Red Hat Enterprise Linux 9

動的プログラミング言語のインストールおよび使用

Red Hat Enterprise Linux 9 での Python および PHP のインストールおよび使用

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 動的プログラミング言語のインストールおよび使用

Red Hat Enterprise Linux 9 での Python および PHP のインストールおよび使用

法律上の通知

概要

Python 3 をインストールおよび使用し、Python 3 RPM をパッケージ化して、Python スクリプトでインタープリターディレクティブを処理する方法について説明します。PHP スクリプト言語をインストールし、Apache HTTP Server または ngninx Web サーバーで PHP を使用し、コマンドラインインターフェイスから PHP スクリプトを実行します。

目次

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 PYTHON の概要

Python は、オブジェクト指向、命令、機能、手順などの複数のプログラミングパラダイムをサポートする、高レベルのプログラミング言語です。Python は動的なセマンティクスを持ち、汎用プログラミングに使用できます。

Red Hat Enterprise Linux では、システムツール、データ分析用のツール、Web アプリケーションを提供するパッケージなど、システムにインストールされている多くのパッケージが Python で記述されています。このパッケージを使用するには、**python*** パッケージがインストールされている必要があります。

1.1. PYTHON のバージョン

Python 3.9 が RHEL 9 におけるデフォルトの **Python** 実装に**Python 3.9** は、BaseOS リポジトリーにあるモジュール以外の **python3** RPM パッケージで配布され、通常はデフォルトでインストールされます。**Python 3.9** は、RHEL 9 のライフサイクル全体でサポートされます。

Python 3 の追加バージョンは、モジュール以外の RPM パッケージとして配布され、RHEL 9 のマイナーリリースの AppStream リポジトリーを通じてライフサイクルが短くなります。**Python 3** のこれらの 追加バージョンは、Python 3.9 と並行してインストールできます。

Python 2 は RHEL 9 に同梱されていません。

表1.1 RHEL 9 の Python バージョン

バージョン	インストールするパッケージ	コマンドの例	利用可能となったバージョン	ライフサイクル
Python 3.9	**python3**	**python3**、**pip3**	RHEL 9.0	RHEL 9 でフルサポート
Python 3.11	**python3.11**	**python3.11**、**pip3.11**	RHEL 9.2	より短い
Python 3.12	**python3.12**	**python3.12**, **pip3.12**	RHEL 9.4	より短い

サポート期間の詳細は、Red Hat Enterprise Linux のライフサイクル および Red Hat Enterprise Linux Application Streams ライフサイクル を参照してください。

1.2. RHEL 8 以降の PYTHON エコシステムの主な相違点

RHEL 8 と比較した場合の RHEL 9 の Python エコシステムの主な変更点は次のとおりです。

バージョンを指定しない python コマンド

バージョンを指定しない **python** コマンド (**/usr/bin/python**) は、**python-unversioned-command** パッケージで利用できます。一部のシステムでは、このパッケージはデフォルトでインストールされていません。バージョンを指定しない **python** コマンドを手動でインストールする場合は、**dnf install /usr/bin/python** コマンドを使用します。

RHEL 9 では、バージョンを指定しない **python** コマンドは、デフォルトの **Python 3.9** バージョンを指し、**python3** コマンドおよび **python3.9** コマンドと同等です。RHEL 9 では、バージョンを指定しないコマンドが **Python 3.9** 以外のバージョンを参照するように設定できません。

python コマンドは、対話式セッションを対象としています。実稼働環境では、**python3**、**python3.9**、**python3.11**、または **python3.12** を明示的に使用することが推奨されます。

バージョンを指定しない **python** コマンドは、**dnf remove /usr/bin/python** コマンドを使用してアンインストールできます。

別の **python** または **python3** コマンドが必要な場合は、**/usr/local/bin** または **~/.local/bin** にカスタムシンボリックリンクを作成するか、Python の仮想環境を使用できます。

バージョンを指定しないコマンドは、**python3-pip** パッケージの **/usr/bin/pip** など、他にもいくつか提供されています。RHEL 9 では、バージョンを指定しないコマンドはすべて、デフォルトの **Python 3.9** バージョンを指します。

