Red Hat Enterprise Linux 9

동적 프로그래밍 언어 설치 및 사용

Red Hat Enterprise Linux 9에서 동적 프로그래밍 언어 설치 및 사용 가이드

Last Updated: 2022-09-04

Red Hat Enterprise Linux 9 동적 프로그래밍 언어 설치 및 사용

Red Hat Enterprise Linux 9에서 동적 프로그래밍 언어 설치 및 사용 가이드

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

법적 공지

초록

이 문서에서는 Red Hat Enterprise Linux 9에 Python 및 PHP와 같은 동적 프로그래밍 언어 설치 및 사용의 기본 사항을 설명합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서에 대한 피드백에 감사드립니다. 어떻게 개선할 수 있는지 알려주십시오.

특정 문구에 대한 의견 제출

1. **Multi-page HTML** 형식으로 설명서를 보고 페이지가 완전히 로드된 후 오른쪽 상단 모서리에 **피드백** 버튼이 표시되는지 확인합니다.
2. 커서를 사용하여 주석 처리할 텍스트 부분을 강조 표시합니다.
3. 강조 표시된 텍스트 옆에 표시되는 **피드백 추가** 버튼을 클릭합니다.
4. 의견을 추가하고 **제출** 을 클릭합니다.

Bugzilla를 통해 피드백 제출(등록 필요)

1. Bugzilla 웹 사이트에 로그인합니다.
2. **버전** 메뉴에서 올바른 버전을 선택합니다.
3. **Summary** (요약) 필드에 설명 제목을 입력합니다.
4. **Description** (설명) 필드에 개선을 위한 제안을 입력합니다. 문서의 관련 부분에 대한 링크를 포함합니다.
5. **버그 제출**을 클릭합니다.

1장. PYTHON 소개

Python은 개체 지향, 필수, 기능 및 절차적 패러다임과 같은 여러 프로그래밍 패러다임을 지원하는 고급 프로그래밍 언어입니다. Python에는 동적 의미 체계가 있으며 일반 목적의 프로그래밍에 사용할 수 있습니다.

Red Hat Enterprise Linux를 사용하면 시스템 툴, 데이터 분석용 툴 또는 웹 애플리케이션을 제공하는 패키지와 같이 시스템에 설치된 많은 패키지가 Python으로 작성됩니다. 이러한 패키지를 사용하려면 **python*** 패키지가 설치되어 있어야 합니다.

1.1. PYTHON 버전

Python 3.9는 RHEL 9의 기본 **Python** 구현입니다. **Python 3.9**는 BaseOS 리포지토리의 비표준 **python3** RPM 패키지로 배포되고 일반적으로 기본적으로 설치됩니다. **Python 3.9**는 RHEL 9의 전체 라이프 사이클 동안 지원됩니다.

향후 **Python 3** 의 추가 버전은 AppStream 리포지토리를 통해 라이프사이클이 짧은 RPM 패키지로 배포됩니다. 이러한 버전은 Python 3.9와 동시에 설치할 수 있습니다.

Python 2는 RHEL 9와 함께 배포되지 않습니다.

1.2. RHEL 8 이후 PYTHON 에코시스템의 주요 차이점

이 섹션에서는 RHEL 8과 비교하여 RHEL 9의 Python 에코시스템의 주요 변경 사항에 대해 간단히 설명합니다.

버전이 없는 python 명령

python 명령(**/usr/bin/python**)의 버전이 지정되지 않은 형식인 **python-unversioned-command** 패키지에서 사용할 수 있습니다. 일부 시스템에서는 이 패키지가 기본적으로 설치되지 않습니다. **python** 명령의 버전되지 않은 양식을 수동으로 설치하려면 **dnf install /usr/bin/python** 명령을 사용합니다.

RHEL 9에서 Python 명령의 버전이 없는 형식은 기본 **Python 3.9** 버전을 가리키며 **python 3 및 python 3.9** 명령과 동일합니다.

python 명령은 대화형 세션을 위한 것입니다. 프로덕션 환경에서 **python3 또는 python 3.9**를 명시적으로 사용하는 것이 좋습니다.

dnf remove /usr/bin/python 명령을 사용하여 버전이 없는 **python** 명령을 설치 제거할 수 있습니다.

