vmware by Broadcom

Tanzu Spring

Tanzu Spring Commercial

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://techdocs.broadcom.com/

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

VMware Tanzu Spring

VMware Tanzu Spring is an enterprise support subscription that includes multiple benefits in addition to the value that the Spring open source projects and ecosystem provide. The following sections cover these high level benefits and point to where you can learn more about each.

Extended support

With Tanzu Spring, you can get premium support for the following open source software:

- Spring
- Apache Tomcat
- Apache HTTP Server
- OpenJDK™

Spring Enterprise Subscription artifact repository

For Spring Boot minor versions that have entered `Enterprise` support and are no longer under `OSS` support, patch releases are made available through a Spring Enterprise Subscription, our VMware Spring artifact repository. The Spring Boot support page shows the current state of minor versions and their support status.

Spring Application Advisor

Spring Application Advisor is a set of tools for continuously and incrementally upgrading Spring application dependencies, source code, and configuration across all your Git repositories. The Spring Application Advisor CLI can be integrated into Continuous Integration pipelines to generate source code updates and merge requests for specific upgrade steps.

Enterprise Spring Boot Extensions

The Enterprise Spring Boot Governance Starter extension generates compliance standard and governance audit information on the `/actuator/governance` endpoint for your application. You may also extend Governance Starter to create your own governance and compliance policy validations inside your applications. Governance Starter is available for inclusion in your application dependencies via the Spring Enterprise Subscription artifact repository.

Spring Cloud Enterprise Components

Customers can use enterprise-ready implementations of Spring Cloud application infrastructure based on popular Spring projects with capabilities such as integrated access control, day-2 operations, and platform-native integrations. These Spring Cloud enterprise components are targeted for VMware Tanzu Platform for Cloud Foundry (formerly called Tanzu Application Service) and Kubernetes environments.

For Tanzu Platform for Cloud Foundry (formerly called Tanzu Application Service)

Tanzu Platform for Cloud Foundry is a modern runtime for microservices built on Cloud Foundry. The following list includes Spring Cloud tiles that can be installed and made available as services to your applications. These Spring Cloud tiles go beyond default OSS project libraries and provide dynamic service binding, automated security patterns, and platform integrations for Tanzu Platform for Cloud Foundry.

- Spring Cloud Services
- Spring Cloud Gateway for VMware Tanzu
- Spring Cloud Data Flow for VMware Tanzu

For Kubernetes

The following Spring Enterprise offerings are available as part of Tanzu Spring and can be deployed on any Kubernetes environment based on their respective prerequisite version support. The use of these Spring Enterprise offerings is enhanced when used as part of Tanzu Platform for Kubernetes that provides secure build and deployment of your Spring applications through a pre-paved path to production.

- Spring Cloud Gateway for Kubernetes
- Spring Cloud Data Flow for Kubernetes
- API portal for VMware Tanzu

VMware Tanzu tc Server

VMware Tanzu tc Server provides tooling to manage the lifecycle of a Java servlet container with enterprise expertise built in, along with a repeatable and scalable configuration approach with templates.

- VMware Tanzu tc Server documentation

VMware Distribution of OpenJDK

VMware provides a binary distribution of OpenJDK that is supported as part of Tanzu Spring.

- VMware Distribution of OpenJDK documentation

Reference Information

- Tanzu Spring overview page
- Tanzu Spring support page on spring.io
- Tanzu Spring Framework overview page

Apache HTTP Server built by VMware

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. The goal of this project is to provide a secure, efficient, and extensible server that provides HTTP services in sync with the current HTTP standards.

Releases

- Apache HTTP Server built by VMware 2.4.63-20250218195700
- Apache HTTP Server built by VMware 2.4.62-20240904201630
- Apache HTTP Server built by VMware 2.4.62-20240828181951
- Apache HTTP Server built by VMware 2.4.62-20240717172113
- Apache HTTP Server built by VMware 2.4.61-20240710201530
- Apache HTTP Server built by VMware 2.4.61-20240703200552

Downloading 2.4.63-20250218195700

This release updates the OpenSSL version to 3.3.2.

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. This topic includes instructions for obtaining an access token, which is required for download.

- httpd-ubuntu-2.4.63-20250218195700.tar.bz2
- httpd-windows-x64-2.4.63-20250218195700.zip
- httpd-rhel-2.4.63-20250218195700.tar.bz2
- httpd-sources-2.4.63-20250218195700.zip
- release-notes-2.4.63-20250218195700.md

Downloading 2.4.62-20240904201630

This release updates the OpenSSL version to 3.3.2.

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. This topic includes instructions for obtaining an access token, which is required for download.

