Red Hat Enterprise Linux 9

安装和使用动态编程语言

在 Red Hat Enterprise Linux 9 中安装和使用 Python 和 PHP

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 安装和使用动态编程语言

在 Red Hat Enterprise Linux 9 中安装和使用 Python 和 PHP

法律通告

摘要

安装和使用 Python 3、软件包 Python 3 RPM，并了解如何在 Python 脚本中处理解释器指令。安装 PHP 脚本语言，将 PHP 与 Apache HTTP 服务器或 ngninx Web 服务器一起使用，并使用命令行界面运行 PHP 脚本。

目录

对红帽文档提供反馈 3

第 1 章 PYTHON 简介 4
1.1. PYTHON 版本 4
1.2. 自 RHEL 8 开始的 PYTHON 生态系统的主要区别 4

第 2 章 安装和使用 PYTHON 6
2.1. 安装 PYTHON 3 6
2.2. 安装其他 PYTHON 3 软件包 6
2.3. 为开发人员安装其他 PYTHON 3 工具 7
2.4. 使用 PYTHON 8

第 3 章 打包 PYTHON 3 RPM 9
3.1. SPEC 文件是 PYTHON 软件包的描述 9
3.2. PYTHON 3 RPM 的常见宏 11
3.3. 为 PYTHON RPM 使用自动生成的依赖项 12

第 4 章 在 PYTHON 脚本中处理解释器指令 13
4.1. 修改 PYTHON 脚本中的解释器指令 13

第 5 章 安装 TCL/TK 15
5.1. TCL/TK 简介 15
5.2. 安装 TCL 15
5.3. 安装 TK 15

第 6 章 使用 PHP 脚本语言 17
6.1. 安装 PHP 脚本语言 17
6.2. 通过 WEB 服务器使用 PHP 脚本语言 17
6.3. 使用命令行界面运行 PHP 脚本 21
6.4. 其它资源 21

对红帽文档提供反馈

我们感谢您对我们文档的反馈。帮助我们如何进行改进。

通过 Jira 提交反馈（需要帐户）

1. 登录到 Jira 网站。
2. 在顶部导航栏中点 **Create**
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您的建议以改进。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 PYTHON 简介

Python 是一种高级编程语言，支持多种编程模式，如面向对象、所需功能以及程序式范式。Python 具有动态语义，可用于通用编程。

使用 Red Hat Enterprise Linux 时，系统中安装的许多软件包（如提供系统工具、数据分析工具或 Web 应用程序的软件包）会使用 Python 编写。要使用这些软件包，您必须安装 **python*** 软件包。

1.1. PYTHON 版本

Python 3.9 是 RHEL 9 中的默认 **Python** 实现。**Python 3.9** 在 BaseOS 存储库中的非模块化 **python3** RPM 软件包中分发，通常默认安装。**Python 3.9** 将支持 RHEL 9 的整个生命周期。

其他版本的 **Python 3** 作为非模块化 RPM 软件包发布，且在 RHEL 9 次版本中通过 AppStream 软件仓库提供较短的生命周期。您可以与 Python 3.9 并行安装这些额外的 **Python 3** 版本。

Python 2 不随 RHEL 9 提供。

表 1.1. RHEL 9 中的 Python 版本

Version	要安装的软件包	命令示例	此后提供	生命周期
Python 3.9	**python3**	**python3**, **pip3**	RHEL 9.0	完整的 RHEL 9
Python 3.11	**python3.11**	**python3.11**, **pip3.11**	RHEL 9.2	较短
Python 3.12	**python3.12**	**python3.12**, **pip3.12**	RHEL 9.4	较短

有关支持长度的详情，请查看 Red Hat Enterprise Linux 生命周期 和 Red Hat Enterprise Linux 应用程序流生命周期。

1.2. 自 RHEL 8 开始的 PYTHON 生态系统的主要区别

与 RHEL 8 相比，以下是 RHEL 9 中 Python 生态系统中的主要变化：

unversioned python 命令

python 命令的未指定版本形式(**/usr/bin/python**)在 **python-unversioned-command** 软件包中提供。在某些系统中，默认情况下不安装此软件包。要手动安装 **python** 命令的未指定版本形式，请使用 **dnf install /usr/bin/python** 命令。

在 RHEL 9 中，**python** 命令的未指定版本形式指向默认的 **Python 3.9** 版本，它相当于 **python3** 和 **python3.9** 命令。在 RHEL 9 中，您无法配置未指定版本的命令以指向 **Python 3.9** 外的不同版本。

python 命令用于交互式会话。在生产环境中，建议明确使用 **python3**、**python 3.9**、**python3.11** 或 **python3.12**。