다른 python 명령이 필요한 경우 **/usr/local/bin** 또는 **~/.local/bin** 또는 Python 가상 환경에서 사용자 정의 심볼릭 링크를 만들 수 있습니다.

python3-pip 패키지의 /usr/bin/pip 와 같은 다른 몇 가지 버전이 없는 명령을 사용할 수 있습니다. RHEL 9에서는 버전이 없는 모든 명령이 기본 **Python 3.9** 버전을 가리킵니다.

아키텍처별 Python 앰퍼 니스

RHEL 9에 **구축된** 아키텍처별 Python wheel은 업스트림 아키텍처 이름 지정을 새로 준수하므로 고객은 RHEL 9 **에서** Python wheel을 빌드하고 RHEL이 아닌 시스템에 설치할 수 있습니다. RHEL의 이전 릴리스에서 **빌드된** Python wheel은 호환 가능하며 RHEL 9에 설치할 수 있습니다. 이는 아키텍처별 **순수** Python 코드가 있는 Python wheel이 아닌 각 아키텍처에 대해 빌드된 Python 확장이 **포함된** wheel에만 영향을 미칩니다.

2장. PYTHON 설치 및 사용

RHEL 9에서 **Python 3.9**는 기본 **Python** 구현입니다. 버전이 없는 **python** 명령은 기본 **Python 3.9** 버전을 가리킵니다.

2.1. PYTHON 3 설치

기본 Python 구현은 일반적으로 기본적으로 설치됩니다. 수동으로 설치하려면 다음 절차를 사용하십시오.

절차

- Python을 설치하려면 다음을 사용합니다.

```
# dnf install python3
```

검증 단계

- 시스템에 설치된 Python 버전을 확인하려면 다음 명령을 사용하십시오.

```
$ python3 --version
```

2.2. 추가 PYTHON 3 패키지 설치

python3 접두사가 지정된 패키지에는 기본 **Python 3.9** 버전의 모듈이 포함되어 있습니다.

절차

- Python에 대한 **Requests** 모듈을 설치하려면 다음을 사용합니다.

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

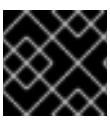
- Python에서 **pip** 패키지 설치 프로그램을 설치하려면 다음을 사용합니다.

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

2.3. 개발자를 위한 추가 PYTHON 3 툴 설치

개발자를 위한 추가 Python 도구는 CodeReady Linux Builder 리포지토리를 통해 배포됩니다.

이 리포지토리에는 예를 들어 **python3-pytest**,**python3-Cython** 패키지 등이 포함됩니다.



중요

CodeReady Linux Builder 리포지토리 및 해당 콘텐츠는 Red Hat에서 지원하지 않습니다.

리포지토리에서 패키지를 설치하려면 다음 절차를 사용하십시오.

절차

1. CodeReady Linux Builder 리포지토리를 활성화합니다.
   -

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

2. **python3-pytest** 패키지를 설치합니다.

```
# dnf install python3-pytest
```

추가 리소스

- CodeReady Linux Builder 내에서 콘텐츠를 활성화하고 사용하는 방법

2.4. PYTHON 사용

다음 절차에서는 Python 인터프리터 또는 Python 관련 명령을 실행하는 예제입니다.

사전 요구 사항

- Python이 설치되어 있는지 확인합니다.

절차

- Python 인터프리터 또는 관련 명령을 실행하려면 다음을 사용합니다. 예를 들면 다음과 같습니다.

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

3장. PYTHON 3 RPM 패키징

pip 설치 프로그램을 사용하거나 DNF 패키지 관리자를 사용하여 업스트림 PyPI 리포지토리에서 시스템에 Python 패키지를 설치할 수 있습니다. DNF는 소프트웨어에 대한 다운스트림 제어를 제공하는 RPM 패키지 형식을 사용합니다.

네이티브 Python 패키지의 패키징 형식은 Python Packaging Authority (PyPA) 사양으로 정의됩니다. 대부분의 Python 프로젝트는 패키징에 **distutils** 또는 **setuptools** 유틸리티를 사용하고 **setup.py** 파일에서 정의된 패키지 정보를 사용합니다. 그러나 기본 Python 패키지를 만들 가능성이 시간이 지남에 따라 발전했습니다. 새로운 패키징 표준에 대한 자세한 내용은 pyproject-rpm-macros 를 참조하십시오.

이 장에서는 **setup.py** 를 RPM 패키지로 사용하는 Python 프로젝트를 패키징하는 방법을 설명합니다. 이 방법은 네이티브 Python 패키지와 비교하여 다음과 같은 이점을 제공합니다.