- httpd-ubuntu-2.4.62-20240904201630.tar.bz2
- httpd-windows-x64-2.4.62-20240904201630.zip
- httpd-rhel-2.4.62-20240904201630.tar.bz2
- httpd-sources-2.4.62-20240904201630.zip

- release-notes-2.4.62-20240904201630.md

Downloading 2.4.62-20240828181951

This release updates the APR version to 1.7.5, which addresses CVE-2023-49582

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. This topic includes instructions for obtaining an access token, which is required for download.

- httpd-ubuntu-2.4.62-20240828181951.tar.bz2
- httpd-windows-x64-2.4.62-20240828181951.zip
- httpd-rhel-2.4.62-20240828181951.tar.bz2
- httpd-sources-2.4.62-20240828181951.zip
- release-notes-2.4.62-20240828181951.md

Downloading 2.4.62-20240717172113

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. This topic includes instructions for obtaining an access token, which is required for download.

- httpd-ubuntu-2.4.62-20240717172113.tar.bz2
- httpd-windows-x64-2.4.62-20240717172113.zip
- httpd-rhel-2.4.62-20240717172113.tar.bz2
- httpd-sources-2.4.62-20240717172113.zip
- release-notes-2.4.62-20240717172113.md

Downloading 2.4.61-20240710201530

This release is for rhel7, it is built from identical sources as 2.4.61-20240703145951

- httpd-rhel-2.4.61-20240710201530.tar.bz2 (RHEL 7+)

Downloading 2.4.61-20240703145951

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. This topic includes instructions for obtaining an access token, which is required for download.

- httpd-ubuntu-2.4.61-20240703200552.tar.bz2
- httpd-windows-x64-2.4.61-20240703200552.zip
- httpd-rhel-2.4.61-20240703200552.tar.bz2 (For RHEL 8+, use 2.4.61-20240710201530 for RHEL 7)
- httpd-sources-2.4.61-20240703200552.zip
- release-notes-2.4.61-20240703200552.md

About Apache HTTP Server

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. The goal of this project is to provide a secure,

efficient, and extensible server that provides HTTP services in sync with the current HTTP standards.

RELEASE-NOTES-2-4-61-20240703200552

Updated: July 03, 2024

Build Date: July 03, 2024

What's in the Release Notes

Package Description

This package includes Apache HTTP Server (httpd), along with a number of frequently updated library components (dependencies).

This package is structured to allow parallel installation of multiple releases of Apache HTTP Server and related components. It contains one directory tree, labeled as 2.4.55-230207 which represents the current version of httpd and of all components bundled in the package as of the effective date. In this case, all of the components reflect current releases as of the releases build date.

Unlike many httpd distributions, the end user instance configuration, server content, and logs are not modified in this directory tree. See the section about Instance Creation for details of creating a server instance with these user maintained files.

In order to build httpd from scratch, see additional details at VMware's github oss-httpd-build project. A tarball of the unix sources and zipfile of the windows sources is provided alongside the binary release downloads, for ready reference.

Versions prior to 2.4.53 used the OpenSSL and PCRE-8.x legacy versions. As of httpd 2.4.53, OpenSSL release 3.0 and PCRE2 release 10.x are used instead. If modules were also compiled to consume OpenSSL or PCRE2 themselves, they must be rebuilt.

Downloading

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. An access token is required to download.

- httpd-ubuntu-2.4.61-20240703200552.tar.bz2
- httpd-windows-x64-2.4.61-20240703200552.zip
- httpd-rhel-2.4.61-20240703200552.tar.bz2
- httpd-sources-2.4.61-20240703041113.zip
- release-notes-2.4.61-20240703200552.md

Included Components

The following components are included in this httpd-2.4.55-230207 build; those marked (*) are not compiled on RHEL 7 and Ubuntu 18.04, but the OS Vendors' distribution packages are used instead. Links to the user change notes and vulnerability indexes are illustrated below. Packages updated since the previous release httpd-2.4.54-220722 are identified in boldface. In cases where the project does not maintain a reference to specific CVE's in an easily web accessible format, the https://www.cvedetails.com/vulnerability-list/ database link is provided; this list is not endorsed as complete or comprehensive and is offered for convenience only.

- Apache HTTPS Server 2.4.61
  - http://www.apache.org/dist/httpd/CHANGES_2.4
  - http://httpd.apache.org/security/vulnerabilities_24.html
- Apache APR library 1.7.4
  - http://www.apache.org/dist/apr/CHANGES-APR-1.7
- Apache APR-iconvlibrary 1.2.2
  - http://www.apache.org/dist/apr/CHANGES-APR-ICONV-1.2
- Apache APR-utillibrary 1.6.3
  - http://www.apache.org/dist/apr/CHANGES-APR-UTIL-1.6
- brotli compression library 1.0.9
  - https://github.com/google/brotli/releases
- Curl 8.8.0
  - https://curl.haxx.se/changes.html https://curl.haxx.se/docs/security.html
- expat 2.6.2
  - https://github.com/libexpat/libexpat/blob/master/expat/Changes
- Jansson 2.14
  - https://jansson.readthedocs.io/en/stable/changes.html
- libxml 2.13.1
  - https://www.cvedetails.com/vulnerability-list/vendor_id-1962/product_id-3311/Xmlsoft-Libxml2.html
  - http://www.xmlsoft.org/news.html (out of date)
- Lua language 5.4.7
  - https://www.cvedetails.com/vulnerability-list/vendor_id-13641/product_id-28436/LUA-LUA.html

