスタンス化されたユニットの使用

1.1. ユニットファイルの概要

ユニットファイルには、ユニットを説明し、その動作を定義する設定ディレクティブが含まれます。複数の **systemctl** コマンドがバックグラウンドでユニットファイルと連携します。より細かく調整するには、ユニットファイルを手動で編集または作成します。システムにユニットファイルが保存される 3 つのメインディレクトリーが存在します。**/etc/systemd/system/** ディレクトリーは、システム管理者が作成またはカスタマイズするユニットファイル用に予約されます。

ユニットファイル名は、以下のフォーマットを使用します。

```
<unit_name>.<type_extension>
```

unit_name はユニットの名前を表し、 type_extension はユニットの種類を表します。

たとえば、システム上には **sshd.service** と **sshd.socket** ユニットが存在します。

ユニットファイルには、追加の設定ファイルのディレクトリーを追加できます。たとえば、カスタム設定オプションを **sshd.service** に追加するには、**sshd.service.d/custom.conf** ファイルを作成し、追加のディレクティブを挿入します。設定ディレクトリーの詳細は、既存のユニットファイルの変更 を参照してください。

systemd システムおよびサービスマネージャーは、**sshd.service.wants/** ディレクトリーと **sshd.service.requires/** ディレクトリーを作成することもできます。このディレクトリーには、**sshd** サービスの依存関係であるユニットファイルへのシンボリックリンクが含まれます。**systemd** は、シンボリックリンクを、[Install] ユニットファイルに基づいてインストール時に、または [Unit] オプションに基づいてランタイム時に自動的に作成します。これらのディレクトリーとシンボリックリンクを手動で作成することもできます。

さらに、**sshd.service.wants/** ディレクトリーおよび **sshd.service.requires/** ディレクトリーを作成することもできます。このディレクトリーには、**sshd** サービスの依存関係であるユニットファイルへのシンボリックリンクが含まれます。シンボリックリンクは、[Install] ユニットファイルに基づいてインストール時に、または [Unit] オプションに基づいてランタイム時に自動的に作成されます。このディレクトリーとシンボリックリンクを手動で作成することもできます。[Install] オプションおよび [Unit] オプションの詳細は、以下の表を参照してください。

多くのユニットファイルオプションは、いわゆる **ユニット指定子** を使用して設定できます。これは、ユニットファイルが読み込まれる際にユニットパラメーターに動的に置き換えられるワイルドカード文字列です。これにより、インスタンス化されたユニットを生成するテンプレートとしてのロールを担う汎用ユニットファイルを作成できます。インスタンス化されたユニットの使用 を参照してください。

1.2. SYSTEMD のユニットファイルの場所

ユニット設定ファイルは、次のいずれかのディレクトリーにあります。

表1.1 systemd のユニットファイルの場所

ディレクトリー	説明
/usr/lib/systemd/system/	インストール済みの RPM パッケージで配布された **systemd** のユニットファイル。
/run/systemd/system/	ランタイム時に作成された **systemd** ユニットファイル。このディレクトリーは、インストール済みのサービスのユニットファイルのディレクトリーよりも優先されます。
/etc/systemd/system/	**systemctl enable** コマンドを使用して作成された **systemd** ユニットファイルと、サービスを拡張するために追加されたユニットファイル。このディレクトリーは、runtime のユニットファイルのディレクトリーよりも優先されます。

systemd のデフォルト設定はコンパイル中に定義され、**/etc/systemd/system.conf** ファイルで確認できます。このファイルを編集すると、**systemd** ユニットの値をシステム全体でオーバーライドしてデフォルト設定を変更できます。

たとえば、タイムアウト制限のデフォルト値 (90 秒) を上書きする場合は、**DefaultTimeoutStartSec** パラメーターを使用して、上書きする値を秒単位で入力します。

```
DefaultTimeoutStartSec=required value
```

1.3. ユニットファイル構造

ユニットファイルは通常、次の 3 つのセクションから構成されます。

[Unit] セクション

ユニットのタイプに依存しない汎用的なオプションが含まれます。このセクションに含まれるオプションはユニットを説明し、ユニットの動作を指定し、他のユニットへの依存関係を設定します。最も頻繁に使用される [Unit] オプションのリストは、重要な [Unit] セクションのオプション を参照してください。

[Unit type] セクション

タイプ固有のディレクティブが含まれます。これらのディレクティブは、ユニットタイプにちなんで命名されたセクションにグループ化されます。たとえば、サービスユニットファイルには **[Service]** セクションが含まれます。

[Install] セクション

systemctl enable および **disable** コマンドで使用されるユニットのインストールに関する情報が含まれます。**[Install]** セクションのオプションリストは、Important [Install] セクションのオプション を参照してください。

関連情報

- [Unit] セクションの重要なオプション
- [Service] セクションの重要なオプション

- [Install] セクションの重要なオプション

1.4. [UNIT] セクションの重要なオプション

以下の表は、[Unit] セクションの重要なオプションを示しています。

表1.2 [Unit] セクションの重要なオプション

オプション [a]	説明
説明	ユニットの説明です。このテキストは、たとえば **systemctl status** コマンドの出力に表示されます。
Documentation	ユニットのドキュメントを参照する URI のリストを提供します。
After[b]	ユニットが開始する順序を定義します。このユニットは、**After** で指定されたユニットがアクティブになると開始します。**Requires** とは異なり、**After** は、指定したユニットを明示的にアクティブにしません。**Before** オプションには、**After** と機能が反対になります。
Requires	その他のユニットに依存関係を設定します。**Requires** にリスト表示されるユニットは、対応するユニットと共にアクティブになります。必要なユニットのいずれかが開始しないと、このユニットはアクティブになりません。
Wants	**Requires** よりも強度の弱い依存関係を設定します。リストに示されるユニットのいずれかが正常に開始しなくても、このユニットのアクティベーションには影響を与えません。これは、カスタムのユニット依存関係を設定する際に推奨される方法です。
Conflicts	**Requires** と反対の依存関係 (負の依存関係) を設定します。