您可以使用 **dnf remove /usr/bin/python** 命令卸载未指定版本的 **python** 命令。

如果您需要不同的 **python** 或 **python3** 命令，您可以在 **/usr/local/bin** 或 **~/.local/bin** 中创建自定义符号链接，或使用 Python 虚拟环境。

还提供其他未版本化的命令，如 **python3-pip** 软件包中的 **/usr/bin/pip**。在 RHEL 9 中，所有未指定版本的命令都指向默认的 **Python 3.9** 版本。

特定于架构的 Python wheels

在 RHEL 9 上构建的特定于体系结构的 Python **wheel** 新建了上游架构命名，允许客户在 RHEL 9 上构建其 Python **wheel** 并在非 RHEL 系统中安装它们。在以前的 RHEL 版本上构建的 Python **wheel** 与后期版本兼容，可以在 RHEL 9 上安装。请注意，这仅影响包含 Python 扩展的 **wheel**，这些扩展针对每个架构构建，而不影响包含纯 Python 代码的 Python **wheels**，这不是特定于架构的 Python wheel。

第 2 章 安装和使用 PYTHON

在 RHEL 9 中，**Python 3.9** 是默认的 **Python** 实施。从 RHEL 9.2 开始，**Python 3.11** 作为 **python3.11** 软件包套件提供，自 RHEL 9.4 开始，**Python 3.12** 作为 **python3.12** 软件包套件提供。

unversioned **python** 命令指向默认的 **Python 3.9** 版本。

2.1. 安装 PYTHON 3

通常默认安装默认的 **Python** 实现。要手动安装它，请使用以下步骤。

步骤

- 要安装 **Python 3.9**，请使用：

```
# dnf install python3
```

- 要安装 **Python 3.11**，请使用：

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

- 要安装 **Python 3.12**，请使用：

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

验证步骤

- 要验证系统上安装的 **Python** 版本，请使用 **--version** 选项以及特定于您所需 **Python** 版本的 **python** 命令。
- 对于 **Python 3.9** ：

```
$ python3 --version
```

- 对于 **Python 3.11** ：

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

- 对于 **Python 3.12** ：

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

2.2. 安装其他 PYTHON 3 软件包

前缀为 **python3-** 的软件包包含默认 **Python 3.9** 版本的附加组件模块。前缀为 **python3.11-** 的软件包包含 **Python 3.11** 的附加组件模块。前缀为 **python3.12-** 的软件包包含 **Python 3.12** 的附加组件模块。

步骤

- 要为 **Python 3.9** 安装 **Requests** 模块，请使用：

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

- 要从 Python 3.9 安装 **pip** 软件包安装程序，请使用：

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

- 要从 Python 3.11 安装 **pip** 软件包安装程序，请使用：

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

- 要从 Python 3.12 安装 **pip** 软件包安装程序，请使用：

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

其他资源

- 有关 Python 附加组件模块的上游文档

2.3. 为开发人员安装其他 PYTHON 3 工具

开发人员的其他 **Python** 工具主要通过 CodeReady Linux Builder (CRB)存储库分发。

python3-pytest 软件包及其依赖项在 AppStream 存储库中提供。

例如，CRB 存储库包含以下软件包：

- **python3*-idle**
- **python3*-debug**
- **python3*-Cython**
- **python3.11-pytest** 及其依赖项
- **python3.12-pytest** 及其依赖项。

重要

红帽不支持 CodeReady Linux Builder 存储库中的内容。

注意

并非所有与上游 **Python** 相关的软件包都在 RHEL 中提供。

要从 CRB 存储库安装软件包，请使用以下流程。

步骤

1. 启用 CodeReady Linux Builder 存储库：

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

2. 安装 **python3*-Cython** 软件包：

   - 对于 Python 3.9 ：

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

   - 对于 Python 3.11 ：

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

   - 对于 Python 3.12 ：

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

其他资源

- 如何在 CodeReady Linux Builder 中启用和使用内容
- 软件包清单

2.4. 使用 PYTHON

以下流程包含运行 Python 解释器或 Python 相关命令的示例。

先决条件

- 确保 Python 已安装。
- 如果要为 Python 3.11 或 Python 3.12 下载并安装第三方应用程序，请安装 **python3.11-pip** 或 **python3.12-pip** 软件包。