    - https://www.lua.org/bugs.html
- nghttp2 library 1.62.1
    - https://github.com/nghttp2/nghttp2/releases
- OpenSSL library openssl-3.3.1
    - https://www.openssl.org/news/vulnerabilities.html
    - https://www.openssl.org/news/changelog.html
- PCRE2 library 10.44
    - https://www.cvedetails.com/vulnerability-list/vendor_id-3265/product_id-33513/Pcre-cre2.html
    - https://www.pcre.org/changelog.txt
- Zlib compression librar 1.3.1
    - https://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-1820/GNU-Zlib.html
    - https://zlib.net/ChangeLog.txt

RHEL 7 Users

The RHEL 7 package requires several commonly installed packages to be available. These can be provisioned with the following command:

```
$ yum install expat jansson libuuid libxml2 lua pcre2 zlib
```

Note the addition of the jansson package to this list since the 2.4.29-171109 release, and the change to pcre2 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/ oss-httpd-build README page.

RHEL 8 Users

The RHEL 7 package is compatible with RHEL 8 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

RHEL 9 Users

The RHEL 7 package is compatible with RHEL 9 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib libxcrypt-compat
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages

are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

Ubuntu 20.04 and 22.04 Users

The Ubuntu package requires several commonly installed packages to be available, these may be provisioned with the following command:

```
$ apt-get -y install libexpat1 libjansson4 liblua5.3-0 libpcre2-8-0 libxml2 zlib1g bzip2
```

Note the addition of the libjansson4 package and corrected liblua5.3-0 and libxml2 package names to this list since the 2.4.29-171109 release, and the change to libpcre2-8-0 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/oss-httpd-build README page.

Users can still build this package from source for the Ubuntu 16.04 operating system, see https://github.com/vmware-tanzu/oss-httpd-build#readme, however, that OS is no longer supported.

Microsoft Windows Users

This package is built using Visual C++ 19 and C Runtime version 14, components of Microsoft Visual Studio 2022. Windows Server 2022 and Windows Server 2019 are both suitable for deployment. Windows 10 Desktop and Windows 11 Desktop are suitable for developer evaluation, but are not suitable for server deployment, as Microsoft restricts the Windows Desktop license, limiting aspects of the operating system behavior, including the Windows Sockets API, and tunes the process scheduler to deliver a better desktop experience. Users must obtain and install the “Microsoft Visual C++ Redistributable for Visual Studio 2022”, x64 edition from https://visualstudio.microsoft.com/downloads/ (currently this is listed under Other Tools and Frameworks, and provides support for Visual Studio 2015, 2017 and 2019 as well.) Install the x64 flavor, and observe the prerequisites noted for that package. Installing this package from Microsoft ensures that this runtime is updated by the Windows Update service for security vulnerabilities within the Universal C Runtime itself.

Note that VMware convenience packages prior to httpd 2.4.53 were built with Visual Studio 2017 or 2019. This may cause issues for users who have compiled third-party modules. Users are advised to rebuild any such modules before combining them with these newer packages. This package relies upon Windows PowerShell to execute the httpd control scripts on Windows computers. All supported Windows versions have PowerShell installed by default, but specific installations of Windows may not. To check whether your version of Windows has PowerShell installed, go to **Start** > **All Programs** > **Accessories** and check for Windows PowerShell in the list.

If Windows PowerShell is not already installed, install it, as directed, at https://docs.microsoft.com/en-us/powershell/scripting/setup/setup-reference.

If necessary, enable Windows PowerShell for script processing; script processing may be disabled by default:

1. Start PowerShell from the Start Menu as an Administrator by opening **Start** > **All Programs**

Accessories > **Windows PowerShell**, then right-clicking on Windows PowerShell and selecting **Run as Administrator**. A PowerShell window starts.

1. Check the current PowerShell setting by executing the following command: PS prompt> Get-ExecutionPolicy.
2. If the command returns Restricted, it means that PowerShell is not yet enabled. Enable it to allow local script processing at a minimum by executing the following command: PS prompt> Set-ExecutionPolicy RemoteSigned
3. You can choose a different execution policy for your organization, and enable PowerShell using Group and User policies. Typically, only the Administrator will be using the server control scripts, so the RemoteSigned execution policy should be adequate in most cases.