[a] [Unit] セクションで設定可能なオプションのリストは、**systemd.unit(5)** の man ページを参照してください。

[b] ほとんどの場合、ユニットファイルオプションの **After** および **Before** で依存関係の並び順を設定するだけで十分です。**Wants** (推奨) または **Requires** で要件の依存関係も設定する場合は、依存関係の並び順を指定する必要があります。これは、並び順と要件の依存関係が相互に依存していないためです。

1.5. [SERVICE] セクションの重要なオプション

以下の表では、[Service] セクションの重要なオプションを紹介しています。

表1.3 [Service] セクションの重要なオプション

オプション [a]	説明
Type	**ExecStart** および関連オプションの機能に影響を与えるユニットプロセスの起動タイプを設定します。以下のいずれかになります。 * **simple** - デフォルト値です。**ExecStart** で起動するプロセスは、サービスのメインプロセスです。 * **forking** - **ExecStart** で起動するプロセスは、サービスのメインプロセスになる子プロセスを起動します。親プロセスは、このプロセスが完了すると終了します。 * **oneshot** - このタイプは **simple** と似ていますが、結果として生じるユニットを起動する前に終了します。 * **dbus** - このタイプは **simple** と似ていますが、メインプロセスが D-Bus 名を取得する前に、結果として生じるユニットが起動します。 * **notify** - このタイプは **simple** と似ていますが、結果として生じるユニットは、通知メッセージが sd_notify() 関数で送信されないと起動しません。 * **idle** - **simple** と似ていますが、サービスバイナリーの実行は、すべてのジョブが終了するまで行いません (遅らせます)。 これにより、ステータスの出力とサービスのシェル出力を分けることができます。
ExecStart	ユニットの開始時に実行するコマンドまたはスクリプトを指定します。**ExecStartPre** および **ExecStartPost** は、**ExecStart** の前後に実行するカスタムコマンドを指定します。**Type=oneshot** を使用すれば、連続して実行する複数のカスタムコマンドを指定できます。
ExecStop	ユニットの停止時に実行するコマンドまたはスクリプトを指定します。
ExecReload	ユニットの再読み込み時に実行するコマンドまたはスクリプトを指定します。
Restart	このオプションを有効にすると、**systemctl** コマンドによる完全な停止の例外により、そのプロセスの終了後にサービスが再起動します。

オプション [a]	説明
RemainAfterExit	True に設定すると、サービスは、そのプロセスがすべて終了していてもアクティブと見なされます。デフォルトの値は False です。このオプションは、特に **Type=oneshot** が設定されている場合に役に立ちます。

[a] [Service] セクションで設定可能なオプションのリストは、**systemd.service (5)** の man ページを参照してください。

1.6. [INSTALL] セクションの重要なオプション

以下の表は、[Install] セクションの重要なオプションを紹介しています。

表1.4 [Install] セクションの重要なオプション

オプション [a]	説明
Alias	ユニット名の追加リストを、スペース区切りで提供します。**systemctl enable** を除くほとんどの **systemctl** コマンドでは、ユニット名ではなくエイリアスを使用できます。
RequiredBy	そのユニットに依存するユニットのリストです。このユニットが有効な場合に、**RequiredBy** にリスト表示されるユニットは、このユニットに関する **Require** 依存関係を取得します。
WantedBy	このユニットへの依存が弱いユニットのリストです。このユニットが有効になると、**WantedBy** にリスト表示されるユニットが、このユニットに関する **Want** 依存関係を取得します。
Also	対応するユニットと共にインストールまたはアンインストールされるユニットのリストを指定します。
DefaultInstance	インスタンス化されているユニットだけが対象となりますが、このオプションは、ユニットが有効になっているデフォルトのインスタンスを指定します。インスタンス化されたユニットの使用 を参照してください。

[a] [Install] セクションで設定可能なオプションのリストは、**systemd.unit (5)** の man ページを参照してください。

1.7. カスタムユニットファイルの作成

最初からユニットファイルを作成するユースケースはいくつかあります。カスタムデーモンを実行し、sshd サービスの 2 番目のインスタンスを使用したカスタムユニットファイルの作成 のように、既存サービスの 2 番目のインスタンスを作成できます。

一方、既存ユニットの動作の変更または拡張のみを実行する場合は、既存のユニットファイルの変更 の手順を使用してください。

手順

1. カスタムサービスを作成するには、サービスを含む実行ファイルを準備します。ファイルには、カスタムで作成されたスクリプト、またはソフトウェアプロバイダーによって提供された実行ファイルを含めることができます。必要な場合は、カスタムサービスのメインプロセスの PID を保持するため、PID ファイルを用意します。サービスのシェル変数を保存する環境ファイルを含めることもできます。(**chmod a+x** を実行して) ソーススクリプトを実行でき、インタラクティブではないことを確認してください。

2. **/etc/systemd/system/** ディレクトリーにユニットファイルを作成し、ファイルに適切なパーミッションがあることを確認します。**root** で以下のコマンドを実行します。

```
# touch /etc/systemd/system/<name>.service

# chmod 664 /etc/systemd/system/<name>.service
```