步骤

- 要运行 Python 3.9 解释器或相关命令，请使用：

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

- 要运行 Python 3.11 解释器或相关命令，请使用：

  ```
  $ python3.11
  $ python3.11 -m venv --help
  $ python3.11 -m pip install package
  $ pip3.11 install package
  ```

- 要运行 Python 3.12 解释器或相关命令，请使用：

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

第 3 章 打包 PYTHON 3 RPM

您可以使用 **pip** 安装程序，或使用 DNF 软件包管理器在系统中安装 Python 软件包。DNF 使用 RPM 软件包格式，它提供了更好的软件下游控制。

原生 Python 软件包的打包格式由 Python Packaging Authority (PyPA) 规范定义。大多数 Python 项目使用 **distutils** 或 **setuptools** 实用程序进行打包，并在 **setup.py** 文件中定义的软件包信息。然而，创建原生 Python 软件包可能会随着时间而有新的演变。有关新兴打包标准的更多信息，请参阅 pyproject-rpm-macros。

本章论述了如何将 **setup.py** 的 Python 项目打包到一个 RPM 软件包中。与原生 Python 软件包相比，此方法提供以下优点：

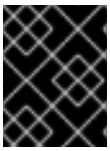
- 可以对 Python 和非 Python 软件包的依赖项，并严格由 **DNF** 软件包管理器强制执行。
- 您可以用加密的方式为软件包签名。使用加密签名，您可以验证、集成和测试 RPM 软件包的内容与操作系统的其余部分。
- 您可以在构建过程中执行测试。