Windows users must take note that extracting the zip file contents using the File Explorer from a remote drive or volume, or from an untrusted “blocked” file will result in untrusted and non-executable files and scripts. For the windows binary package, copy the .zip file to a local drive before using the File Explorer extraction tool. If the .zip file was downloaded, then using the Windows File Explorer examine the .zip file properties, and under **Security** below the **Attributes** item, check the **Unblock** check box to mark the zip file contents as trusted. If the **Security** item is not present, the file is already unblocked.

In the command-line example given below, unzip is provided by info-zip, while mklink is an intrinsic cmd.exe command which is not provided by PowerShell.

Installation

Create the desired install path, such as /opt/vmware/webserver or C:\VMware\WebServer, and unpack the tar.bz2 or .zip file into that directory. From this root directory, then invoke the fixrootpath script to correct the embedded paths to the current path, and finally create a symlink ‘httpd-2.4’ in parallel to the installed httpd product path, once ready to adopt this installation as the “accepted” httpd-2.4 installation. During an upgrade, restart each server instance individually and verify the correct operation of that instance’s hosts. If there is a problem resulting from an upgrade, simply restore the symlink to the previously installed httpd path, and restart the servers with the old version to avoid unnecessary interruption. When correct operation is verified the older httpd version can be expunged.

Unix users (running as root); Windows users (in a Command window ‘Run as Administrator’);

Instance Creation

This distribution of Apache HTTP Server is parameterized to allow multiple instances to be created and managed independently, without duplicating the binary files. The instance directory is typically named for a primary server hostname and contains the instance-specific directories conf, htdocs, logs and ssl (for certificates and keys).

Unix users (running as root):

```
$ mkdir -p /opt/vmware/webserver
$ cd /opt/vmware/webserver
$ tar -xjvf {path-to}/httpd-2.4.55-230207-{arch}.tar.bz2
$ httpd-2.4.55-230207/bin/fixrootpath.pl httpd-2.4.55-230207
$ ln -s httpd-2.4.55-230207 httpd-2.4
Windows users (in PowerShell 'Run as Administrator');
C:\> mkdir \VMware\WebServer
```

```
C:\> cd \VMware\WebServer
C:\[...]> unzip {path-to}\httpd-2.4.55-230207-windows-x64.zip
C:\[...]> powershell httpd-2.4.55-230207\bin\fixrootpath.ps1 httpd-2.4.55-230207
C:\[...]> mklink /d httpd-2.4 httpd-2.4.55-230207
Unix users (running as root);
$ cd /opt/vmware/webserver
$ httpd-2.4/bin/newserver.pl --server {hostname}
$ cd {hostname}
$ bin/httpdctl install
$ bin/httpdctl start
Modify the files in {hostname}/conf/ to customize the server behavior. Use the httpden
v script
in the bin directory of the instance to have access to the various tools shipped in th
e httpd-2.4
bin directory;
$ . bin/httpdenv.sh
```

Or on Windows:

```
PS C:\VMware\WebServer\example.com> bin\httpdenv.ps1
```

The httpdctl uninstall command will remove the service from automatic startup at boot time. Updating Instances In general, no special action is required when upgrading between httpd-2.4.x releases, directives should be backwards-compatible. Restarting the server with httpdctl should be sufficient. From time to time, httpdctl itself is upgraded, and to update the instance with refreshed control scripts, it is best to uninstall any system service associated with the instance, use the –update feature of newserver.ps, and finally re-install the system service (with potentially a new service name.) Unix Users (running as root):

1. Stop and uninstall the old instance:

```
$ cd /opt/vmware/webserver
$ {instance}/bin/httpdctl stop
$ {instance}/bin/httpdctl uninstall
```

1. Update the script with new features plus any revised service names:

```
$ httpd-2.4/bin/newserver.pl --server={instance} --update
```

1. Install and start the service with the new name:

```
$ {instance}/bin/httpdctl install
$ {instance}/bin/httpdctl start
```

1. Repeat steps 1-3 for each service.

RELEASE-NOTES-2-4-61-20240710201530

Updated: July 10, 2024

Build Date: July 10, 2024

What’s in the Release Notes

- Package Description
- Included Components

- RHEL 7 Users
- RHEL 8 Users
- RHEL 9 Users
- Ubuntu Users
- Microsoft Windows Users
- Installation
- Instance Creation
- Updating Instances

Package Description

This package includes Apache HTTP Server (httpd), along with a number of frequently updated library components (dependencies).

This package is structured to allow parallel installation of multiple releases of Apache HTTP Server and related components. It contains one directory tree, labeled as 2.4.61-20240710201530 which represents the current version of httpd and of all components bundled in the package as of the effective date. In this case, all of the components reflect current releases as of the releases build date.