アーキテクチャー固有の Python wheels

RHEL 9 にビルドされたアーキテクチャー固有の Python **wheels** は、アップストリームアーキテクチャーの命名に準拠しています。これにより、RHEL 9 で Python **wheels** をビルドし、RHEL 以外のシステムにインストールできます。以前の RHEL リリースにビルドされた Python **wheels** は、新しいバージョンと互換性があり、RHEL 9 にインストールできます。これは、Python 拡張機能を含む **wheels** (アーキテクチャーごとにビルド) にのみ影響を及ぼし、純粋な Python コードの Python **wheels** (アーキテクチャー固有ではない) には影響を及ぼさない点に注意してください。

第2章 PYTHON のインストールおよび使用

RHEL 9 では、**Python 3.9** はデフォルトの **Python** 実装になります。RHEL 9.2 以降では、**Python 3.11** は **python3.11** パッケージスイートとして利用でき、RHEL 9.4 以降では、**Python 3.12** は **python3.12** パッケージスイートとして利用できます。

バージョンを指定しない **python** コマンドは、デフォルトの **Python 3.9** バージョンを指します。

2.1. PYTHON 3 のインストール

デフォルトの **Python** 実装は通常、デフォルトでインストールされます。手動でインストールするには、以下の手順に従います。

手順

- **Python 3.9** をインストールするには、以下を使用します。

```
# dnf install python3
```

- **Python 3.11** をインストールするには、以下を使用します。

```
# dnf install python3.11
```

- **Python 3.12** をインストールするには、以下を使用します。

```
# dnf install python3.12
```

検証手順

- お使いのシステムにインストールされている **Python** のバージョンを確認するには、**Python** の必要なバージョン固有の **python** コマンドで **--version** をオプションに指定します。
- **Python 3.9** の場合

```
$ python3 --version
```

- **Python 3.11** の場合

```
$ python3.11 --version
```

- **Python 3.12** の場合:

```
$ python3.12 --version
```

2.2. PYTHON 3 追加パッケージのインストール

python3- で始まるパッケージには、デフォルトの **Python 3.9** バージョンのアドオンモジュールが含まれています。**python3.11-** で始まるパッケージには、**Python 3.11** のアドオンモジュールが含まれています。**python3.12-** で始まるパッケージには、**Python 3.12** のアドオンモジュールが含まれています。

手順

- Python 3.9 の **Requests** モジュールをインストールするには、以下を使用します。

```
# dnf install python3-requests
```

- Python 3.9 から **pip** パッケージインストーラーをインストールするには、以下を使用します。

```
# dnf install python3-pip
```

- Python 3.11 から **pip** パッケージインストーラーをインストールするには、以下を使用します。

```
# dnf install python3.11-pip
```

- Python 3.12 から **pip** パッケージインストーラーをインストールするには、以下を使用します。

```
# dnf install python3.12-pip
```

関連情報

- Python アドオンモジュールに関するアップストリームドキュメント

2.3. 開発者用の PYTHON 3 追加ツールのインストール

開発者向けの追加の **Python** ツールは、主に CodeReady Linux Builder (CRB) リポジトリーを介して配布されます。

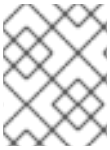
python3-pytest パッケージとその依存関係は、AppStream リポジトリーで利用できます。

たとえば、CRB リポジトリーには以下のパッケージが含まれます。

- **python3*-idle**
- **python3*-debug**
- **python3*-Cython**
- **python3.11-pytest** とその依存関係
- **python3.12-pytest** とその依存関係

重要

CodeReady Linux Builder リポジトリーの内容は、Red Hat ではサポートされていません。



注記

アップストリームの **Python** 関連のパッケージがすべて RHEL で利用できるわけではありません。

CRB リポジトリーからパッケージをインストールするには、以下の手順を実行してください。

手順

1. CodeReady Linux Builder リポジトリーを有効にします。

```
# subscription-manager repos --enable codeready-builder-for-rhel-9-x86_64-rpms
```