- Python 및 비 Python 패키지에 대한 종속 항목을 사용할 수 있으며 **DNF** 패키지 관리자가 엄격하게 적용할 수 있습니다.
- 패키지를 암호화 방식으로 서명할 수 있습니다. 암호화 서명을 사용하면 RPM 패키지의 콘텐츠를 나머지 운영 체제와 검증, 통합 및 테스트할 수 있습니다.
- 빌드 프로세스 중에 테스트를 실행할 수 있습니다.

3.1. PYTHON 패키지에 대한 SPEC 파일 설명

SPEC 파일에는 **rpmbuild** 유틸리티가 RPM을 빌드하는 데 사용하는 지침이 포함되어 있습니다. 지침은 여러 섹션에 포함되어 있습니다. SPEC 파일에는 섹션이 정의된 두 가지 주요 부분이 있습니다.

- Preamble (본문에 사용되는 일련의 메타데이터 항목 포함)
- 본문(명령의 주요 부분 포함)

Python 프로젝트의 RPM SPEC 파일에는 Python이 아닌 SPEC 파일에 비해 몇 가지 구체적인 내용이 있습니다.

중요

Python 라이브러리의 RPM 패키지 이름에는 항상 **python3-** 접두사가 포함되어야 합니다.

기타 세부 사항은 **python3-pello 패키지에 대한 다음 SPEC 파일 예제에**표시됩니다. 이러한 세부 사항에 대한 설명은 예제 아래의 노트를 참조하십시오.

```
Name:           python-pello                    1
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:  python3-devel                   2

# Build dependencies needed to be specified manually
```

```
BuildRequires:  python3-setuptools

# Test dependencies needed to be specified manually
# Also runtime dependencies need to be BuildRequired manually to run tests during build
BuildRequires:  python3-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 python3-pello                                    3
Summary:        %{summary}

%description -n python3-pello %_description


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


%build
# The macro only supported projects with setup.py
%py3_build                                                   4


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


%check                                                       5
%{pytest}


# Note that there is no %%files section for the unversioned python module
%files -n python3-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 Python 프로젝트를 RPM에 패키징할 때 항상 프로젝트의 원래 이름에 **python-** 접두사를 추가합니다. 여기에서 원래 이름은 **pello** 이므로 Source RPM(SRPM)의 이름은 **python-pello** 입니다.

2 **BuildRequires** 는 이 패키지를 빌드하고 테스트하는 데 필요한 패키지를 지정합니다. **BuildRequires** 에서 항상 Python 패키지를 빌드하는 데 필요한 도구인 **python3-devel** 및 패키지하는 특정 소프트웨어에 필요한 관련 프로젝트(예: **python3-setuptools** 또는 %check 섹션에서 테스트를 실행하는 데 필요한 런타임 및 테스트 종속 항목)를 포함해야 합니다.

3 바이너리 RPM(사용자가 설치할 수 있는 패키지)의 이름을 선택할 때 현재 **python3-** 인 버전 Python 접두사를 추가합니다. 따라서 결과 바이너리 RPM의 이름은 **python3-pello** 입니다.

4 **%py3_build** 및 **%py3_install** 매크로는 각각 **setup.py build** 및 **setup.py install** 명령을 실행하여 설치 위치, 사용할 인터프리터를 지정하는 추가 인수와 함께 실행합니다.

5 **%check** 섹션에서는 패키지 프로젝트의 테스트를 실행해야 합니다. 정확한 명령은 프로젝트 자체에 따라 크게 달라지지만 **%pytest** 매크로를 사용하여 RPM에 친숙한 방식으로 **pytest** 명령을 실행할 수 있습니다. **%{python3}** 매크로에는 Python 3 인터프리터의 경로(예: **/usr/bin/python** 3)가 포함되어 있습니다. 항상 리터럴 경로가 아닌 매크로를 사용하는 것이 좋습니다.

3.2. PYTHON 3 RPM의 일반적인 매크로

SPEC 파일에서는 항상 값을 하드 코딩하지 않고 다음 *Macros for Python 3 RPMs* 테이블에 설명된 매크로를 사용합니다.

표 3.1. Python 3 RPM용 매크로

매크로	일반 정의	설명
%{python3}	/usr/bin/python3	Python 3 인터프리터
%{python3_version}	3.9	Python 3 인터프리터의 major.minor 버전
%{python3_sitelib}	/usr/lib/python3.9/site-packages	pure-Python 모듈이 설치된 위치
%{python3_sitearch}	/usr/lib64/python3.9/site-packages	아키텍처별 확장 모듈을 포함하는 모듈이 설치된 위치
%py3_build		RPM 패키지에 적합한 인수와 함께 **setup.py build** 명령 실행