Unlike many httpd distributions, the end user instance configuration, server content, and logs are not modified in this directory tree. See the section about Instance Creation for details of creating a server instance with these user maintained files.

A tarball of the sources is provided alongside the binary release downloads, for ready reference.

Versions prior to 2.4.53 used the OpenSSL and PCRE-8.x legacy versions. As of httpd 2.4.53, OpenSSL release 3.0 and PCRE2 release 10.x are used instead. If modules were also compiled to consume OpenSSL or PCRE2 themselves, they must be rebuilt.

Downloading

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. An access token is required to download.

- httpd-rhel-2.4.61-20240710201530.tar.bz2

Included Components

The following components are included in this httpd-2.4.55-230207 build; those marked (*) are not compiled on RHEL 7 and Ubuntu 18.04, but the OS Vendors' distribution packages are used instead. Links to the user change notes and vulnerability indexes are illustrated below. Packages updated since the previous release httpd-2.4.54-220722 are identified in boldface. In cases where the project does not maintain a reference to specific CVE's in an easily web accessible format, the https://www.cvedetails.com/vulnerability-list/ database link is provided; this list is not endorsed as complete or comprehensive and is offered for convenience only.

- Apache HTTPS Server 2.4.61
    - http://www.apache.org/dist/httpd/CHANGES_2.4

    - http://httpd.apache.org/security/vulnerabilities_24.html
- Apache APR library 1.7.4
    - http://www.apache.org/dist/apr/CHANGES-APR-1.7
- Apache APR-iconvlibrary 1.2.2
    - http://www.apache.org/dist/apr/CHANGES-APR-ICONV-1.2
- Apache APR-utillibrary 1.6.3
    - http://www.apache.org/dist/apr/CHANGES-APR-UTIL-1.6
- brotli compression library 1.0.9
    - https://github.com/google/brotli/releases
- Curl 8.8.0
    - https://curl.haxx.se/changes.html https://curl.haxx.se/docs/security.html
- expat 2.6.2
    - https://github.com/libexpat/libexpat/blob/master/expat/Changes
- Jansson 2.14
    - https://jansson.readthedocs.io/en/stable/changes.html
- libxml 2.13.1
    - https://www.cvedetails.com/vulnerability-list/vendor_id-1962/product_id-3311/Xmlsoft-Libxml2.html
    - http://www.xmlsoft.org/news.html (out of date)
- Lua language 5.4.7
    - https://www.cvedetails.com/vulnerability-list/vendor_id-13641/product_id-28436/LUA-LUA.html
    - https://www.lua.org/bugs.html
- nghttp2 library 1.62.1
    - https://github.com/nghttp2/nghttp2/releases
- OpenSSL library openssl-3.3.1
    - https://www.openssl.org/news/vulnerabilities.html
    - https://www.openssl.org/news/changelog.html
- PCRE2 library 10.44
    - https://www.cvedetails.com/vulnerability-list/vendor_id-3265/product_id-33513/Pcre-cre2.html
    - https://www.pcre.org/changelog.txt
- Zlib compression librar 1.3.1
    - https://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-1820/GNU-Zlib.html
    - https://zlib.net/ChangeLog.txt

RHEL 7 Users

The RHEL 7 package requires several commonly installed packages to be available. These can be provisioned with the following command:

```
$ yum install expat jansson libuuid libxml2 lua pcre2 zlib
```

Note the addition of the jansson package to this list since the 2.4.29-171109 release, and the change to pcre2 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/ oss-httpd-build README page.

RHEL 8 Users

The RHEL 7 package is compatible with RHEL 8 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

RHEL 9 Users

The RHEL 7 package is compatible with RHEL 9 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib libxcrypt-compat
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

Ubuntu 20.04 and 22.04 Users

The Ubuntu package requires several commonly installed packages to be available, these may be provisioned with the following command:

```
$ apt-get -y install libexpat1 libjansson4 liblua5.3-0 libpcre2-8-0 libxml2 zlib1g bzi
p2
```

Note the addition of the libjansson4 package and corrected liblua5.3-0 and libxml2 package names to this list since the 2.4.29-171109 release, and the change to libpcre2-8-0 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/oss-httpd-build README page.

Users can still build this package from source for the Ubuntu 16.04 operating system, see https://github.com/vmware-tanzu/oss-httpd-build#readme, however, that OS is no longer supported.