2. **python3*-Cython** パッケージをインストールします。

- Python 3.9 の場合

```
# dnf install python3-Cython
```

- Python 3.11 の場合

```
# dnf install python3.11-Cython
```

- Python 3.12 の場合:

```
# dnf install python3.12-Cython
```

関連情報

- CodeReady Linux Builder 内のコンテンツを有効にして使用する方法
- パッケージマニフェスト

2.4. PYTHON の使用

以下の手順では、Python インタープリターまたは Python 関連のコマンドを実行する例を示しています。

前提条件

- Python がインストールされていることを確認している。
- Python 3.11 または Python 3.12 用のサードパーティーアプリケーションをダウンロードしてインストールする場合は、**python3.11-pip** または **python3.12-pip** パッケージをインストールします。

手順

- Python 3.9 インタープリターまたは関連コマンドを実行するには、たとえば、以下を使用します。

```
$ python3
$ python3 -m venv --help
$ python3 -m pip install package
$ pip3 install package
```

- Python 3.11 インタープリターまたは関連コマンドを実行するには、たとえば、以下を使用します。

```
$ python3.11
$ python3.11 -m venv --help
$ python3.11 -m pip install package
```

```
$ pip3.11 install package
```

- Python 3.12 インタープリターまたは関連コマンドを実行するには、たとえば、以下を使用します。

```
$ python3.12
$ python3.12 -m venv --help
$ python3.12 -m pip install package
$ pip3.12 install package
```

第3章 PYTHON 3 RPM のパッケージ化

Python パッケージは、**pip** インストーラーを使用してアップストリームの PyPI リポジトリーから、または DNF パッケージマネージャーを使用してシステムにインストールできます。DNF は RPM パッケージ形式を使用します。これにより、ソフトウェアのダウンストリーム制御が強化されます。

ネイティブ Python パッケージのパッケージ形式は、Python Packaging Authority (PyPA) 仕様 によって定義されています。ほとんどの Python プロジェクトでは、パッケージ化に **distutils** または **setuptools** ユーティリティーを使用し、**setup.py** ファイルでパッケージ情報を定義しています。ただし、ネイティブ Python パッケージ作成の可能性は、時代とともに進化してきています。新しいパッケージング標準の詳細は、pyproject-rpm-macros を参照してください。

この章では、**setup.py** を使用する Python プロジェクトを RPM パッケージにパッケージ化する方法を説明します。このアプローチには、ネイティブ Python パッケージと比較して次の利点があります。

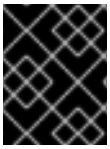
- Python および Python 以外のパッケージへの依存が可能です。依存関係は **DNF** パッケージマネージャーによって厳密に適用されます。
- パッケージに暗号で署名できます。暗号化署名を使用すると、RPM パッケージのコンテンツを、オペレーティングシステムの他の部分を使用して検証、統合、およびテストできます。
- ビルドプロセス中にテストを実行できます。

3.1. PYTHON パッケージ用の SPEC ファイルの説明

SPEC ファイルには、RPM のビルドに **rpmbuild** ユーティリティーを使用する命令が含まれています。命令は、一連のセクションに含まれています。SPEC ファイルには、セクションが定義されている 2 つの主要部分があります。

- プリアンブル (ボディーに使用されている一連のメタデータ項目が含まれています)
- ボディー (命令の主要部分が含まれています)

Python プロジェクトの RPM SPEC ファイルには、非 Python RPM SPEC ファイルと比較していくつかの詳細があります。



重要

Python ライブラリーの RPM パッケージの名前には、常に **python3-**、**python3.11-**、または **python3.12-** の接頭辞が含まれている必要があります。

その他の詳細は、以下の SPEC ファイルの **python3*-pello** パッケージの例に記載されています。その詳細の説明は、例の下に記載されている注意事項を参照してください。