%py3_install		RPM 패키지에 적합한 인수와 함께 **setup.py install** 명령 실행
%{py3_shebang_flags}	s	Python 인터프리터 지시문 매크로, **%py3_shebang_fix**의 기본 플래그 세트
%py3_shebang_fix		Python 인터프리터 지시문을 **#! %{python3}** 로 변경하고 기존 플래그(있는 경우)를 유지하고 **%{py3_shebang_flags}** 매크로에 정의된 플래그를 추가합니다.

추가 리소스

- 업스트림 문서의 Python 매크로

3.3. PYTHON RPM에 자동 생성된 종속 항목 사용

다음 절차에서는 Python 프로젝트를 RPM으로 패키징할 때 자동으로 생성된 종속 항목을 사용하는 방법을 설명합니다.

사전 요구 사항

- RPM용 SPEC 파일이 있습니다. 자세한 내용은 Python 패키지에 대한 SPEC 파일 설명을 참조하십시오.

절차

1. 업스트림 제공 메타데이터가 포함된 다음 디렉터리 중 하나가 결과 RPM에 포함되어 있는지 확인합니다.
   - **.dist-info**
   - **.egg-info**
     RPM 빌드 프로세스는 다음과 같이 이러한 디렉터리에서 제공하는 가상 **pythonX.Ydist** 를 자동으로 생성합니다.

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

     그런 다음 Python 종속성 생성기는 업스트림 메타데이터를 읽고 생성된 **pythonX.Ydist** 가상 기능을 사용하여 각 RPM 패키지에 대한 런타임 요구 사항을 생성합니다. 예를 들어 생성된 요구 사항 태그는 다음과 같을 수 있습니다.

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

2. 생성된 요구 사항을 검사합니다.
3. 생성된 요구 중 일부를 제거하려면 다음 방법 중 하나를 사용합니다.
   a. SPEC 파일의 **%prep** 섹션에서 업스트림 제공 메타데이터를 수정합니다.
   b. 업스트림 설명서에 설명된 종속성을 자동 필터링하여 사용합니다.
4. 자동 종속성 생성기를 비활성화하려면 기본 패키지의 **%description** 선언 위에 **%{?python_disable_dependency_generator}** 매크로를 포함합니다.

추가 리소스

- 자동 생성된 종속 항목

4장. PYTHON 스크립트에서 인터프리터 지시문 처리

Red Hat Enterprise Linux 9에서는 실행 가능한 Python 스크립트는 최소 주요 Python 버전에서 명시적으로 지정하는 인터프리터 지시문(hashbangs 또는 shebangs라고도 함)을 사용해야 합니다. 예를 들어 다음과 같습니다.

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

RPM 패키지를 빌드할 때 **/usr/lib/rpm/redhat/brp-mangle-shebangs** BRP(Buildroot 정책) 스크립트를 자동으로 실행하고 모든 실행 파일에서 인터프리터 지시문을 수정하려고 시도합니다.

BRP 스크립트는 다음과 같은 모호한 인터프리터 지시문을 사용하여 Python 스크립트를 시작할 때 오류를 생성합니다.

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

또는

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

4.1. PYTHON 스크립트에서 인터프리터 지시문 수정

다음 절차에 따라 RPM 빌드 시 빌드 오류가 발생하는 Python 스크립트에서 인터프리터 지시문을 수정합니다.

사전 요구 사항

- Python 스크립트의 인터프리터 지시문 중 일부는 빌드 오류가 발생합니다.

절차

- 인터프리터 지시문을 수정하려면 다음 작업 중 하나를 완료합니다.
  - SPEC 파일의 **%prep** 섹션에서 다음 매크로를 사용합니다.

    ```
    # %py3_shebang_fix SCRIPTNAME …
    ```

    SCRIPTNAME 은 모든 파일, 디렉터리 또는 파일 및 디렉터리 목록일 수 있습니다.

    결과적으로 나열된 디렉터리의 모든 파일 및 모든 **.py** 파일은 **%{python3}** 을 가리키도록 인터프리터 지시문을 수정합니다. 원래 인터프리터 지시문의 기존 플래그는 보존되고 **%{py3_shebang_flags}** 매크로에 정의된 추가 플래그가 추가됩니다. SPEC 파일에서 **%{py3_shebang_flags}** 매크로를 재정의하여 추가할 플래그를 변경할 수 있습니다.

  - **python3-devel** 패키지에서 **pathfix.py** 스크립트를 적용합니다.

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

    여러 경로를 지정할 수 있습니다. ***PATH*** 가 디렉터리인 경우 **pathfix.py** 는 모호한 인터프리터 지시문이 있는 것뿐만 아니라 **^[a-zA-Z0-9_]+\.py$** 패턴과 일치하는 Python 스크립트를 반복적으로 검사합니다. 위의 명령을 **%prep** 섹션에 추가하거나 **%install** 섹션의 끝에 추가합니다.

- 패키지된 Python 스크립트를 수정하여 예상 형식을 준수하도록 합니다. 이를 위해 RPM 빌드 프로세스 외부의 **pathfix.py** 스크립트도 사용할 수 있습니다. RPM 빌드 외부에서 **pathfix.py** 를 실행하는 경우 위의 예에서 **%{python3}** 을 **/usr/bin/python3** 과 같은 인터프리터 지시문 경로로 교체합니다.

추가 리소스

- 인터프리터 호출

5장. PHP 스크립팅 언어 사용

하이퍼텍스트 Preprocessor (PHP)는 주로 서버 측 스크립팅에 사용되는 범용 스크립팅 언어이며, 웹 서버를 사용하여 PHP 코드를 실행할 수 있습니다.

5.1. PHP 스크립팅 언어 설치

이 섹션에서는 PHP를 설치하는 방법을 설명합니다.

절차

- PHP를 설치하려면 다음을 사용합니다.