Microsoft Windows Users

This package is built using Visual C++ 19 and C Runtime version 14, components of Microsoft Visual Studio 2022. Windows Server 2022 and Windows Server 2019 are both suitable for deployment. Windows 10 Desktop and Windows 11 Desktop are suitable for developer evaluation, but are not suitable for server deployment, as Microsoft restricts the Windows Desktop license, limiting aspects of the operating system behavior, including the Windows Sockets API, and tunes the process scheduler to deliver a better desktop experience. Users must obtain and install the “Microsoft Visual C++ Redistributable for Visual Studio 2022”, x64 edition from https://visualstudio.microsoft.com/downloads/ (currently this is listed under Other Tools and Frameworks, and provides support for Visual Studio 2015, 2017 and 2019 as well.) Install the x64 flavor, and observe the prerequisites noted for that package. Installing this package from Microsoft ensures that this runtime is updated by the Windows Update service for security vulnerabilities within the Universal C Runtime itself.

Note that VMware convenience packages prior to httpd 2.4.53 were built with Visual Studio 2017 or 2019. This may cause issues for users who have compiled third-party modules. Users are advised to rebuild any such modules before combining them with these newer packages. This package relies upon Windows PowerShell to execute the httpd control scripts on Windows computers. All supported Windows versions have PowerShell installed by default, but specific installations of Windows may not. To check whether your version of Windows has PowerShell installed, go to **Start** > **All Programs** > **Accessories** and check for Windows PowerShell in the list.

If Windows PowerShell is not already installed, install it, as directed, at https://docs.microsoft.com/en-us/powershell/scripting/setup/setup-reference.

If necessary, enable Windows PowerShell for script processing; script processing may be disabled by default:

1. Start PowerShell from the Start Menu as an Administrator by opening **Start** > **All Programs**

Accessories > **Windows PowerShell**, then right-clicking on Windows PowerShell and selecting **Run as Administrator**. A PowerShell window starts.

1. Check the current PowerShell setting by executing the following command: PS prompt> Get-ExecutionPolicy.
2. If the command returns Restricted, it means that PowerShell is not yet enabled. Enable it to allow local script processing at a minimum by executing the following command: PS prompt> Set-ExecutionPolicy RemoteSigned
3. You can choose a different execution policy for your organization, and enable PowerShell using Group and User policies. Typically, only the Administrator will be using the server control scripts, so the RemoteSigned execution policy should be adequate in most cases.

Windows users must take note that extracting the zip file contents using the File Explorer from a remote drive or volume, or from an untrusted “blocked” file will result in untrusted and non-executable files and scripts. For the windows binary package, copy the .zip file to a local drive before using the File Explorer extraction tool. If the .zip file was downloaded, then using the Windows File Explorer examine the .zip file properties, and under **Security** below the **Attributes** item, check the **Unblock** check box to mark the zip file contents as trusted. If the **Security** item is not present, the file is already unblocked.

In the command-line example given below, unzip is provided by info-zip, while mklink is an intrinsic cmd.exe command which is not provided by PowerShell.

Installation

Create the desired install path, such as /opt/vmware/webserver or C:\VMware\WebServer, and unpack the tar.bz2 or .zip file into that directory. From this root directory, then invoke the fixrootpath script to correct the embedded paths to the current path, and finally create a symlink ‘httpd-2.4’ in parallel to the installed httpd product path, once ready to adopt this installation as the “accepted” httpd-2.4 installation. During an upgrade, restart each server instance individually and verify the correct operation of that instance’s hosts. If there is a problem resulting from an upgrade, simply restore the symlink to the previously installed httpd path, and restart the servers with the old version to avoid unnecessary interruption. When correct operation is verified the older httpd version can be expunged.

Unix users (running as root); Windows users (in a Command window ‘Run as Administrator’);

Instance Creation

This distribution of Apache HTTP Server is parameterized to allow multiple instances to be created and managed independently, without duplicating the binary files. The instance directory is typically named for a primary server hostname and contains the instance-specific directories conf, htdocs, logs and ssl (for certificates and keys).

Unix users (running as root):

```
$ mkdir -p /opt/vmware/webserver
$ cd /opt/vmware/webserver
$ tar -xjvf {path-to}/httpd-2.4.55-230207-{arch}.tar.bz2
$ httpd-2.4.55-230207/bin/fixrootpath.pl httpd-2.4.55-230207
$ ln -s httpd-2.4.55-230207 httpd-2.4
Windows users (in PowerShell 'Run as Administrator');
C:\> mkdir \VMware\WebServer
C:\> cd \VMware\WebServer
C:\[...]> unzip {path-to}\httpd-2.4.55-230207-windows-x64.zip
C:\[...]> powershell httpd-2.4.55-230207\bin\fixrootpath.ps1 httpd-2.4.55-230207
C:\[...]> mklink /d httpd-2.4 httpd-2.4.55-230207
Unix users (running as root);
$ cd /opt/vmware/webserver
$ httpd-2.4/bin/newserver.pl --server {hostname}
$ cd {hostname}
$ bin/httpdctl install
$ bin/httpdctl start
Modify the files in {hostname}/conf/ to customize the server behavior. Use the httpden
v script
in the bin directory of the instance to have access to the various tools shipped in th
e httpd-2.4
bin directory;
$ . bin/httpdenv.sh
```