Python で書かれた pello プログラムの SPEC ファイルサンプル

```
%global python3_pkgversion 3.11                    (1)

Name:       python-pello                           (2)
Version:    1.0.2
Release:    1%{?dist}
Summary:    Example Python library

License:    MIT
```

```
URL:            https://github.com/fedora-python/Pello
Source:         %{url}/archive/v%{version}/Pello-%{version}.tar.gz

BuildArch:      noarch
BuildRequires:  python%{python3_pkgversion}-devel                    3

# Build dependencies needed to be specified manually
BuildRequires:  python%{python3_pkgversion}-setuptools

# Test dependencies needed to be specified manually
# Also runtime dependencies need to be BuildRequired manually to run tests during build
BuildRequires:  python%{python3_pkgversion}-pytest >= 3


%global _description %{expand:
Pello is an example package with an executable that prints Hello World! on the command line.}

%description %_description

%package -n python%{python3_pkgversion}-pello                        4
Summary:        %{summary}

%description -n python%{python3_pkgversion}-pello %_description


%prep
%autosetup -p1 -n Pello-%{version}


%build
# The macro only supported projects with setup.py
%py3_build                                                           5


%install
# The macro only supported projects with setup.py
%py3_install


%check                                                               6
%{pytest}


# Note that there is no %%files section for the unversioned python module
%files -n python%{python3_pkgversion}-pello
%doc README.md
%license LICENSE.txt
%{_bindir}/pello_greeting

# The library files needed to be listed manually
%{python3_sitelib}/pello/

# The metadata files needed to be listed manually
%{python3_sitelib}/Pello-*.egg-info/
```

1. **python3_pkgversion** マクロを定義することで、このパッケージがビルドされる Python バージョンを設定します。デフォルトの Python バージョン 3.9 用にビルドするには、マクロをデフォルト

2. Python プロジェクトを RPM にパッケージ化するときは、常に **python-** 接頭辞をプロジェクトの元の名前に追加してください。ここでの元の名前は **pello** であるため、**ソース RPM (SRPM) の名前** は、**python-pello** になります。

3. **BuildRequires** は、このパッケージのビルドおよびテストに必要なパッケージを指定します。**BuildRequires** には、Python パッケージのビルドに必要なツールを提供するアイテム **python3-devel** (もしくは **python3.11-devel** または **python3.12-devel**) と、パッケージ化する特定のソフトウェアに必要な関連プロジェクト **python3-setuptools** (もしくは **python3.11-setuptools** または **python3.12-setuptools**)、あるいは %check セクションでテストを実行するために必要なランタイムとテストの依存関係を常に含めます。

4. バイナリー RPM (ユーザーがインストールできるパッケージ) の名前を選択する際には、バージョン管理された Python 接頭辞を追加します。デフォルトの Python 3.9 の場合は **python3-** 接頭辞、Python 3.11 の場合は **python3.11-** 接頭辞、Python 3.12 の場合は **python3.12-** 接頭辞を使用します。**%{python3_pkgversion}** マクロを使用できます。これは、明示的なバージョン (**3.11** など) に設定しない限り、デフォルトの Python バージョン 3.9 の場合は **3** と評価されます(脚注 1 を参照)。

5. %py3_build マクロおよび %py3_install マクロは、インストール場所、使用するインタープリター、その他の詳細を指定する追加の引数を使用して、**setup.py build** コマンドおよび **setup.py install** コマンドをそれぞれ実行します。

6. %check セクションは、パッケージ化されたプロジェクトのテストを実行する必要があります。正確なコマンドはプロジェクト自体に依存しますが、**%pytest** マクロを使用して、RPM に適した方法で **pytest** コマンドを実行することができます。

3.2. PYTHON 3 RPM の一般的なマクロ

SPEC ファイルでは、値をハードコーディングするのではなく、以下の **Python 3 RPM のマクロ** の表で説明されているマクロを常に使用します。SPEC ファイルの上に **python3_pkgversion** マクロを定義することで、これらのマクロで使用する Python 3 バージョンを再定義できます(「Python パッケージ用の SPEC ファイルの説明」 を参照)。**python3_pkgversion** マクロを定義すると、以下の表で説明されているマクロの値は、指定された Python 3 バージョンを反映します。

表3.1 Python 3 RPM 用のマクロ

マクロ	一般的な定義	説明
%{python3_pkgversion}	3	他のすべてのマクロで使用される Python バージョン。Python 3.11 を使用するには **3.11** に再定義し、Python 3.12 を使用するには **3.12** に再定義できます。
%{python3}	/usr/bin/python3	Python3 インタープリター
%{python3_version}	3.9	Python3 インタープリターの major.minor バージョン
%{python3_sitelib}	/usr/lib/python3.9/site-packages	pure-Python モジュールがインストールされている場所

