Red Hat Enterprise Linux 9

Installation et utilisation de langages de programmation dynamiques

Installation et utilisation de Python et PHP dans Red Hat Enterprise Linux 9

Last Updated: 2024-02-15

Red Hat Enterprise Linux 9 Installation et utilisation de langages de programmation dynamiques

Installation et utilisation de Python et PHP dans Red Hat Enterprise Linux 9

Notice légale

Résumé

Installer et utiliser Python 3, empaqueter les RPM de Python 3 et apprendre à gérer les directives de l'interpréteur dans les scripts Python. Installer le langage de script PHP, utiliser PHP avec le serveur HTTP Apache ou le serveur web ngninx, et exécuter un script PHP à partir d'une interface en ligne de commande.

Table des matières

RENDRE L'OPEN SOURCE PLUS INCLUSIF

Red Hat s'engage à remplacer les termes problématiques dans son code, sa documentation et ses propriétés Web. Nous commençons par ces quatre termes : master, slave, blacklist et whitelist. En raison de l'ampleur de cette entreprise, ces changements seront mis en œuvre progressivement au cours de plusieurs versions à venir. Pour plus de détails, voir le message de notre directeur technique Chris Wright.

FOURNIR UN RETOUR D'INFORMATION SUR LA DOCUMENTATION DE RED HAT

Nous apprécions vos commentaires sur notre documentation. Faites-nous savoir comment nous pouvons l'améliorer.

Soumettre des commentaires sur des passages spécifiques

1. Consultez la documentation au format **Multi-page HTML** et assurez-vous que le bouton **Feedback** apparaît dans le coin supérieur droit après le chargement complet de la page.
2. Utilisez votre curseur pour mettre en évidence la partie du texte que vous souhaitez commenter.
3. Cliquez sur le bouton **Add Feedback** qui apparaît près du texte en surbrillance.
4. Ajoutez vos commentaires et cliquez sur **Submit**.

Soumettre des commentaires via Bugzilla (compte requis)

1. Connectez-vous au site Web de Bugzilla.
2. Sélectionnez la version correcte dans le menu **Version**.
3. Saisissez un titre descriptif dans le champ **Summary**.
4. Saisissez votre suggestion d'amélioration dans le champ **Description**. Incluez des liens vers les parties pertinentes de la documentation.
5. Cliquez sur **Submit Bug**.

CHAPITRE 1. INTRODUCTION À PYTHON

Python est un langage de programmation de haut niveau qui prend en charge plusieurs paradigmes de programmation, tels que les paradigmes orientés objet, impératifs, fonctionnels et procéduraux. Python possède une sémantique dynamique et peut être utilisé pour la programmation générale.

Avec Red Hat Enterprise Linux, de nombreux paquetages installés sur le système, tels que les paquetages fournissant des outils système, des outils d'analyse de données ou des applications web, sont écrits en Python. Pour utiliser ces paquets, les paquets **python*** doivent être installés.

1.1. VERSIONS DE PYTHON

Python 3.9 est l'implémentation par défaut de **Python** dans RHEL 9. **Python 3.9** est distribué dans un paquet RPM **python3** non modulaire dans le dépôt BaseOS et est généralement installé par défaut. **Python 3.9** sera pris en charge pendant toute la durée de vie de RHEL 9.

Des versions supplémentaires de **Python 3** sont distribuées sous forme de paquets RPM non modulaires avec un cycle de vie plus court via le dépôt AppStream dans les versions mineures de RHEL 9. Vous pouvez installer ces versions supplémentaires de **Python 3** en parallèle avec Python 3.9.

Python 2 n'est pas distribué avec RHEL 9.

Tableau 1.1. Versions de Python dans RHEL 9

Version	Paquet à installer	Exemples de commandes	Disponible depuis	Cycle de vie
Python 3.9	**python3**	**python3**, **pip3**	RHEL 9.0	rHEL 9 complet
Python 3.11	**python3.11**	**python3.11**, **pip3.11**	RHEL 9.2	plus courte

Pour plus de détails sur la durée de l'assistance, voir Red Hat Enterprise Linux Life Cycle et Red Hat Enterprise Linux Application Streams Life Cycle.