3.1. SPEC 文件是 PYTHON 软件包的描述

SPEC 文件包含 **rpmbuild** 实用程序用于构建 RPM 的指令。这些指令包含在不同的部分。SPEC 文件有两个主要部分用于定义构建指令：

- Preamble（包含一系列在 Body 中使用的元数据项）
- Body（包含指令的主要部分）

与非 Python RPM SPEC 文件相比，Python 项目的 RPM SPEC 文件有一些特定信息。



重要

Python 库的任何 RPM 软件包的名称必须始终包含 **python3-**、**python3.11-** 或 **python3.12-** 前缀。

以下 **python3*-pello** 软件包的 SPEC 文件示例中显示了其他具体信息。有关此类特定描述，请查看示例中的备注。

使用 Python 编写的 pello 程序的 SPEC 文件示例

```
%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. 通过定义 **python3_pkgversion** 宏，您可以设置将为哪个 Python 版本构建此软件包。要为默认的 Python 版本 3.9 构建，请将宏设置为默认值 **3** 或删除整行。

2. 将 Python 项目打包到 RPM 中时，需要将 **python-** 前缀添加到项目的原始名称中。此处的原始名称为 **pello**，因此 **源 RPM (SRPM)的名称** 是 **python-pello**。

3 **BuildRequires** 指定了构建和测试此软件包所需的软件包。在 **BuildRequires** 中，始终包括提供构建 Python 软件包所需的工具： **python3-devel** （或 **python3.11-devel** 或 **python3.12-devel**）以及您

4 当为二进制 RPM 选择名称（用户将能够安装的软件包）时，请添加版本化的 Python 前缀。将 **python3-** 前缀用于默认的 Python 3.9、Python 3.11 的 **python3.11-** 前缀，或 Python 3.12 的 **python3.12-** 前缀。您可以使用 **%{python3_pkgversion}** 宏，它针对默认的 Python 版本 3.9 评估为 **3**，除非您将其设置为显式版本，例如 **3.11** （请参阅脚注 1）。

5 **%py3_build** 和 **%py3_install** 宏会分别运行 **setup.py build** 和 **setup.py install** 命令，使用附加参数来指定安装位置、要使用的解释器以及其他详情。

6 **%check** 部分应该运行打包项目的测试。确切的命令取决于项目本身，但可以使用 **%pytest** 宏以 RPM 友好的方式运行 **pytest** 命令。

3.2. PYTHON 3 RPM 的常见宏

在 SPEC 文件中，使用*用于 Python 3 RPM 的宏*表中的内容来使用宏而不是使用硬编码。您可以通过在 SPEC 文件之上定义 **python3_pkgversion** 宏来重新定义在这些宏中使用哪个 Python 3 版本（请参阅 第 3.1 节 “SPEC 文件是 Python 软件包的描述”）。如果您定义了 **python3_pkgversion** 宏，则下表中描述的宏的值将反映指定的 Python 3 版本。

表 3.1. Python 3 RPM 宏

Macro	常规定义	描述
%{python3_pkgversion}	3	所有其他宏使用的 Python 版本。可以重新定义到 **3.11** 以使用 Python 3.11，或把 **3.12** 重新配置为使用 Python 3.12
%{python3}	/usr/bin/python3	Python 3 解释器
%{python3_version}	3.9	Python 3 解释器的 major.minor 版本
%{python3_sitelib}	/usr/lib/python3.9/site-packages	安装纯 Python 模块的位置
%{python3_sitearch}	/usr/lib64/python3.9/site-packages	安装包含特定于架构扩展模块的模块的位置
%py3_build		使用适用于 RPM 软件包的参数运行 **setup.py build** 命令
%py3_install		使用适用于 RPM 软件包的参数运行 **setup.py install** 命令
%{py3_shebang_flags}	s	Python 解释器指令宏的默认标记集，**%py3_shebang_fix**
%py3_shebang_fix		将 Python 解释器指令改为 **#! %{python3}**，保留任何现有标志（如果找到），并添加在 **%{py3_shebang_flags}** 宏中定义的标记

其它资源

- 上游文档中的 Python 宏

3.3. 为 PYTHON RPM 使用自动生成的依赖项

以下流程描述了如何在将 Python 项目打包为 RPM 时使用自动生成的依赖项。

先决条件

- RPM 的 SPEC 文件存在。如需更多信息，请参阅 Python 软件包的 SPEC 文件描述。

步骤

1. 确保以下包含上游提供元数据的目录之一包含在生成的 RPM 中：
   - **.dist-info**
   - **.egg-info**
     RPM 构建过程会自动从这些目录中生成虚拟 **pythonX.Ydist**，例如：

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

     然后，Python 依赖项生成器读取上游元数据，并使用生成的 **pythonX.Ydist** 虚拟提供为每个 RPM 软件包生成运行时要求。例如，生成的要求标签可能如下所示：

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

2. 检查生成的要求。
3. 要删除其中的一些生成的需要，请使用以下方法之一：
   a. 在 SPEC 文件的 **%prep** 部分中修改上游提供的元数据。
   b. 使用上游文档中描述的依赖项自动过滤。
4. 要禁用自动依赖项生成器，请在主软件包的 **%description** 声明中包含 **%{?python_disable_dependency_generator}** 宏。

其它资源

- 自动生成的依赖项

第 4 章 在 PYTHON 脚本中处理解释器指令

在 Red Hat Enterprise Linux 9 中，可执行 Python 脚本应该使用解析程序指令（也称为 hashbangs 或 shebangs），至少指定主 Python 版本。例如：

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

在构建任何 RPM 软件包时，**/usr/lib/rpm/redhat/brp-mangle-shebangs** buildroot 策略 (BRP) 脚本会自动运行，并尝试在所有可执行文件中更正解释器指令。

当遇到带有模糊的解释解释器指令的 Python 脚本时，BRP 脚本会生成错误，例如：

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

或者

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

4.1. 修改 PYTHON 脚本中的解释器指令

使用以下步骤修改 Python 脚本中的解释器指令，以便在 RPM 构建时出现错误。

先决条件

- Python 脚本中的一些解释器指令会导致构建错误。

步骤

- 要修改解释器指令，请完成以下任务之一：
  - 在您的 SPEC 文件的 **%prep** 部分中使用以下宏：

    ```
    # %py3_shebang_fix SCRIPTNAME ...
    ```

    SCRIPTNAME 可以是任何文件、目录或文件和目录列表。

    因此，列出的所有文件以及列出目录中所有 **.py** 文件都会修改其解释器指令以指向 **%{python3}**。将保留原始解释器指令的现有标记，并将添加 **%{py3_shebang_flags}** 宏中定义的其他标志。您可以在 SPEC 文件中重新定义 **%{py3_shebang_flags}** 宏，以更改将要添加的标志。

  - 从 **python3-devel** 软件包应用 **pathfix.py** 脚本：

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

    您可以指定多个路径。如果 ***PATH*** 是一个目录，则 **pathfix.py** 会递归扫描与模式 **^[a-zA-Z0-9_]+\.py$** 匹配的 Python 脚本，而不仅仅是具有模糊的解释器指令。将上述命令添加到 **%prep** 的上面，或添加到 **%install** 部分的末尾。

- 修改打包的 Python 脚本，以便它们符合预期格式。为此，您也可以使用 RPM 构建进程之外的 **pathfix.py** 脚本。在 RPM 构建外运行 **pathfix.py** 时，将上例中的 **%{python3}** 替换为解释器指令的路径，如 **/usr/bin/python3** 或 **/usr/bin/python3.11**。

其它资源

- 解释器调用

第 5 章 安装 TCL/TK

5.1. TCL/TK 简介

Tcl 是一个动态编程语言，**Tk** 是一个图形用户界面(GUI)工具包。它们为开发带有图形界面的跨平台应用程序提供了强大而易用的平台。作为动态编程语言，"Tcl"为编写脚本提供了简单而灵活的语法。**tcl** 软件包为此语言和 C 库提供了解释器。您可以使用 **Tk** 作为 GUI 工具包，其为创建图形界面提供一组工具和小部件。您可以使用各种用户界面元素，如按钮、菜单、对话框、文本框及画布来画图。**Tk** 是许多动态编程语言的 GUI。

有关 Tcl/Tk 的更多信息，请参阅 Tcl/Tk manual 或 Tcl/Tk 文档网页。

5.2. 安装 TCL

通常默认安装了默认的 **Tcl** 实现。要手动安装它，请使用以下步骤。

步骤

- 要安装 **Tcl**，请使用：