   <name> は、作成するサービスの名前に置き換えます。ファイルは実行可能である必要はないことに注意してください。

3. 作成した **<name>.service** ファイルを開き、サービス設定オプションを追加します。作成するサービスのタイプに応じてさまざまなオプションを使用できます。ユニットファイル構造 を参照してください。
   以下は、ネットワーク関連サービスのユニットの設定例になります。

```
[Unit]
Description=<service_description>
After=network.target

[Service]
ExecStart=<path_to_executable>
Type=forking
PIDFile=<path_to_pidfile>

[Install]
WantedBy=default.target
```

   - <service_description> は、ジャーナルログファイルおよび **systemctl status** コマンドの出力に表示される有用な説明です。
   - **After** 設定により、このサービスは、ネットワークの実行後にのみ開始されます。関連するサービスまたはターゲットは、スペースで区切って追加します。
   - path_to_executable は、サービス実行ファイルへのパスを表します。
   - **Type=forking** は、fork システム呼び出しを行うデーモンに使用します。サービスのメインプロセスは、**path_to_pidfile** で指定した PID で作成されます。 重要な [Service] セクションのオプション で別の起動タイプを検索できます。

- **WantedBy** では、サービスを開始する必要がある 1 つ以上のターゲットを指定します。ターゲットは、従来のランレベルの概念に代わるものとお考えください。

4. 新しい **<name>.service** ファイルが存在することを **systemd** に通知します。

```
# systemctl daemon-reload

# systemctl start <name>.service
```

警告

新しいユニットファイルを作成したり、既存のユニットファイルを修正したら常に **systemctl daemon-reload** コマンドを実行します。このコマンドを実行しないと、**systemd** のステータスと、ディスクの実際のサービスユニットファイルのステータスが一致しなくなるため、**systemctl start** コマンドや **systemctl enable** コマンドが失敗する可能性があります。ユニット数が多いシステムでは、各ユニットのステータスをシリアライズし、その後再読み込み時にデシリアライズする必要があるため、これには時間がかかることがあります。

1.8. SSHD サービスの 2 番目のインスタンスを使用したカスタムユニットファイルの作成

サービスの複数のインスタンスを設定して実行する必要がある場合は、元のサービス設定ファイルのコピーを作成し、特定のパラメーターを変更して、サービスのプライマリーインスタンスとの競合を回避できます。

手順

sshd サービスの 2 つ目のインスタンスを作成するには、次の手順を実行します。

1. 2 つ目のデーモンが使用する **sshd_config** ファイルのコピーを作成します。

```
# cp /etc/ssh/sshd{,-second}_config
```

2. 作成した **sshd-second_config** ファイルを編集し、2 つ目のデーモンに別のポート番号と PID ファイルを割り当てます。

```
Port 22220
PidFile /var/run/sshd-second.pid
```

Port オプションおよび **PidFile** オプションの詳細は、man ページの **sshd_config**(5) を参照してください。他のサービスで使用されていないポートを選択してください。PID ファイルはサービスの実行時に存在していなければいけないものではありません。存在しない場合は、サービスの起動時に自動的に生成されます。

3. **sshd** サービスの **systemd** ユニットファイルのコピーを作成します。

```
# cp /usr/lib/systemd/system/sshd.service /etc/systemd/system/sshd-second.service
```

4. 作成した **sshd-second.service** を変更します。

   a. **Description** オプションを変更します。

   ```
   Description=OpenSSH server second instance daemon
   ```

   b. **After** オプションを指定するサービスに **sshd.service** を追加し、最初のインスタンスが起動した場合に限り 2 つ目のインスタンスが起動するようにします。

   ```
   After=syslog.target network.target auditd.service sshd.service
   ```

   c. **ExecStartPre=/usr/sbin/sshd-keygen** 行を削除します。鍵の生成は **sshd** の最初のインスタンスに含まれます。

   d. **sshd** コマンドに **-f /etc/ssh/sshd-second_config** パラメーターを追加して、代替の設定ファイルが使用されるようにします。

   ```
   ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config $OPTIONS
   ```