1.2. PRINCIPALES DIFFÉRENCES DANS L'ÉCOSYSTÈME PYTHON DEPUIS RHEL 8

Cette section résume les principaux changements apportés à l'écosystème Python dans RHEL 9 par rapport à RHEL 8.

La commande non versionnée python

La forme non versionnée de la commande **python** (**/usr/bin/python**) est disponible dans le paquetage **python-unversioned-command**. Sur certains systèmes, ce paquetage n'est pas installé par défaut. Pour installer manuellement la forme non versionnée de la commande **python**, utilisez la commande **dnf install /usr/bin/python**.

Dans RHEL 9, la forme non versionnée de la commande **python** pointe vers la version par défaut **Python 3.9** et est équivalente aux commandes **python3** et **python3.9**. Dans RHEL 9, la commande unversioned ne peut pas être configurée pour pointer vers une version différente de **Python 3.9**.

La commande **python** est destinée aux sessions interactives. En production, il est recommandé d'utiliser explicitement **python3**, **python3.9**, ou **python3.11**.

Vous pouvez désinstaller la commande non versionnée **python** en utilisant la commande **dnf remove /usr/bin/python**.

Si vous avez besoin d'une commande Python différente, vous pouvez créer des liens symboliques personnalisés dans **/usr/local/bin** ou **~/.local/bin** ou dans un environnement virtuel Python.

Plusieurs autres commandes non versionnées sont disponibles, telles que **/usr/bin/pip** dans le paquetage **python3-pip**. Dans RHEL 9, toutes les commandes non versionnées pointent vers la version par défaut **Python 3.9**.

Python spécifique à l'architecture wheels

Python **wheels** spécifique à l'architecture construit sur RHEL 9 adhère désormais à la dénomination de l'architecture en amont, ce qui permet aux clients de construire leur Python **wheels** sur RHEL 9 et de l'installer sur des systèmes non RHEL. Python **wheels** construit sur des versions antérieures de RHEL est compatible avec les versions ultérieures et peut être installé sur RHEL 9. Notez que cela n'affecte que **wheels** contenant des extensions Python, qui sont construites pour chaque architecture, et non Python **wheels** avec du code Python pur, qui n'est pas spécifique à l'architecture.

CHAPITRE 2. INSTALLATION ET UTILISATION DE PYTHON

Dans RHEL 9, **Python 3.9** est l'implémentation par défaut de **Python**. Depuis RHEL 9.2, **Python 3.11** est disponible sous la forme de la suite de paquets **python3.11**.

La commande non versionnée **python** renvoie à la version par défaut **Python 3.9**.

2.1. INSTALLATION DE PYTHON 3

L'implémentation par défaut de **Python** est généralement installée par défaut. Pour l'installer manuellement, utilisez la procédure suivante.

Procédure

- Pour installer **Python 3.9**, utilisez :

```
# dnf install python3
```

- Pour installer **Python 3.11**, utilisez :

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

Verification steps

- Pour vérifier la version de **Python** installée sur votre système, utilisez l'option **--version** avec la commande **python** spécifique à la version de **Python** dont vous avez besoin.
- Pour **Python 3.9**:

```
$ python3 --version
```

- Pour **Python 3.11**:

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

2.2. INSTALLATION DE PAQUETS PYTHON 3 SUPPLÉMENTAIRES

Les paquets dont le préfixe est **python3-** contiennent des modules complémentaires pour la version par défaut **Python 3.9**. Les paquets préfixés par **python3.11-** contiennent des modules complémentaires pour **Python 3.11**.

Procédure

- Pour installer le module **Requests** pour **Python 3.9**, utilisez :

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

- Pour installer le programme d'installation du paquet **pip** à partir de **Python 3.9**, utiliser :

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

- Pour installer le programme d'installation du paquet **pip** à partir de **Python 3.11**, utiliser :

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

Ressources supplémentaires

- Documentation en amont sur les modules complémentaires Python

2.3. INSTALLATION D'OUTILS PYTHON 3 SUPPLÉMENTAIRES POUR LES DÉVELOPPEURS

D'autres outils **Python** destinés aux développeurs sont distribués principalement par le biais du dépôt CodeReady Linux Builder (CRB).