```
# dnf install tcl
```

验证步骤

- 要验证系统上是否安装了 Tcl 版本，请运行解释器 **tclsh**。

```
$ tclsh
```

- 在解释器中运行这个命令：

```
% info patchlevel
8.6
```

- 您可以通过按 **Ctrl**+**C**退出解释器界面

5.3. 安装 TK

通常默认安装了默认的 **Tk** 实现。要手动安装它，请使用以下步骤。

步骤

- 要安装 **Tk**，请使用：

```
# dnf install tk
```

验证步骤

- 要验证系统上是否安装了 **Tk** 版本，请运行 window shell **wish**。您需要运行一个图形显示。

```
$ wish
```

- 在 shell 中运行这个命令：

```
% puts $tk_version
8.6
```

- 您可以通过按 **Ctrl**+**C**退出解释器界面

第 6 章 使用 PHP 脚本语言

超文本 Preprocessor (PHP)是主要用于服务器端脚本的通用脚本语言。您可以使用 Web 服务器使用 PHP 运行 PHP 代码。

6.1. 安装 PHP 脚本语言

在 RHEL 9 中，提供以下版本和格式的 PHP ：

- PHP 8.0 作为 **php** RPM 软件包
- PHP 8.1 作为 **php:8.1** 模块流
- PHP 8.2 作为 **php:8.2** 模块流

步骤

根据您的场景，完成以下步骤之一：

- 要安装 PHP 8.0，请输入：

```
# dnf install php
```

- 要使用默认配置集安装 **php:8.1** 或 **php:8.2** 模块流，请输入：

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

默认 **common** 配置文件也安装 **php-fpm** 软件包，并预配置 PHP ，以与 Apache HTTP 服务器或 nginx 一起使用。

- 要安装 **php:8.1** 或 **php:8.2** 模块流的特定配置集，请使用：

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

可用的配置文件如下：

  - **common** - 使用 Web 服务器进行服务器端脚本的默认配置文件。它包括最广泛使用的扩展。
  - **minimal** - 此配置集只安装命令行界面以用于使用 PHP 编写脚本，而无需使用 Web 服务器。
  - **devel** - 此配置文件包含来自 common 配置文件的软件包，以及用于开发用途的其他软件包。
    例如，要安装 PHP 8.1 以在没有 web 服务器的情况下使用，请使用：

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

其它资源

- 使用 DNF 工具管理软件

6.2. 通过 WEB 服务器使用 PHP 脚本语言

6.2.1. 在 Apache HTTP 服务器中使用 PHP

在 Red Hat Enterprise Linux 9 中，**Apache HTTP 服务器** 可让您将 PHP 作为 FastCGI 进程服务器运行。FastCGI Process Manager(FPM)是一种替代 PHP FastCGI 守护进程，它允许网站管理高负载。默认情况下，PHP 在 RHEL 9 中使用 FastCGI Process Manager。

您可以使用 FastCGI 进程服务器运行 PHP 代码。

先决条件

- 在您的系统上安装 PHP 脚本语言。

步骤

1. 安装 **httpd** 软件包：

```
# dnf install httpd
```

2. 启动 **Apache HTTP 服务器** ：

```
# systemctl start httpd
```

或者，如果 **Apache HTTP** 服务器已在您的系统中运行，请在安装 PHP 后重启 **httpd** 服务：