マクロ	一般的な定義	説明
%{python3_sitearch}	/usr/lib64/python3.9/site-packages	アーキテクチャー固有の拡張モジュールを含むモジュールがインストールされている場所
%py3_build		RPM パッケージに適した引数で **setup.py build** コマンドを実行します。
%py3_install		RPM パッケージに適した引数で **setup.py install** コマンドを実行します。
%{py3_shebang_flags}	s	Python インタープリターディレクティブマクロのデフォルトのフラグセット **%py3_shebang_fix**
%py3_shebang_fix		Python インタープリターディレクティブを **#! %{python3}** に変更すると、既存のフラグ (見つかった場合) を保持し、**%{py3_shebang_flags}** マクロで定義されたフラグを追加します。

関連情報

- アップストリームドキュメントの Python マクロ

3.3. PYTHON RPM の自動生成された依存関係の使用

次の手順では、Python プロジェクトを RPM としてパッケージ化するときに自動生成された依存関係を使用する方法を説明します。

前提条件

- RPM の SPEC ファイルが存在する。詳細は、Python パッケージの SPEC ファイルの説明 を参照してください。

手順

1. アップストリームで提供されるメタデータを含む次のディレクトリーのいずれかが、結果の RPM に含まれていることを確認します。
   - **.dist-info**
   - **.egg-info**
     RPM ビルドプロセスは、これらのディレクトリーから仮想 **pythonX.Ydist** Provides を自動的に生成します。次に例を示します。

     ```
     python3.9dist(pello)
     ```

     次に、Python 依存関係ジェネレーターはアップストリームメタデータを読み取り、生成された **pythonX.Ydist** 仮想 Provides を使用して各 RPM パッケージのランタイム要件を生成します。たとえば、生成された要件タグは次のようになります。

```
Requires: python3.9dist(requests)
```

2. 生成された require を検査します。
3. 生成された require の一部を削除するには、次のいずれかの方法を使用します。
   a. SPEC ファイルの **%prep** セクションでアップストリーム提供のメタデータを変更します。
   b. アップストリームドキュメント で説明されている依存関係の自動フィルタリングを使用します。
4. 自動依存関係ジェネレーターを無効にするには、メインパッケージの **%description** 宣言の上に **%{?python_disable_dependency_generator}** マクロを含めます。

関連情報

- Automatically generated dependencies

第4章 PYTHON スクリプトでのインタープリターディレクティブの処理

Red Hat Enterprise Linux 9 では、実行可能な Python スクリプトは、少なくとも主要な Python バージョンを明示的に指定するインタープリターディレクティブ (別名 hashbangs または shebangs) を使用することが想定されます。以下に例を示します。

```
#!/usr/bin/python3
#!/usr/bin/python3.9
#!/usr/bin/python3.11
#!/usr/bin/python3.12
```

/usr/lib/rpm/redhat/brp-mangle-shebangs BRP (buildroot policy) スクリプトは、RPM パッケージをビルドする際に自動的に実行され、実行可能なすべてのファイルでインタープリターディレクティブを修正しようとします。

BRP スクリプトは、以下のようにあいまいなインタープリターディレクティブを含む Python スクリプトを検出すると、エラーを生成します。

```
#!/usr/bin/python
```

または

```
#!/usr/bin/env python
```

4.1. PYTHON スクリプトでのインタープリターディレクティブの変更

次の手順を使用して、RPM ビルド時にビルドエラーが発生する Python スクリプト内のインタープリターディレクティブを変更します。

前提条件

- Python スクリプトのインタープリターディレクティブの一部でビルドエラーが発生する。

手順

- インタープリターディレクティブを変更するには、以下のタスクのいずれかを実行します。
  - SPEC ファイルの **%prep** セクションで次のマクロを使用します。