Le paquet **python3-pytest** et ses dépendances sont disponibles dans le dépôt AppStream.

Le référentiel CRB contient, par exemple, les paquets suivants :

- **python3*-idle**
- **python3*-debug**
- **python3*-Cython**
- **python3.11-pytest** et ses dépendances.

IMPORTANT

Le contenu du référentiel CodeReady Linux Builder n'est pas pris en charge par Red Hat.

Pour installer des paquets à partir du référentiel CRB, utilisez la procédure suivante.

Procédure

1. Activer le référentiel CodeReady Linux Builder :

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

2. Installez le paquetage **python3*-Cython**:
   - Pour **Python 3.9**:

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

   - Pour **Python 3.11**:

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

Ressources supplémentaires

- Comment activer et utiliser le contenu de CodeReady Linux Builder ?
- Manifeste de l'emballage

2.4. UTILISATION DE PYTHON

La procédure suivante contient des exemples d'exécution de l'interpréteur **Python** ou des commandes liées à **Python**.

Conditions préalables

- Assurez-vous que **Python** est installé.
- Si vous souhaitez télécharger et installer des applications tierces pour **Python 3.11**, installez le paquet **python3.11-pip**.

Procédure

- Pour lancer l'interpréteur **Python 3.9** ou les commandes correspondantes, utilisez, par exemple, la commande suivante

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

- Pour lancer l'interpréteur **Python 3.11** ou les commandes correspondantes, utilisez, par exemple, la commande suivante

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

CHAPITRE 3. EMBALLAGE DES RPMS DE PYTHON 3

Vous pouvez installer des paquets Python sur votre système soit à partir du dépôt PyPI en amont en utilisant le programme d'installation **pip**, soit en utilisant le gestionnaire de paquets DNF. DNF utilise le format de paquetage RPM, qui offre plus de contrôle en aval sur le logiciel.

Le format d'empaquetage des paquets Python natifs est défini par les spécifications de la Python Packaging Authority (PyPA). La plupart des projets Python utilisent les utilitaires **distutils** ou **setuptools** pour l'empaquetage et définissent des informations sur les paquets dans le fichier **setup.py**. Cependant, les possibilités de créer des paquets Python natifs ont évolué au fil du temps. Pour plus d'informations sur les normes d'empaquetage émergentes, voir pyproject-rpm-macros.

Ce chapitre décrit comment empaqueter un projet Python qui utilise **setup.py** dans un paquetage RPM. Cette approche présente les avantages suivants par rapport aux paquets Python natifs :

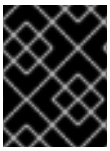
- Les dépendances sur les paquets Python et non-Python sont possibles et strictement appliquées par le gestionnaire de paquets **DNF**.
- Vous pouvez signer les paquets de manière cryptographique. La signature cryptographique permet de vérifier, d'intégrer et de tester le contenu des paquets RPM avec le reste du système d'exploitation.
- Vous pouvez exécuter des tests pendant le processus de construction.

3.1. DESCRIPTION DU FICHIER SPEC POUR UN PAQUETAGE PYTHON

Un fichier SPEC contient des instructions que l'utilitaire **rpmbuild** utilise pour construire un RPM. Les instructions sont incluses dans une série de sections. Un fichier SPEC se compose de deux parties principales dans lesquelles les sections sont définies :

- Préambule (contient une série de métadonnées utilisées dans le corps du texte)
- Corps (contient la partie principale des instructions)

Un fichier RPM SPEC pour les projets Python présente certaines spécificités par rapport aux fichiers RPM SPEC non-Python.



IMPORTANT

Le nom de tout paquetage RPM d'une bibliothèque Python doit toujours inclure le préfixe **python3-** ou **python3.11-**.