```
# systemctl restart httpd
```

3. 启动 **php-fpm** 服务：

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

4. 可选：在引导时启用这两个服务：

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

5. 要获取有关 PHP 设置的信息，请在 **/var/www/html/** 目录中创建带有以下内容的 **index.php** 文件：

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

6. 要运行 **index.php** 文件，请将浏览器指向：

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

7. 可选：如果您有特定要求，请调整配置：

- **/etc/httpd/conf/httpd.conf** - 一般的 **httpd** 配置
- **/etc/httpd/conf.d/php.conf** - **httpd**特定 PHP 配置
- **/usr/lib/systemd/system/httpd.service.d/php-fpm.conf** - 默认情况下， **php-fpm** 服务与 **httpd** 一起启动
- **/etc/php-fpm.conf** - FPM 主配置

- **/etc/php-fpm.d/www.conf** - 默认 **www** 池配置

例 6.1. 运行"Hello, World!"使用 Apache HTTP 服务器的 PHP 脚本

1. 在 **/var/www/html/** 目录中为您的项目创建一个 **hello** 目录：

```
# mkdir hello
```

2. 在 **/var/www/html/hello/** 目录中创建 **hello.php** 文件，其内容如下：

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

3. 启动 **Apache HTTP 服务器** ：

```
# systemctl start httpd
```

4. 要运行 **hello.php** 文件，请将浏览器指向：

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

因此，会显示带有 "Hello, World!" 文本的网页。

其它资源

- 设置 Apache HTTP web 服务器

6.2.2. 使用带有 nginx web 服务器的 PHP

您可以通过 **nginx** web 服务器运行 PHP 代码。

先决条件

- 在您的系统上安装 PHP 脚本语言。

步骤

1. 安装**nginx**软件包：

```
# dnf install nginx
```

2. 启动 **nginx** 服务器：

```
# systemctl start nginx
```

或者，如果 **nginx** 服务器已在您的系统中运行，请在安装 PHP 后重启 **nginx** 服务：

```
# systemctl restart nginx
```

3. 启动 **php-fpm** 服务：

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

4. 可选：在引导时启用这两个服务：

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

5. 要获取 PHP 设置的信息，请在 **/usr/share/nginx/html/** 目录中使用以下内容创建 **index.php** 文件：

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

6. 要运行 **index.php** 文件，请将浏览器指向：

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

7. 可选：如果您有特定要求，请调整配置：

- **/etc/nginx/nginx.conf** - **nginx** 主配置
- **/etc/nginx/conf.d/php-fpm.conf** - FPM 配置 **nginx**
- **/etc/php-fpm.conf** - FPM 主配置
- **/etc/php-fpm.d/www.conf** - 默认 **www** 池配置

例 6.2. 运行"Hello, World!"使用 nginx 服务器的 PHP 脚本

1. 在 **/usr/share/nginx/html/** 目录中为您的项目创建一个 **hello** 目录：

```
# mkdir hello
```

2. 在 **/usr/share/nginx/html/hello/** 目录中创建一个包含以下内容的 **hello.php** 文件：

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

3. 启动 **nginx** 服务器：

```
# systemctl start nginx
```

4. 要运行 **hello.php** 文件，请将浏览器指向：

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

因此，会显示带有 "Hello, World!" 文本的网页。

其它资源

- 设置和配置 NGINX

6.3. 使用命令行界面运行 PHP 脚本

PHP 脚本通常使用 Web 服务器运行，但也可以使用 命令行界面来运行。

先决条件

- 在您的系统上安装 PHP 脚本语言。

步骤

1. 在文本编辑器中，创建一个 ***filename*.php** 文件
将 *filename* 替换为您的文件名称。
2. 从命令行执行创建 ***filename*.php** 文件：

```
# php filename.php
```

例 6.3. 运行"Hello, World!"使用命令行界面 PHP 脚本

1. 使用文本编辑器，创建包含以下内容的 **hello.php** 文件：

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

2. 从命令行执行 **hello.php** 文件：

```
# php hello.php
```

结果会输出 "Hello, World!"。

6.4. 其它资源

- **httpd(8)** - **httpd**服务的手册页，包含其命令行选项的完整列表。
- **httpd.conf(5)** - **httpd** 配置的 man page，描述 **httpd** 配置文件的结构和位置。

- **nginx(8)** - **nginx** web 服务器的 man page，其中包含其命令行选项的完整列表和信号列表。
- **php-fpm(8)** - PHP FPM 的 man page 描述其命令行选项和配置文件的完整列表。