```
# dnf install php
```

5.2. 웹 서버에서 PHP 스크립팅 언어 사용

5.2.1. Apache HTTP Server에서 PHP 사용

Red Hat Enterprise Linux 9에서 **Apache HTTP Server** 를 사용하면 PHP를 FastCGI 프로세스 서버로 실행할 수 있습니다. FastCGI Process Manager (FPM)는 웹 사이트가 높은 부하를 관리 할 수있는 대체 PHP FastCGI 데몬입니다. PHP는 RHEL 9에서 기본적으로 FastCGI Process Manager를 사용합니다.

이 섹션에서는 FastCGI 프로세스 서버를 사용하여 PHP 코드를 실행하는 방법을 설명합니다.

사전 요구 사항

- PHP 스크립팅 언어가 시스템에 설치되어 있습니다.

절차

1. **httpd** 패키지를 설치합니다.

```
# dnf install httpd
```

2. **Apache HTTP Server** 를 시작합니다.

```
# systemctl start httpd
```

   또는 시스템에서 **Apache HTTP Server** 가 이미 실행 중인 경우 PHP를 설치한 후 **httpd** 서비스를 다시 시작합니다.

```
# systemctl restart httpd
```

3. **php-fpm** 서비스를 시작합니다.

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

4. 선택 사항: 두 서비스 모두 부팅 시 시작되도록 활성화합니다.

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

5. PHP 설정에 대한 정보를 얻으려면 **/var/www/html/** 디렉터리에 다음 콘텐츠를 사용하여 **index.php** 파일을 생성합니다.

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

6. **index.php** 파일을 실행하려면 브라우저가 다음을 가리키도록 합니다.

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

7. 선택 사항: 특정 요구 사항이 있는 경우 구성을 조정합니다.
   - **/etc/httpd/conf/httpd.conf** - generic **httpd** configuration
   - **/etc/httpd/conf.d/php.conf** - **httpd**에 대한 PHP 특정 설정
   - **/usr/lib/systemd/system/httpd.service.d/php-fpm.conf** - 기본적으로 **php-fpm** 서비스가 **httpd**로 시작됩니다.
   - **/etc/php-fpm.conf** - FPM 메인 구성
   - **/etc/php-fpm.d/www.conf** - 기본 **WWW** 풀 구성

예 5.1. "Hello, World!"를 실행합니다. Apache HTTP Server를 사용한 PHP 스크립트

1. **/var/www/html/** 디렉터리에 프로젝트의 **hello** 디렉터리를 만듭니다.

```
# mkdir hello
```

2. 다음 콘텐츠를 사용하여 **/var/www/html/hello/** 디렉터리에 **hello.php** 파일을 생성합니다.

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

3. **Apache HTTP Server** 를 시작합니다.