D'autres particularités sont présentées dans l'exemple de fichier SPEC suivant pour le paquet **python3*-pello**. Pour la description de ces spécificités, voir les notes sous l'exemple.

```
%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. En définissant la macro **python3_pkgversion**, vous définissez la version de Python pour laquelle ce paquet sera construit. Pour construire pour la version 3.9 de Python, mettez la macro à sa valeur

2. Lorsque vous compilez un projet Python dans un RPM, ajoutez toujours le préfixe **python-** au nom original du projet. Le nom original est ici **pello** et, par conséquent, le **name of the Source RPM (SRPM)** est **python-pello**.

3. **BuildRequires** spécifie les paquets nécessaires pour construire et tester ce paquet. Dans **BuildRequires**, incluez toujours les éléments fournissant les outils nécessaires à la construction des paquets Python : **python3-devel** (ou **python3.11-devel**) et les projets pertinents nécessaires au logiciel spécifique que vous empaquetez, par exemple, **python3-setuptools** (ou **python3.11-setuptools**) ou les dépendances d'exécution et de test nécessaires à l'exécution des tests dans la section **\feck**.

4. Lorsque vous choisissez un nom pour le RPM binaire (le paquet que les utilisateurs pourront installer), ajoutez un préfixe de version de Python. Utilisez le préfixe **python3-** pour Python 3.9 par défaut ou le préfixe **python3.11-** pour Python 3.11. Vous pouvez utiliser la macro **%{python3_pkgversion}**, qui évalue **3** pour la version 3.9 de Python par défaut, à moins que vous ne lui attribuiez une version explicite, par exemple **3.11** (voir note de bas de page 1).

5. Les macros **%py3_build** et **%py3_install** exécutent respectivement les commandes **setup.py build** et **setup.py install**, avec des arguments supplémentaires pour spécifier les emplacements d'installation, l'interpréteur à utiliser et d'autres détails.

6. La section **\feck** doit exécuter les tests du projet empaqueté. La commande exacte dépend du projet lui-même, mais il est possible d'utiliser la macro **%pytest** pour exécuter la commande **pytest** d'une manière adaptée à RPM.

3.2. MACROS COMMUNES POUR LES RPMS PYTHON 3

Dans un fichier SPEC, utilisez toujours les macros décrites dans le tableau suivant *Macros for Python 3 RPMs* plutôt que de coder en dur leurs valeurs. Vous pouvez redéfinir la version de Python 3 utilisée dans ces macros en définissant la macro **python3_pkgversion** au sommet de votre fichier SPEC (voir Section 3.1, « Description du fichier SPEC pour un paquetage Python »). Si vous définissez la macro **python3_pkgversion**, les valeurs des macros décrites dans le tableau suivant refléteront la version de Python 3 spécifiée.

Tableau 3.1. Macros pour les RPM de Python 3

Macro	Définition normale	Description
%{python3_pkgversion}	3	La version de Python utilisée par toutes les autres macros. Peut être redéfinie à **3.11** pour utiliser Python 3.11
%{python3}	/usr/bin/python3	L'interpréteur Python 3
%{python3_version}	3.9	La version majeure et mineure de l'interpréteur Python 3
%{python3_sitelib}	/usr/lib/python3.9/site-packages	L'emplacement où les modules Pure-Python sont installés

Macro	Définition normale	Description
%{python3_sitearch}	/usr/lib64/python3.9/site-packages	Emplacement où sont installés les modules contenant des modules d'extension spécifiques à l'architecture
%py3_build		Exécute la commande **setup.py build** avec des arguments adaptés à un paquetage RPM
%py3_install		Exécute la commande **setup.py install** avec des arguments adaptés à un paquetage RPM
%{py3_shebang_flags}	s	L'ensemble des drapeaux par défaut pour la macro des directives de l'interpréteur Python, **%py3_shebang_fix**
%py3_shebang_fix		Modifie les directives de l'interpréteur Python en **#! %{python3}**, préserve les drapeaux existants (s'ils existent) et ajoute les drapeaux définis dans la macro **%{py3_shebang_flags}**

Ressources supplémentaires

- Macros Python dans la documentation en amont

3.3. UTILISATION DES DÉPENDANCES GÉNÉRÉES AUTOMATIQUEMENT POUR LES RPM PYTHON

La procédure suivante décrit comment utiliser les dépendances générées automatiquement lors de l'empaquetage d'un projet Python sous forme de RPM.

Conditions préalables

- Il existe un fichier SPEC pour le RPM. Pour plus d'informations, voir Description du fichier SPEC pour un paquetage Python.

Procédure

1. Assurez-vous que l'un des répertoires suivants contenant les métadonnées fournies en amont est inclus dans le RPM résultant :
   - **.dist-info**
   - **.egg-info**
     Le processus de construction du RPM génère automatiquement des versions virtuelles de **pythonX.Ydist** à partir de ces répertoires, par exemple :

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

     Le générateur de dépendances Python lit ensuite les métadonnées en amont et génère des

exigences d'exécution pour chaque paquet RPM en utilisant les **pythonX.Ydist** virtual provides générés. Par exemple, une balise d'exigences générée peut ressembler à ce qui suit :

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

2. Inspecter les demandes générées.
3. Pour supprimer certaines des exigences générées, utilisez l'une des approches suivantes :
   a. Modifier les métadonnées fournies en amont dans la section **%prep** du fichier SPEC.
   b. Utiliser le filtrage automatique des dépendances décrit dans la documentation en amont.
4. Pour désactiver le générateur automatique de dépendances, incluez la macro **%{?python_disable_dependency_generator}** au-dessus de la déclaration **Þscription** du paquet principal.

Ressources supplémentaires

- Dépendances générées automatiquement

CHAPITRE 4. GESTION DES DIRECTIVES DE L'INTERPRÉTEUR DANS LES SCRIPTS PYTHON

Dans Red Hat Enterprise Linux 9, les scripts Python exécutables sont censés utiliser des directives d'interpréteur (également connues sous le nom de hashbangs ou shebangs) qui spécifient explicitement au minimum la version majeure de Python. Par exemple :

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

Le script **/usr/lib/rpm/redhat/brp-mangle-shebangs** buildroot policy (BRP) est exécuté automatiquement lors de la construction de tout paquetage RPM et tente de corriger les directives de l'interpréteur dans tous les fichiers exécutables.

Le script BRP génère des erreurs lorsqu'il rencontre un script Python avec une directive d'interprétation ambiguë, telle que :

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

ou

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

4.1. MODIFIER LES DIRECTIVES DE L'INTERPRÉTEUR DANS LES SCRIPTS PYTHON

Utilisez la procédure suivante pour modifier les directives de l'interpréteur dans les scripts Python qui provoquent des erreurs de compilation au moment de la compilation du RPM.

Conditions préalables

- Certaines directives de l'interpréteur dans vos scripts Python provoquent une erreur de compilation.

Procédure

- Pour modifier les directives de l'interpréteur, effectuez l'une des tâches suivantes :
  - Utilisez la macro suivante dans la section **%prep** de votre fichier SPEC :

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

    SCRIPTNAME peut être un fichier, un répertoire ou une liste de fichiers et de répertoires.

    En conséquence, tous les fichiers listés et tous les fichiers **.py** dans les répertoires listés verront leurs directives d'interprétation modifiées pour pointer vers **%{python3}**. Les drapeaux existants de la directive d'interprétation originale seront préservés et des drapeaux supplémentaires définis dans la macro **%{py3_shebang_flags}** seront ajoutés. Vous pouvez redéfinir la macro **%{py3_shebang_flags}** dans votre fichier SPEC pour modifier les drapeaux qui seront ajoutés.

- Appliquer le script **pathfix.py** du paquet **python3-devel**:

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

  Vous pouvez spécifier plusieurs chemins. Si un ***PATH*** est un répertoire, **pathfix.py** recherche de manière récursive tous les scripts Python correspondant au modèle **^[a-zA-Z0-9_] \.py$**, et pas seulement ceux dont la directive d'interprétation est ambiguë. Ajoutez la commande ci-dessus à la section **%prep** ou à la fin de la section **%install**.

- Modifiez les scripts Python fournis afin qu'ils soient conformes au format attendu. À cette fin, vous pouvez également utiliser le script **pathfix.py** en dehors du processus de compilation du RPM. Lorsque vous exécutez **pathfix.py** en dehors d'une compilation RPM, remplacez **%{python3}** de l'exemple précédent par un chemin d'accès à la directive de l'interpréteur, tel que **/usr/bin/python3** ou **/usr/bin/python3.11**.

Ressources supplémentaires

- Invocation de l'interprète

CHAPITRE 5. UTILISATION DU LANGAGE DE SCRIPT PHP

Hypertext Preprocessor (PHP) est un langage de script universel principalement utilisé pour les scripts côté serveur, ce qui vous permet d'exécuter le code PHP à l'aide d'un serveur web.

Dans RHEL 9, PHP est disponible dans les versions et formats suivants :

- PHP 8.0 en tant que paquetage RPM **php**
- PHP 8.1 en tant que flux de modules **php:8.1**

5.1. INSTALLATION DU LANGAGE DE SCRIPT PHP

Cette section décrit comment installer PHP.

Procédure

- Pour installer PHP 8.0, utilisez :