   e. 変更後、**sshd-second.service** ユニットファイルには次の設定が含まれます。

   ```
   [Unit]
   Description=OpenSSH server second instance daemon
   After=syslog.target network.target auditd.service sshd.service

   [Service]
   EnvironmentFile=/etc/sysconfig/sshd
   ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config $OPTIONS
   ExecReload=/bin/kill -HUP $MAINPID
   KillMode=process
   Restart=on-failure
   RestartSec=42s

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

5. SELinux を使用している場合は、**sshd** の 2 番目のインスタンスのポートを SSH ポートに追加します。追加しないと、**sshd** の 2 番目のインスタンスがポートにバインドされません。

   ```
   # semanage port -a -t ssh_port_t -p tcp 22220
   ```

6. **sshd-second.service** を有効にして、起動時に自動的に開始するようにします。

   ```
   # systemctl enable sshd-second.service
   ```

7. **systemctl status** コマンドを使用して **sshd-second.service** が実行中かどうかを確認します。

8. さらに、サービスに接続して、ポートが正しく有効化されていることを確認します。

   ```
   $ ssh -p 22220 user@server
   ```

   sshd の 2 番目のインスタンスへの接続を許可するようにファイアウォールを設定します。

1.9. SYSTEMD サービスの説明の検索

#description で始まる行で、スクリプトに関する説明の情報を確認します。この説明は、サービス名と共に、ユニットファイルの [Unit] セクションの **Description** オプションで使用します。ヘッダーの **#Short-Description** 行および **#Description** 行に同様のデータが含まれる場合があります。

1.10. SYSTEMD サービス依存関係の検索

Linux standard base (LSB) ヘッダーには、サービス間の依存関係を形成するいくつかのディレクティブが含まれる場合があります。そのほとんどは、systemd ユニットオプションに変換できます。以下の表を参照してください。

表1.5 LSB ヘッダーの依存関係オプション

LSB オプション	説明	同等のユニットファイル
Provides	サービスの起動ファシリティー名を指定します。この名前は他の init スクリプトで参照できます ("$" 接頭辞を使用)。ただし、ユニットファイルが他のユニットをファイル名で参照できるようになったため、これは不要になりました。	–
Required-Start	必要なサービスの起動ファシリティー名が含まれます。これは、並び順の依存関係として変換され、起動ファシリティー名は、対応するサービスまたはそのサービスが属するターゲットに置き換えられます。たとえば、**postfix** の場合、$network の Required-Start 依存関係は、network.target の After 依存関係に変換されました。	**After**、**Before**
Should-Start	Required-Start よりも弱い依存関係を設定します。Should-Start 依存関係が失敗しても、サービスの起動には影響を及ぼしません。	**After**、**Before**
Required-Stop、**Should-Stop**	負の依存関係を設定します。	**Conflicts**

1.11. サービスのデフォルトターゲットの検索

#chkconfig で始まる行には 3 つの数値があります。最も重要な値は最初の数値で、サービスが起動するデフォルトのランレベルを示しています。ランレベルは、同等の systemd ターゲットに対応します。次に、これらのターゲットを、ユニットファイルの [Install] セクションの **WantedBy** オプションに記述します。たとえば、**postfix** がランレベルの 2、3、4、および 5 で起動していた場合、これは multi-user.target および graphical.target に対応します。ただし、graphical.target は multiuser.target に依存

するため、両方を記述する必要はありません。また、LSB ヘッダーの **#Default-Start** および **#Default-Stop** 行に、デフォルト、および動作するべきでないランレベルの情報がある場合は、そちらも参照してください。

#chkconfig 行で指定した他の 2 つの値は、init スクリプトの起動およびシャットダウンの優先順位を表します。この値は、init スクリプトが読み込まれる場合は **systemd** により解釈されますが、同等のユニットファイルはありません。

1.12. サービスで使用されるファイルの検索

init スクリプトでは、専用ディレクトリーから関数ライブラリーを読み込み、設定ファイル、環境ファイル、および PID ファイルのインポートを許可します。環境変数は init スクリプトヘッダーの **#config** で始まる行で指定され、これは、**EnvironmentFile** ユニットファイルオプションに変換されます。**#pidfile** init スクリプト行に指定した PID ファイルは、 **PIDFile** オプションでユニットファイルにインポートされます。

init スクリプトヘッダーに含まれない主要な情報は、サービス実行ファイルへのパス、またはサービスで必要になる可能性のあるその他のファイルへのパスです。以前のバージョンの Red Hat Enterprise Linux では、init スクリプトは、カスタム定義のアクションと共に **起動**、**停止**、**再起動** などのデフォルトアクションのサービスの動作を定義する Bash ケーステートメントを使用しました。**postfix** init スクリプトからの以下の抜粋は、サービス起動時に実行するコードのブロックを示しています。

```
conf_check() {
    [ -x /usr/sbin/postfix ] || exit 5
    [ -d /etc/postfix ] || exit 6
    [ -d /var/spool/postfix ] || exit 5
}

make_aliasesdb() {
 if [ "$(/usr/sbin/postconf -h alias_database)" == "hash:/etc/aliases" ]
 then
  # /etc/aliases.db might be used by other MTA, make sure nothing
  # has touched it since our last newaliases call
  [ /etc/aliases -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -ot /etc/aliases.db ] || return
  /usr/bin/newaliases
  touch -r /etc/aliases.db "$ALIASESDB_STAMP"
 else
  /usr/bin/newaliases
 fi
}

start() {
 [ "$EUID" != "0" ] && exit 4
 # Check that networking is up.
 [ ${NETWORKING} = "no" ] && exit 1
 conf_check
 # Start daemons.
 echo -n $"Starting postfix: "
 make_aliasesdb >/dev/null 2>&1
 [ -x $CHROOT_UPDATE ] && $CHROOT_UPDATE
 /usr/sbin/postfix start 2>/dev/null 1>&2 && success || failure $"$prog start"
 RETVAL=$?
```