    ```
    # %py3_shebang_fix SCRIPTNAME …
    ```

    SCRIPTNAME には、任意のファイル、ディレクトリー、またはファイルおよびディレクトリーのリストを指定できます。

    結果として、リストしたすべてのファイルと、リストしたディレクトリー内のすべての **.py** ファイルのインタープリターディレクティブが、**%{python3}** を指すように変更されます。元のインタープリターディレクティブの既存のフラグは保持され、**%{py3_shebang_flags}** マクロで定義された追加のフラグが追加されます。SPEC ファイルの **%{py3_shebang_flags}** マクロを再定義すると、追加されるフラグを変更できます。

  - **python3-devel** パッケージから **pathfix.py** スクリプトを適用します。

```
# pathfix.py -pn -i %{python3} PATH ...
```

複数のパスを指定できます。**PATH** がディレクトリーの場合、**pathfix.py** はあいまいなインタープリターディレクティブを持つスクリプトだけでなく、**^[a-zA-Z0-9_]+\.py$** のパターンに一致する Python スクリプトを再帰的にスキャンします。上記のコマンドを **%prep** セクションまたは **%install** セクションの最後に追加します。

- パッケージ化した Python スクリプトを、想定される形式に準拠するように変更します。この目的のために、RPM ビルドプロセスの外部で **pathfix.py** スクリプトを使用することもできます。**pathfix.py** を RPM ビルド以外で実行する場合は、前述の例の **%{python3}** を、**/usr/bin/python3** または **/usr/bin/python3.11** などのインタープリターディレクティブのパスに置き換えます。

関連情報

- Interpreter invocation

第5章 TCL/TK のインストール

5.1. TCL/TK の概要

Tcl は動的プログラミング言語であり、**Tk** はグラフィカルユーザーインターフェイス (GUI) ツールキットです。これらは、グラフィカルインターフェイスを備えたクロスプラットフォームアプリケーションを開発するための強力で使いやすいプラットフォームを提供します。動的プログラミング言語として、'Tcl' はスクリプトを作成するためのシンプルで柔軟な構文を提供します。**tcl** パッケージは、この言語と C ライブラリーのインタープリターを提供します。**Tk** は、GUI ツールキットとして使用でき、グラフィカルインターフェイスを作成するためのツールとウィジェットのセットを提供します。ボタン、メニュー、ダイアログボックス、テキストボックス、キャンバスなどのさまざまなユーザーインターフェイス要素をグラフィックの描画に使用できます。**Tk** は、多くの動的プログラミング言語の GUI です。

Tcl/Tk の詳細は Tcl/Tk マニュアル または Tcl/Tk ドキュメントの Web ページ を参照してください。

5.2. TCL のインストール

通常、デフォルトの **Tcl** 実装がデフォルトでインストールされます。手動でインストールするには、以下の手順に従います。

手順

- **Tcl** をインストールするには、以下を使用します。

```
# dnf install tcl
```

検証手順

- システムにインストールされている Tcl バージョンを確認するには、インタープリター **tclsh** を実行します。

```
$ tclsh
```

- インタープリターで次のコマンドを実行します。

```
% info patchlevel
8.6
```

- **Ctrl**+**C** を押すと、インタープリターインターフェイスを終了できます。

5.3. TK のインストール

通常、デフォルトの **Tk** 実装がデフォルトでインストールされます。手動でインストールするには、以下の手順に従います。

手順

- **Tk** をインストールするには、以下を使用します。

```
# dnf install tk
```

検証手順

検証手順

- システムにインストールされている **Tk** バージョンを確認するには、ウィンドウシェル **wish** を実行します。グラフィカル表示を実行している必要があります。

```
$ wish
```

- シェルで次のコマンドを実行します。

```
% puts $tk_version
8.6
```

- **Ctrl**+**C** を押すと、インタープリターインターフェイスを終了できます。

第6章 PHP スクリプト言語の使用

Hypertext Preprocessor (PHP) は、主にサーバー側スクリプトに使用される汎用スクリプト言語です。PHP を使用すると、Web サーバーを使用して PHP コードを実行できます。

6.1. PHP スクリプト言語のインストール

RHEL 9 では、次のバージョンと形式の PHP を利用できます。

- **php** RPM パッケージ形式の PHP 8.0
- **php:8.1** モジュールストリーム形式の PHP 8.1
- **php:8.2** モジュールストリーム形式の PHP 8.2