Or on Windows:

```
PS C:\VMware\WebServer\example.com> bin\httpdenv.ps1
```

The httpdctl uninstall command will remove the service from automatic startup at boot time. Updating Instances In general, no special action is required when upgrading between httpd-2.4.x releases, directives should be backwards-compatible. Restarting the server with httpdctl should be sufficient. From time to time, httpdctl itself is upgraded, and to update the instance with refreshed control scripts, it is best to uninstall

any system service associated with the instance, use the –update feature of newserver.ps, and finally re-install the system service (with potentially a new service name.) Unix Users (running as root):

1. Stop and uninstall the old instance:

```
$ cd /opt/vmware/webserver
$ {instance}/bin/httpdctl stop
$ {instance}/bin/httpdctl uninstall
```

1. Update the script with new features plus any revised service names:

```
$ httpd-2.4/bin/newserver.pl --server={instance} --update
```

1. Install and start the service with the new name:

```
$ {instance}/bin/httpdctl install
$ {instance}/bin/httpdctl start
```

1. Repeat steps 1-3 for each service.

RELEASE-NOTES-2-4-62-20240717172113

Updated: July 03, 2024

Build Date: July 03, 2024

What’s in the Release Notes

- Package Description
- Included Components
- RHEL 7 Users
- RHEL 8 Users
- RHEL 9 Users
- Ubuntu Users
- Microsoft Windows Users
- Installation
- Instance Creation
- Updating Instances

Package Description

This package includes Apache HTTP Server (httpd), along with a number of frequently updated library components (dependencies).

This package is structured to allow parallel installation of multiple releases of Apache HTTP Server and related components. It contains one directory tree, labeled as 2.4.62-20240717172113 which represents the current version of httpd and of all components bundled in the package as of the effective date. In this case, all of the components reflect current releases as of the releases build date.

Unlike many httpd distributions, the end user instance configuration, server content, and logs are not modified in this directory tree. See the section about Instance Creation for details of creating a server instance with these user maintained files.

A tarball of the sources is provided alongside the binary release downloads, for ready reference.

Versions prior to 2.4.53 used the OpenSSL and PCRE-8.x legacy versions. As of httpd 2.4.53, OpenSSL release 3.0 and PCRE2 release 10.x are used instead. If modules were also compiled to consume OpenSSL or PCRE2 themselves, they must be rebuilt.

Downloading

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. An access token is required to download.

- httpd-ubuntu-2.4.62-20240717172113.tar.bz2
- httpd-windows-x64-2.4.62-20240717172113.zip
- httpd-rhel-2.4.62-20240717172113.tar.bz2
- httpd-sources-2.4.62-20240717172113.zip
- release-notes-2.4.62-20240717172113.md

Included Components

The following components are included in this httpd-2.4.55-230207 build; those marked (*) are not compiled on RHEL 7 and Ubuntu 18.04, but the OS Vendors' distribution packages are used instead. Links to the user change notes and vulnerability indexes are illustrated below. Packages updated since the previous release httpd-2.4.54-220722 are identified in boldface. In cases where the project does not maintain a reference to specific CVE's in an easily web accessible format, the https://www.cvedetails.com/vulnerability-list/ database link is provided; this list is not endorsed as complete or comprehensive and is offered for convenience only.

- Apache HTTPS Server 2.4.62
    - http://www.apache.org/dist/httpd/CHANGES_2.4
    - http://httpd.apache.org/security/vulnerabilities_24.html
- Apache APR library 1.7.4
    - http://www.apache.org/dist/apr/CHANGES-APR-1.7
- Apache APR-iconvlibrary 1.2.2
    - http://www.apache.org/dist/apr/CHANGES-APR-ICONV-1.2
- Apache APR-utillibrary 1.6.3
    - http://www.apache.org/dist/apr/CHANGES-APR-UTIL-1.6
- brotli compression library 1.0.9
    - https://github.com/google/brotli/releases
- Curl 8.8.0
    - https://curl.haxx.se/changes.html https://curl.haxx.se/docs/security.html

- expat 2.6.2
    - https://github.com/libexpat/libexpat/blob/master/expat/Changes
- Jansson 2.14
    - https://jansson.readthedocs.io/en/stable/changes.html
- libxml 2.13.2
    - https://www.cvedetails.com/vulnerability-list/vendor_id-1962/product_id-3311/Xmlsoft-Libxml2.html
    - http://www.xmlsoft.org/news.html (out of date)
- Lua language 5.4.7
    - https://www.cvedetails.com/vulnerability-list/vendor_id-13641/product_id-28436/LUA-LUA.html
    - https://www.lua.org/bugs.html
- nghttp2 library 1.62.1
    - https://github.com/nghttp2/nghttp2/releases
- OpenSSL library openssl-3.3.1
    - https://www.openssl.org/news/vulnerabilities.html
    - https://www.openssl.org/news/changelog.html
- PCRE2 library 10.44
    - https://www.cvedetails.com/vulnerability-list/vendor_id-3265/product_id-33513/Pcre-cre2.html
    - https://www.pcre.org/changelog.txt
- Zlib compression librar 1.3.1
    - https://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-1820/GNU-Zlib.html
    - https://zlib.net/ChangeLog.txt