```
 [ $RETVAL -eq 0 ] && touch $lockfile
        echo
 return $RETVAL
}
```

init スクリプトの拡張性により、**start()** 関数ブロックから呼び出される **conf_check()** および **make_aliasesdb()** の 2 つのカスタム関数を指定することができました。さらに詳しくみると、上記コードでは外部のファイルおよびディレクトリーが複数記述されています (主なサービス実行ファイル **/usr/sbin/postfix**、設定ディレクトリー **/etc/postfix/**、**/var/spool/postfix/**、および **/usr/sbin/postconf/** ディレクトリー)。

systemd は、事前に定義されたアクションのみをサポートしますが、オプションの **ExecStart**、**ExecStartPre**、**ExecStartPost**、**ExecStop**、**ExecReload** でカスタムの実行ファイルを有効にできます。**/usr/sbin/postfix** は、対応するスクリプトとともに、サービスの起動時に実行します。複雑な init スクリプトを変換する際には、スクリプトのすべてのステートメントの目的を理解している必要があります。一部のステートメントはオペレーティングシステムのバージョンに固有のものであるため、そのステートメントを変換する必要はありません。一方、新規の環境では、サービス実行ファイルおよびサポートファイルやユニットファイルで調整が一部必要となる場合があります。

1.13. 既存のユニットファイルの変更

既存のユニットファイルを変更する場合は、**/etc/systemd/system/** ディレクトリーに進みます。システムが **/usr/lib/systemd/system/** ディレクトリーに保存しているデフォルトのユニットファイルは変更しないでください。

手順

1. 必要とされる変更の程度に応じて、以下の方法のいずれかを実施してください。

   - 補助設定ファイルのディレクトリーを **/etc/systemd/system/<unit>.d/** に作成します。この方法は、ほとんどのユースケースで推奨されます。元のユニットファイルを参照しつつも、デフォルト設定を追加の機能で拡張できます。この場合、パッケージのアップグレードで導入されるデフォルトユニットへの変更は自動的に適用されます。詳細は、デフォルトのユニット設定の拡張 を参照してください。

   - **/usr/lib/systemd/system/ ディレクトリーの元のユニットファイルのコピーを `/etc/systemd/system/** ディレクトリーに作成し、そこで変更を加えます。コピーは元のファイルを上書きするため、パッケージの更新で導入される変更は適用されません。この方法は、パッケージの更新とは無関係に永続する重要なユニット変更を行う際に役に立ちます。詳細は、デフォルトのユニット設定のオーバーライド を参照してください。

2. ユニットのデフォルト設定に戻るには、**/etc/systemd/system/** でカスタム作成した設定ファイルを削除します。

3. システムを再起動せずにユニットファイルに変更を適用します。

   ```
   # systemctl daemon-reload
   ```

   daemon-reload オプションは、すべてのユニットファイルを再読み込みし、ユニットファイルへの変更をすぐに適用するのに必要な依存関係ツリー全体を再作成します。別の方法として、次のコマンドで同じ結果を得ることができます。

   ```
   # init q
   ```

4. 変更されたユニットファイルが実行中のサービスに属している場合は、サービスを再起動します。

```
# systemctl restart <name>.service
```

重要

SysV init スクリプトが処理しているサービスのプロパティー (依存関係やタイムアウトなど) を変更するときは、init スクリプト自体は変更しないでください。代わりに、デフォルトのユニット設定の拡張 と デフォルトのユニット設定のオーバーライド にあるように、サービスの **systemd** ドロップイン設定ファイルを作成します。

その後、通常の **systemd** サービスと同じ方法でサービスを管理します。

たとえば、**network** サービスの設定を拡張するときは、init スクリプトファイル **/etc/rc.d/init.d/network** を変更しないでください。代わりに、新しいディレクトリー **/etc/systemd/system/network.service.d/** と、**systemd** ドロップインファイル **/etc/systemd/system/network.service.d/my_config.conf** を作成します。そして、ドロップインファイルの値を変更します。**systemd** は、**network** サービスを **network.service** として認識することに注意してください。作成したディレクトリーが **network.service.d** と命名されるのはそのためです。

1.14. デフォルトのユニット設定の拡張

追加の systemd 設定オプションを使用して、デフォルトのユニットファイルを拡張できます。

手順

1. **/etc/systemd/system/** に設定ディレクトリーを作成します。

```
# mkdir /etc/systemd/system/<name>.service.d/
```

<name> を、拡張するサービスの名前に置き換えます。この構文はすべてのユニットタイプに適用されます。

2. **.conf** 接尾辞が付いた設定ファイルを作成します。

```
# touch /etc/systemd/system/name.service.d/<config_name>.conf
```

<config_name> を、設定ファイルの名前に置き換えます。このファイルは、通常のユニットファイル構造に基づくため、すべてのディレクティブは該当するセクションで指定する必要があります。ユニットファイル構造 を参照してください。

たとえば、カスタムの依存性を追加するには、以下の内容で設定ファイルを作成します。

```
[Unit]
Requires=<new_dependency>
After=<new_dependency>
```

<new_dependency> は、依存関係としてマークされるユニットを表します。次の例は、30 秒の遅延後のメインプロセス終了後にサービスを再起動する設定ファイルです。

```
[Service]
Restart=always
RestartSec=30
```

1 つのタスクだけを扱う簡単な設定ファイルを作成します。これにより、他のサービスの設定ディレクトリーに簡単に移動したり、リンクできます。

3. 変更をユニットに適用します。

```
# systemctl daemon-reload
# systemctl restart <name>.service
```

例1.1 httpd.service 設定の拡張

Apache サービスの起動時にカスタムシェルスクリプトが自動的に実行されるように **httpd.service** ユニットを変更するには、以下の手順を実行します。

1. ディレクトリーおよびカスタム設定ファイルを作成します。

```
# mkdir /etc/systemd/system/httpd.service.d/
```

```
# touch /etc/systemd/system/httpd.service.d/custom_script.conf
```

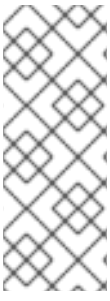
2. 次のテキストを **custom_script.conf** ファイルに挿入して、メインサービスプロセスの後に実行するスクリプトを指定します。

```
[Service]
ExecStartPost=/usr/local/bin/custom.sh
```

3. ユニットの変更を適用します。

```
# systemctl daemon-reload
```

```
# systemctl restart httpd.service
```



注記

/etc/systemd/system/ の設定ディレクトリーの設定ファイルは、**/usr/lib/systemd/system/** のユニットファイルに優先します。そのため、設定ファイルに、一度だけ指定できるオプション (**Description**、**ExecStart** など) が含まれる場合は、このオプションのデフォルト値が上書きされます。上書きされたユニットの監視 で説明されているように、**systemd-delta** コマンドの出力では、一部のオプションは実際に上書きされますが、該当するユニットは常に [EXTENDED] とマークされます。

1.15. デフォルトのユニット設定の上書き

ユニットファイル設定に変更を加え、ユニットファイルを提供するパッケージの更新後も変更が維持されるようにできます。

手順

1. **root** として次のコマンドを入力して、ユニットファイルを **/etc/systemd/system/** ディレクトリーにコピーします。