手順

シナリオに応じて、次のいずれかの手順を実行します。

- PHP 8.0 をインストールするには、以下を実行します。

```
# dnf install php
```

- デフォルトのプロファイルで **php:8.1** または **php:8.2** モジュールストリームをインストールするには、以下を実行します。

```
# dnf module install php:8.1
```

  デフォルトの **common** プロファイルは **php-fpm** パッケージもインストールし、Apache HTTP Server または nginx で使用する PHP を事前設定します。

- **php:8.1** または **php:8.2** モジュールストリームの特定のプロファイルをインストールするには、たとえば以下のようになります。

```
# dnf module install php:8.1/profile
```

  使用可能なプロファイルは次のとおりです。

  - **common**: Web サーバーを使用したサーバー側のスクリプトのデフォルトプロファイル。これには、最も広く使用されているエクステンションが含まれています。
  - **minimal**: このプロファイルは、Web サーバーを使用せずに PHP でのスクリプト用のコマンドラインインターフェイスのみをインストールします。
  - **devel** - このプロファイルには、common プロファイルのパッケージと開発用の追加パッケージが含まれます。
    たとえば、Web サーバーを使用しない PHP 8.1 をインストールするには、以下を使用します。

```
# dnf module install php:8.1/minimal
```

関連情報

- DNF ツールを使用したソフトウェアの管理

6.2. WEB サーバーでの PHP スクリプト言語の使用

6.2.1. Apache HTTP Server での PHP の使用

Red Hat Enterprise Linux 9 では、**Apache HTTP Server** で PHP を FastCGI プロセスサーバーとして実行できます。FastCGI Process Manager (FPM) は、Web サイトで高負荷を管理できるようにする代替の PHP FastCGI デーモンです。RHEL 9 では、PHP はデフォルトで FastCGI Process Manager を使用します。

FastCGI プロセスサーバーを使用して PHP コードを実行できます。

前提条件

- PHP スクリプト言語がシステムにインストールされている。

手順

1. **httpd** パッケージをインストールします。

```
# dnf install httpd
```

2. **Apache HTTP Server** を起動します。

```
# systemctl start httpd
```

   または、**Apache HTTP Server** をシステムで実行している場合は、PHP のインストール後に **httpd** サービスを再起動します。

```
# systemctl restart httpd
```

3. **php-fpm** サービスを起動します。

```
# systemctl start php-fpm
```

4. オプション:両方のサービスが起動時に開始できるようにします。

```
# systemctl enable php-fpm httpd
```

5. PHP の設定に関する情報を取得するために、以下の内容を含む **index.php** ファイルを **/var/www/html/** ディレクトリーに作成します。

```
# echo '<?php phpinfo(); ?>' > /var/www/html/index.php
```

6. **index.php** ファイルを実行するために、ブラウザーで以下を指定します。

```
http://<hostname>/
```

7. オプション:特定の要件がある場合は、設定を調整します。
   - **/etc/httpd/conf/httpd.conf** - 一般的な **httpd** 設定
   - **/etc/httpd/conf.d/php.conf** - **httpd** の PHP 固有の設定

- **/usr/lib/systemd/system/httpd.service.d/php-fpm.conf** - デフォルトでは、**php-fpm** サービスは **httpd** と一緒に起動します。
- **/etc/php-fpm.conf** - FPM の主な設定
- **/etc/php-fpm.d/www.conf** - デフォルトの **www** プール設定

例6.1 "Hello, World!" の実行Apache HTTP Server を使用した PHP スクリプト

1. **/var/www/html/** ディレクトリーにプロジェクト用の **hello** ディレクトリーを作成します。

```
# mkdir hello
```

2. 以下の内容を含む **/var/www/html/hello/** ディレクトリーに **hello.php** ファイルを作成します。

```
# <!DOCTYPE html>
<html>
<head>
<title>Hello, World! Page</title>
</head>
<body>
<?php
    echo 'Hello, World!';
?>
</body>
</html>
```