```
# dnf install php
```

- Pour installer le flux du module **php:8.1** avec le profil par défaut, utilisez :

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

  Le profil par défaut **common** installe également le paquetage **php-fpm**, et préconfigure PHP pour une utilisation avec le serveur HTTP Apache ou nginx.

- Pour installer un profil spécifique du flux de modules **php:8.1**, utilisez :

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

  Les profils disponibles sont les suivants :

- **common** - Le profil par défaut pour les scripts côté serveur utilisant un serveur web. Il comprend les extensions les plus utilisées.
- **minimal** - Ce profil n'installe que l'interface en ligne de commande pour l'écriture de scripts avec PHP sans utiliser de serveur web.
- **devel** - Ce profil comprend des paquets du profil commun et des paquets supplémentaires à des fins de développement.
  Par exemple, pour installer PHP 8.1 pour une utilisation sans serveur web, utilisez :

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

Ressources supplémentaires

- Gérer les logiciels avec l'outil DNF

5.2. UTILISATION DU LANGAGE DE SCRIPT PHP AVEC UN SERVEUR WEB

5.2.1. Utilisation de PHP avec le serveur HTTP Apache

Dans Red Hat Enterprise Linux 9, le site **Apache HTTP Server** vous permet d'exécuter PHP en tant que serveur de processus FastCGI. FastCGI Process Manager (FPM) est un démon PHP FastCGI alternatif qui permet à un site web de gérer des charges élevées. PHP utilise FastCGI Process Manager par défaut dans RHEL 9.

Cette section décrit comment exécuter le code PHP en utilisant le serveur de processus FastCGI.

Conditions préalables

- Le langage de script PHP est installé sur votre système.

Procédure

1. Installez le paquetage **httpd**:

```
# dnf install httpd
```

2. Démarrer le site **Apache HTTP Server**:

```
# systemctl start httpd
```

   Ou, si le service **Apache HTTP Server** fonctionne déjà sur votre système, redémarrez le service **httpd** après avoir installé PHP :

```
# systemctl restart httpd
```

3. Démarrez le service **php-fpm**:

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

4. Facultatif : Activez les deux services pour qu'ils démarrent au moment du démarrage :

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

5. Pour obtenir des informations sur vos paramètres PHP, créez le fichier **index.php** avec le contenu suivant dans le répertoire **/var/www/html/**:

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

6. Pour exécuter le fichier **index.php**, pointez le navigateur sur :

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

7. En option : Ajustez la configuration si vous avez des besoins spécifiques :
   - **/etc/httpd/conf/httpd.conf** - configuration générique **httpd**
   - **/etc/httpd/conf.d/php.conf** - Configuration spécifique à PHP pour **httpd**
   - **/usr/lib/systemd/system/httpd.service.d/php-fpm.conf** - par défaut, le service **php-fpm** est démarré avec **httpd**

- **/etc/php-fpm.conf** - Configuration principale du FPM
- **/etc/php-fpm.d/www.conf** - configuration par défaut du pool **www**

Exemple 5.1. Exécution d'un script PHP "Hello, World!" En utilisant le serveur HTTP Apache

1. Créez un répertoire **hello** pour votre projet dans le répertoire **/var/www/html/**:

```
# mkdir hello
```

2. Créez un fichier **hello.php** dans le répertoire **/var/www/html/hello/** avec le contenu suivant :

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

3. Démarrer le site **Apache HTTP Server**:

```
# systemctl start httpd
```

4. Pour exécuter le fichier **hello.php**, pointez le navigateur sur :

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

Le résultat est l'affichage d'une page web avec le texte "Hello, World !

Ressources supplémentaires

- Configuration du serveur web Apache HTTP

5.2.2. Utiliser PHP avec le serveur web nginx

Cette section décrit comment exécuter du code PHP via le serveur web **nginx**.

Conditions préalables

- Le langage de script PHP est installé sur votre système.

Procédure

1. Installez le paquetage **nginx**:

```
# dnf install nginx
```

2. Démarrez le serveur **nginx**:

```
# systemctl start nginx
```

Ou, si le serveur **nginx** fonctionne déjà sur votre système, redémarrez le service **nginx** après avoir installé PHP :

```
# systemctl restart nginx
```

3. Démarrez le service **php-fpm**:

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

4. Facultatif : Activez les deux services pour qu'ils démarrent au moment du démarrage :

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

5. Pour obtenir des informations sur vos paramètres PHP, créez le fichier **index.php** avec le contenu suivant dans le répertoire **/usr/share/nginx/html/**:

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

6. Pour exécuter le fichier **index.php**, pointez le navigateur sur :

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

7. En option : Ajustez la configuration si vous avez des besoins spécifiques :

- **/etc/nginx/nginx.conf** - **nginx** configuration principale
- **/etc/nginx/conf.d/php-fpm.conf** - Configuration FPM pour **nginx**
- **/etc/php-fpm.conf** - Configuration principale du FPM
- **/etc/php-fpm.d/www.conf** - configuration par défaut du pool **www**

Exemple 5.2. Exécution d'un script PHP de type "Hello, World!" En utilisant le serveur nginx

1. Créez un répertoire **hello** pour votre projet dans le répertoire **/usr/share/nginx/html/**:

```
# mkdir hello
```

2. Créez un fichier **hello.php** dans le répertoire **/usr/share/nginx/html/hello/** avec le contenu suivant :

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

```
?>
</body>
</html>
```

3. Démarrez le serveur **nginx**:

```
# systemctl start nginx
```

4. Pour exécuter le fichier **hello.php**, pointez le navigateur sur :

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

Le résultat est l'affichage d'une page web avec le texte "Hello, World !

Ressources supplémentaires

- Mise en place et configuration de NGINX

5.3. EXÉCUTER UN SCRIPT PHP À L'AIDE DE L'INTERFACE DE LIGNE DE COMMANDE

Un script PHP est généralement exécuté à l'aide d'un serveur web, mais il peut également être exécuté à l'aide de l'interface de ligne de commande.

Conditions préalables

- Le langage de script PHP est installé sur votre système.

Procédure

1. Dans un éditeur de texte, créez un ***filename*.php** fichier
Remplacez *filename* par le nom de votre fichier.

2. Exécutez le fichier créé ***filename*.php** à partir de la ligne de commande :

```
# php filename.php
```

Exemple 5.3. Exécution d'un script PHP "Hello, World!" PHP à l'aide de l'interface de ligne de commande

1. Créez un fichier **hello.php** avec le contenu suivant à l'aide d'un éditeur de texte :

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

2. Exécutez le fichier **hello.php** à partir de la ligne de commande :

```
# php hello.php
```

Le résultat est "Hello, World !".

5.4. RESSOURCES SUPPLÉMENTAIRES

- **httpd(8)** - La page de manuel du service **httpd** contenant la liste complète de ses options de ligne de commande.
- **httpd.conf(5)** - La page de manuel pour la configuration de **httpd**, décrivant la structure et l'emplacement des fichiers de configuration de **httpd**.
- **nginx(8)** - La page de manuel du serveur web **nginx** contenant la liste complète des options de la ligne de commande et la liste des signaux.
- **php-fpm(8)** - La page de manuel de PHP FPM décrivant la liste complète des options de la ligne de commande et des fichiers de configuration.