```
# cp /usr/lib/systemd/system/<name>.service /etc/systemd/system/<name>.service
```

2. コピーしたファイルをテキストエディターで開き、変更を加えます。

3. ユニットの変更を適用します。

```
# systemctl daemon-reload
# systemctl restart <name>.service
```

1.16. タイムアウト制限の変更

サービスごとにタイムアウト値を指定すると、正常に動作していないサービスによってシステムがフリーズすることを防ぐことができます。指定しない場合、タイムアウトのデフォルト値は、通常のサービスの場合は 90 秒、SysV 互換サービスの場合は 300 秒です。

手順

httpd サービスのタイムアウト制限を延長するには、以下を実行します。

1. **httpd** ユニットファイルを、**/etc/systemd/system/** ディレクトリーにコピーします。

```
# cp /usr/lib/systemd/system/httpd.service /etc/systemd/system/httpd.service
```

2. **/etc/systemd/system/httpd.service** ファイルを開き、**[Service]** セクションに **TimeoutStartUSec** 値を指定します。

```
...
[Service]
...
PrivateTmp=true
TimeoutStartSec=10

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

3. **systemd** デーモンを再ロードします。

```
# systemctl daemon-reload
```

4. **オプション:**新しいタイムアウト値を確認します。

```
# systemctl show httpd -p TimeoutStartUSec
```

注記

グローバルでタイムアウト制限を変更するには、**/etc/systemd/system.conf** ファイルの **DefaultTimeoutStartSec** を変更します。

1.17. 上書きされたユニットの監視

systemd-delta コマンドを使用すると、オーバーライドまたは変更されたユニットファイルの概要を表示できます。

手順

- オーバーライドまたは変更されたユニットファイルの概要を表示します。

```
# systemd-delta
```

上記のコマンドを実行すると、以下のような出力になります。

```
[EQUIVALENT] /etc/systemd/system/default.target → /usr/lib/systemd/system/default.target
[OVERRIDDEN] /etc/systemd/system/autofs.service →
/usr/lib/systemd/system/autofs.service

--- /usr/lib/systemd/system/autofs.service       2014-10-16 21:30:39.000000000 -0400
+++ /etc/systemd/system/autofs.service  2014-11-21 10:00:58.513568275 -0500
@@ -8,7 +8,8 @@
 EnvironmentFile=-/etc/sysconfig/autofs
 ExecStart=/usr/sbin/automount $OPTIONS --pid-file /run/autofs.pid
 ExecReload=/usr/bin/kill -HUP $MAINPID
-TimeoutSec=180
+TimeoutSec=240
+Restart=Always

 [Install]
 WantedBy=multi-user.target

[MASKED]    /etc/systemd/system/cups.service → /usr/lib/systemd/system/cups.service
[EXTENDED]   /usr/lib/systemd/system/sssd.service →
/etc/systemd/system/sssd.service.d/journal.conf

4 overridden configuration files found.
```

1.18. インスタンス化されたユニットの使用

単一のテンプレート設定を使用して、サービスの複数のインスタンスを管理できます。ユニットの汎用テンプレートを定義し、ランタイム時に特定のパラメーターを使用してそのユニットの複数のインスタンスを生成できます。テンプレートはアットマーク (@) で示されます。インスタンス化されたユニットは、(**Requires** オプションまたは **Wants** オプションを使用して) 別のユニットから開始することも、**systemctl start** コマンドで開始することもできます。インスタンス化されたサービスユニットの名前は以下のような形式となります。

```
<template_name>@<instance_name>.service
```

<template_name> は、テンプレート設定ファイルの名前です。<instance_name> を、ユニットインスタンスの名前に置き換えます。複数のインスタンスが同じテンプレートファイルを参照し、このテンプレートには、ユニットの全インスタンスに共通する設定オプションが含まれます。テンプレートユニットの名前には以下の形式が使用されます。

```
<unit_name>@.service
```

たとえば、ユニットファイルに次の **Wants** 設定を指定すると、

```
Wants=getty@ttyA.service getty@ttyB.service
```

この設定により、systemd が、最初に指定したサービスユニットを検索します。該当するユニットが見つからないと、@とタイプ接尾辞の間にある部分は無視され、**systemd** が **getty@.service** ファイルを検索し、そこから設定を読み取り、サービスを起動します。

たとえば、**getty@.service** テンプレートには以下のディレクティブが含まれます。

```
[Unit]
Description=Getty on %I
...
[Service]
ExecStart=-/sbin/agetty --noclear %I $TERM
...
```

上記のテンプレートから **getty@ttyA.service** および **getty@ttyB.service** をインスタンス化する場合、**Description**= は **Getty on ttyA** および **Getty on ttyB** として解決されます。

1.19. 重要なユニット指定子

すべてのユニット設定ファイルでは、**ユニット指定子** と呼ばれるワイルドカード文字を使用できます。ユニット指定子は、特定のユニットパラメーターを置き換え、ランタイム時に解釈されます。

表1.6 重要なユニット指定子

ユニット指定子	意味	説明
%n	完全ユニット名	タイプ接尾辞を含む完全ユニット名を表します。**%N** には同じ意味がありますが、禁止文字を ASCII コードに置き換えます。
%p	接頭辞名	タイプ接尾辞が削除されたユニット名を表します。インスタンス化されたユニットの %p は、ユニット名の@文字の前の部分を表します。
%i	インスタンス名	インスタンス化されたユニット名の@文字およびタイプ接尾辞間の部分です。**%I** には同じ意味がありますが、禁止文字を ASCII コードにも置き換えます。
%H	ホスト名	ユニット設定を読み込んだ時に稼働しているシステムのホスト名を表します。