3. **Apache HTTP Server** を起動します。

```
# systemctl start httpd
```

4. **hello.php** ファイルを実行するには、ブラウザーに以下を指定します。

```
http://<hostname>/hello/hello.php
```

結果として、"Hello, World!" というテキストを含む Web ページが表示されます。

関連情報

- Apache HTTP Web サーバーの設定

6.2.2. nginx Web サーバーでの PHP の使用

nginx Web サーバーを介して PHP コードを実行できます。

前提条件

- PHP スクリプト言語がシステムにインストールされている。

手順

1. **nginx** パッケージをインストールします。

```
# dnf install nginx
```

2. **nginx** サーバーを起動します。

```
# systemctl start nginx
```

または、使用中のシステムで **nginx** サーバーを実行している場合は、PHP のインストール後に **nginx** サービスを再起動します。

```
# systemctl restart nginx
```

3. **php-fpm** サービスを起動します。

```
# systemctl start php-fpm
```

4. オプション:両方のサービスが起動時に開始できるようにします。

```
# systemctl enable php-fpm nginx
```

5. PHP の設定に関する情報を取得するには、以下の内容を含む **index.php** ファイルを **/usr/share/nginx/html/** ディレクトリーに作成します。

```
# echo '<?php phpinfo(); ?>' > /usr/share/nginx/html/index.php
```

6. **index.php** ファイルを実行するために、ブラウザーで以下を指定します。

```
http://<hostname>/
```

7. オプション:特定の要件がある場合は、設定を調整します。

- **/etc/nginx/nginx.conf** - **nginx** の主な設定
- **/etc/nginx/conf.d/php-fpm.conf** - **nginx** の FPM 設定
- **/etc/php-fpm.conf** - FPM の主な設定
- **/etc/php-fpm.d/www.conf** - デフォルトの **www** プール設定

例6.2 "Hello, World!" の実行nginx サーバーを使用した PHP スクリプト

1. プロジェクトの **hello** ディレクトリーを **/usr/share/nginx/html/** ディレクトリーに作成します。

```
# mkdir hello
```

2. 以下の内容で **/usr/share/nginx/html/hello/** ディレクトリーに **hello.php** ファイルを作成します。

```
# <!DOCTYPE html>
<html>
<head>
```

```
<title>Hello, World! Page</title>
</head>
<body>
<?php
    echo 'Hello, World!';
?>
</body>
</html>
```

3. **nginx** サーバーを起動します。

```
# systemctl start nginx
```

4. **hello.php** ファイルを実行するには、ブラウザーに以下を指定します。

```
http://<hostname>/hello/hello.php
```

結果として、"Hello, World!" というテキストを含む Web ページが表示されます。

関連情報

- NGINX の設定および設定

6.3. コマンドラインインターフェイスを使用した PHP スクリプトの実行

通常、PHP スクリプトは Web サーバーを使用して実行されますが、コマンドラインインターフェイスを使用して実行することも可能です。

前提条件

- PHP スクリプト言語がシステムにインストールされている。

手順

1. テキストエディターで **filename.php** ファイルを作成します。
filename は、使用するファイル名に置き換えます。
2. コマンドラインから、作成した **filename.php** ファイルを実行します。

```
# php filename.php
```

例6.3 "Hello, World!" の実行コマンドラインインターフェイスを使用した PHP スクリプト

1. テキストエディターを使用して、以下の内容で **hello.php** ファイルを作成します。

```
<?php
    echo 'Hello, World!';
?>
```

2. コマンドラインで **hello.php** ファイルを実行します。

```
# php hello.php
```

結果として、"Hello, World!" が出力されます。

6.4. 関連情報

- **httpd(8)** - **httpd** サービスの man ページ。コマンドラインオプションの全リストが記載されています。
- **httpd.conf (5)** - **httpd** 設定の man ページ。 **httpd** 設定ファイルの構造と場所が説明されています。
- **nginx(8)** - **nginx** Web サーバーの man ページ。コマンドラインオプションの全リストとシグナルのリストが記載されています。
- **php-fpm(8)** - PHP FPM の man ページ。コマンドラインオプションおよび設定ファイルの全リストが記載されています。