```
# systemctl start httpd
```

4. **hello.php** 파일을 실행하려면 브라우저가 다음을 가리키도록 합니다.

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

결과적으로 "Hello, World!" 텍스트가 있는 웹 페이지가 표시됩니다.

추가 리소스

- Apache HTTP 웹 서버 설정

5.2.2. nginx 웹 서버에서 PHP 사용

이 섹션에서는 **nginx** 웹 서버를 통해 PHP 코드를 실행하는 방법을 설명합니다.

사전 요구 사항

- PHP 스크립팅 언어가 시스템에 설치되어 있습니다.

절차

1. **nginx** 패키지를 설치합니다.

```
# dnf install nginx
```

2. **nginx** 서버를 시작합니다.

```
# systemctl start nginx
```

또는 **nginx** 서버가 시스템에서 이미 실행 중인 경우 PHP를 설치한 후 **nginx** 서비스를 다시 시작합니다.

```
# systemctl restart nginx
```

3. **php-fpm** 서비스를 시작합니다.

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

4. 선택 사항: 두 서비스 모두 부팅 시 시작되도록 활성화합니다.

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

5. PHP 설정에 대한 정보를 얻으려면 **/usr/share/nginx/html/** 디렉터리에 다음 콘텐츠를 사용하여 **index.php** 파일을 만듭니다.

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

6. **index.php** 파일을 실행하려면 브라우저가 다음을 가리키도록 합니다.

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

7. 선택 사항: 특정 요구 사항이 있는 경우 구성을 조정합니다.
   - **/etc/nginx/nginx.conf** - **nginx** 기본 구성
   - **/etc/nginx/conf.d/php-fpm.conf** - **nginx**에 대한 FPM 구성
   - **/etc/php-fpm.conf** - FPM 메인 구성
   - **/etc/php-fpm.d/www.conf** - 기본 **WWW** 풀 구성

예 5.2. "Hello, World!"를 실행합니다. nginx 서버를 사용하는 PHP 스크립트

1. **/usr/share/nginx/html/** 디렉터리에 프로젝트의 **hello** 디렉터리를 만듭니다.

```
# mkdir hello
```

2. 다음 콘텐츠를 사용하여 **/usr/share/nginx/html/hello/** 디렉터리에 **hello.php** 파일을 만듭니다.

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

3. **nginx** 서버를 시작합니다.

```
# systemctl start nginx
```

4. **hello.php** 파일을 실행하려면 브라우저가 다음을 가리키도록 합니다.

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

결과적으로 "Hello, World!" 텍스트가 있는 웹 페이지가 표시됩니다.

추가 리소스

- NGINX 설정 및 구성

5.3. 명령줄 인터페이스를 사용하여 PHP 스크립트 실행

PHP 스크립트는 일반적으로 웹 서버를 사용하여 실행되지만 명령줄 인터페이스를 사용하여 실행할 수도 있습니다.

사전 요구 사항

- PHP 스크립팅 언어가 시스템에 설치되어 있습니다.

절차

1. 텍스트 편집기에서 파일 이름 . ***php*** 파일 생성
   파일 이름을 파일 이름으로 바꿉니다.

2. 명령줄에서 생성된 ***파일 이름*.php** 파일을 실행합니다.

```
# php filename.php
```

-

예 5.3. "Hello, World!"를 실행합니다. 명령줄 인터페이스를 사용한 PHP 스크립트

1. 텍스트 편집기를 사용하여 다음 내용으로 **hello.php** 파일을 생성합니다.

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

2. 명령줄에서 **hello.php** 파일을 실행합니다.

```
# php hello.php
```

결과적으로 "Hello, World!"가 출력됩니다.

5.4. 추가 리소스

- **httpd(8)** - 명령줄 옵션의 전체 목록을 포함하는 **httpd** 서비스의 도움말 페이지입니다.
- **httpd.conf(5)** - **httpd** 구성의 도움말 페이지로, **httpd** 구성 파일의 구조와 위치를 설명합니다.
- **nginx(8)** - 명령줄 옵션의 전체 목록 및 신호 목록이 포함된 **nginx** 웹 서버의 도움말 페이지.
- **PHP-fpm(8)** - PHP FPM의 설명서 페이지에서 명령줄 옵션 및 구성 파일의 전체 목록을 설명합니다.