ユニット指定子	意味	説明
%t	ランタイムディレクトリー	ランタイムディレクトリーを表します。これは、**root** ユーザーの **/run**、または非特権ユーザーの XDG_RUNTIME_DIR 変数の値になります。

ユニット指定子の詳細なリストは、**systemd.unit(5)** の man ページを参照してください。

1.20. 関連情報

- How to set limits for services in RHEL and systemd
- How to write a service unit file which enforces that particular services have to be started
- How to decide what dependencies a systemd service unit definition should have

第2章 起動時間を短縮するための SYSTEMD の最適化

システム管理者は、システムのパフォーマンスを最適化し、起動時間を短縮できます。**systemd** が起動中に開始するサービスを確認し、その必要性を評価できます。起動時に開始される特定のサービスを無効にすると、システムの起動時間を短縮できます。

2.1. システムの起動パフォーマンスを調べる

システムの起動時のパフォーマンスを調べる場合は、**systemd-analyze** コマンドを使用できます。特定のオプションを使用すると、systemd を調整して起動時間を短縮できます。

前提条件

- オプション: **systemd** を調べて起動時間を調整する前に、有効なサービスをすべてリスト表示します。

```
$ systemctl list-unit-files --state=enabled
```

手順

分析したい情報を選択します。

- 最後に正常に起動したときの起動時間に関する情報を分析します。

```
$ systemd-analyze
```

- 各 **systemd** ユニットのユニット初期化時間を分析します。

```
$ systemd-analyze blame
```

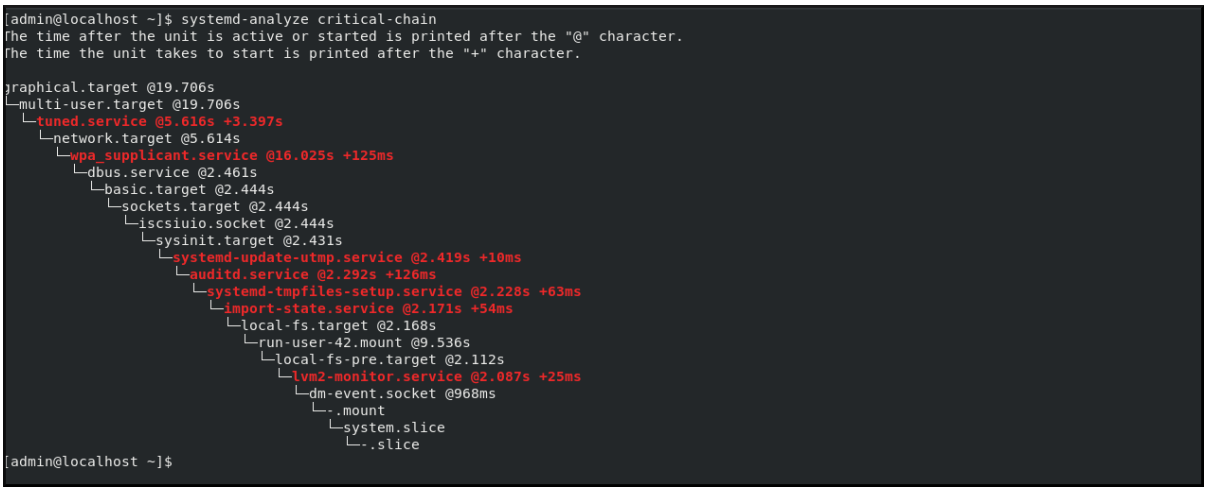
この出力では、システムが最後に起動した時に初期化にかかった時間に応じて、ユニットが降順で表示されます。

- 最後に正常に起動したときに、初期化に最も時間がかかったクリティカルなユニットを特定します。

```
$ systemd-analyze critical-chain
```

この出力では、起動に非常に時間がかかっているユニットが、赤字で強調表示されています。

図2.1 systemd-analyze critical-chain コマンドの出力



関連情報

- **systemd-analyze(1)** man ページ

2.2. 無効にしても安全なサービスを選択するためのガイド

デフォルトで起動時に有効になっている特定のサービスを無効にすることで、システムの起動時間を短縮できます。

- 有効なサービスをリスト表示します。

```
$ systemctl list-unit-files --state=enabled
```

- サービスを無効にします。

```
# systemctl disable <service_name>
```

お使いのオペレーティングシステムが安全で、希望通りに機能できるように、特定のサービスは有効にしたままにしておく必要があります。

無効にしても安全なサービスを選択するためのガイドとして、次の表を参照してください。この表には、Red Hat Enterprise Linux の最小インストールでデフォルトで有効になるすべてのサービスがリスト表示されています。

表2.1 RHEL の最小インストールで、デフォルトで有効になっているサービス

サービス名	無効にすることは可能か？	詳細情報

サービス名	無効にすることは可能か？	詳細情報
auditd.service	はい	**auditd.service** は、カーネルからの監査メッセージが必要ない場合に限り無効にします。**auditd.service** を無効にすると、**/var/log/audit/audit.log** ファイルが生成されないことに注意してください。無効にすると、ユーザーログイン、サービスの起動、パスワードの変更などの、一般的に確認されるアクションまたはレビューをさかのぼって確認することはできません。auditd には、カーネルの部分と、サービスそのものが含まれることに注意してください。**systemctl disable auditd** コマンドを実行すると、サービスを無効にするだけで、カーネル部分は無効にしません。システムの監査を完全に無効にするには、カーネルコマンドラインに **audit=0** と設定します。
autovt@.service	いいえ	このサービスは、本当に必要な場合に限り実行されるため、無効にする必要はありません。
crond.service	はい	crond.service を無効にすると crontab からアイテムが実行しないことに注意してください。
dbus-org.fedoraproject.FirewallD1.service	はい	**firewalld.service** へのシンボリックリンク
dbus-org.freedesktop.NetworkManager.service	はい	**NetworkManager.service** へのシンボリックリンク
dbus-org.freedesktop.nm-dispatcher.service	はい	**NetworkManager-dispatcher.service** へのシンボリックリンク
firewalld.service	はい	ファイアウォールが必要ない場合に限り **firewalld.service** を無効にします。
getty@.service	いいえ	このサービスは、本当に必要な場合に限り実行されるため、無効にする必要はありません。
import-state.service	はい	**import-state.service** は、ネットワークストレージからの起動が必要ない場合に限り無効にします。
irqbalance.service	はい	**irqbalance.service** は、CPU が 1 つしかない場合に限り無効にします。システムに CPU が複数ある場合は **irqbalance.service** を無効にしないでください。
kdump.service	はい	**kdump.service** は、カーネルクラッシュからのレポートが必要ない場合に限り無効にします。

サービス名	無効にすることは可能か？	詳細情報
loadmodules.service	はい	このサービスは、**/etc/rc.modules** ディレクトリーまたは **/etc/sysconfig/modules** ディレクトリーがなければ起動しません。つまり、RHEL の最小インストールでは起動しません。
lvm2-monitor.service	はい	**lvm2-monitor.service** は、論理ボリュームマネージャー (LVM) を使用しない場合に限り無効にします。
microcode.service	いいえ	そのサービスは、CPU 内のマイクロコードソフトウェアの更新を提供するため、無効にしないでください。
NetworkManager-dispatcher.service	はい	**NetworkManager-dispatcher.service** は、ネットワーク設定変更の通知が必要ない場合 (静的ネットワークなど) に限り無効にします。
NetworkManager-wait-online.service	はい	**NetworkManager-wait-online.service** は、システムの起動直後にネットワーク接続が利用できるようになっている必要がない場合に限り無効にします。このサービスを有効にすると、ネットワーク接続が有効になるまで、システムの起動が完了しません。これにより、起動時間が大幅に長くなることがあります。
NetworkManager.service	はい	**NetworkManager.service** は、ネットワークへの接続が必要ない場合に限り無効にします。
nis-domainname.service	はい	**nis-domainname.service** は、ネットワークインフォメーションサービス (NIS) を使用しない場合に限り無効にします。
rhsmcertd.service	いいえ	
rngd.service	はい	**rngd.service** は、システムでエントロピーがそれほど必要ない場合、またはハードウェアジェネレーターがない場合に限り無効にします。このサービスは、X.509 証明書の生成に使用されるシステム (たとえば FreeIPA サーバー) など、良好なエントロピーを多数必要とする環境で必要になります。
rsyslog.service	はい	**rsyslog.service** は、永続的なログが必要ない場合に限り、または **systemd-journald** を永続モードに設定した場合に限り無効にします。
selinux-autorelabel-mark.service	はい	**selinux-autorelabel-mark.service** は、SELinux を使用しない場合に限り無効にします。
sshd.service	はい	**sshd.service** は、OpenSSH サーバーへのリモートログインが必要ない場合に限り無効にします。

サービス名	無効にすることは可能か？	詳細情報
sssd.service	はい	**sssd.service** は、ネットワークを介して (たとえば LDAP や Kerberos を使用して) システムにログインするユーザーがいない場合に限り無効にします。**sssd.service** を無効にした場合は、**sssd-*** ユニットをすべて無効にすることを Red Hat は推奨します。
syslog.service	はい	**rsyslog.service** のエイリアス
tuned.service	はい	**tuned.service** は、パフォーマンスチューニングを使用する必要がない場合に限り無効にします。
lvm2-lvmpolld.socket	はい	**lvm2-lvmpolld.socket** は、論理ボリュームマネージャー (LVM) を使用しない場合に限り無効にします。
dnf-makecache.timer	はい	**dnf-makecache.timer** は、パッケージメターデータを自動的に更新する必要がない場合に限り無効にします。
unbound-anchor.timer	はい	**unbound-anchor.timer** は、DNSSEC (DNS Security Extensions) のルートトラストアンカーを毎日更新する必要がない場合に限り無効にします。このルートトラストアンカーは、DNSSEC 検証の Unbound リゾルバー、およびリゾルバーライブラリーにより使用されます。

サービスの詳細は、次のいずれかのコマンドを使用して表示できます。

```
$ systemctl cat <service_name>
```

```
$ systemctl help <service_name>
```

systemctl cat コマンドは、それぞれの **/usr/lib/systemd/system/<service>** サービスファイルの内容と、適用可能なすべてのオーバーライドを提供します。適用可能なオーバーライドには、**/etc/systemd/system/<service>** ファイルからのユニットファイルオーバーライド、または対応する **unit.type.d** ディレクトリーのドロップインファイルが含まれます。

関連情報

- **systemd.unit(5)** man ページ
- 特定のサービスの man ページを表示する **systemd help** コマンド

2.3. 関連情報

- **systemctl**(1) man ページ
- **systemd**(1) man ページ
- **systemd-delta**(1) man ページ

- **systemd.directives**(7) man ページ
- **systemd.unit**(5) man ページ
- **systemd.service**(5) man ページ
- **systemd.target**(5) man ページ
- **systemd.kill**(5) man ページ
- systemd Home Page