RHEL 7 Users

The RHEL 7 package requires several commonly installed packages to be available. These can be provisioned with the following command:

```
$ yum install expat jansson libuuid libxml2 lua pcre2 zlib
```

Note the addition of the jansson package to this list since the 2.4.29-171109 release, and the change to pcre2 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/ oss-httpd-build README page.

RHEL 8 Users

The RHEL 7 package is compatible with RHEL 8 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

RHEL 9 Users

The RHEL 7 package is compatible with RHEL 9 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib libxcrypt-compat
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

Ubuntu 20.04 and 22.04 Users

The Ubuntu package requires several commonly installed packages to be available, these may be provisioned with the following command:

```
$ apt-get -y install libexpat1 libjansson4 liblua5.3-0 libpcre2-8-0 libxml2 zlib1g bzi
p2
```

Note the addition of the libjansson4 package and corrected liblua5.3-0 and libxml2 package names to this list since the 2.4.29-171109 release, and the change to libpcre2-8-0 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/oss-httpd-build README page.

Users can still build this package from source for the Ubuntu 16.04 operating system, see https://github.com/vmware-tanzu/oss-httpd-build#readme, however, that OS is no longer supported.

Microsoft Windows Users

This package is built using Visual C++ 19 and C Runtime version 14, components of Microsoft Visual Studio 2022. Windows Server 2022 and Windows Server 2019 are both suitable for deployment. Windows 10 Desktop and Windows 11 Desktop are suitable for developer evaluation, but are not suitable for server deployment, as Microsoft restricts the Windows Desktop license, limiting aspects of the operating system behavior, including the Windows Sockets API, and tunes the process scheduler to deliver a better desktop experience. Users must obtain and install the “Microsoft Visual C++ Redistributable for Visual Studio 2022”, x64 edition from https://visualstudio.microsoft.com/downloads/ (currently this is listed under Other Tools and Frameworks, and provides support for Visual Studio 2015, 2017 and 2019 as well.) Install the x64 flavor, and observe the prerequisites noted for that package. Installing this package from Microsoft ensures that this runtime is updated by the Windows Update service for security vulnerabilities within the Universal C Runtime itself.

Note that VMware convenience packages prior to httpd 2.4.53 were built with Visual Studio 2017 or 2019. This may cause issues for users who have compiled third-party modules. Users are advised to rebuild any

such modules before combining them with these newer packages. This package relies upon Windows PowerShell to execute the httpd control scripts on Windows computers. All supported Windows versions have PowerShell installed by default, but specific installations of Windows may not. To check whether your version of Windows has PowerShell installed, go to **Start** > **All Programs** > **Accessories** and check for Windows PowerShell in the list.

If Windows PowerShell is not already installed, install it as directed at; https://docs.microsoft.com/en-us/powershell/scripting/setup/setup-reference.

If necessary, enable Windows PowerShell for script processing; script processing may be disabled by default:

1. Start PowerShell from the Start Menu as an Administrator by opening **Start** > **All Programs**

Accessories > **Windows PowerShell**, then right-clicking on Windows PowerShell and selecting **Run as Administrator**. A PowerShell window starts.

1. Check the current PowerShell setting by executing the following command: PS prompt> Get-ExecutionPolicy.
2. If the command returns Restricted, it means that PowerShell is not yet enabled. Enable it to allow local script processing at a minimum by executing the following command: PS prompt> Set-ExecutionPolicy RemoteSigned
3. You can choose a different execution policy for your organization, and enable PowerShell using Group and User policies. Typically, only the Administrator will be using the server control scripts, so the RemoteSigned execution policy should be adequate in most cases.

Windows users must take note that extracting the zip file contents using the File Explorer from a remote drive or volume, or from an untrusted “blocked” file will result in untrusted and non-executable files and scripts. For the windows binary package, copy the .zip file to a local drive before using the File Explorer extraction tool. If the .zip file was downloaded, then using the Windows File Explorer examine the .zip file properties, and under **Security** below the **Attributes** item, check the **Unblock** check box to mark the zip file contents as trusted. If the **Security** item is not present, the file is already unblocked.

In the command-line example given below, unzip is provided by info-zip, while mklink is an intrinsic cmd.exe command which is not provided by PowerShell.

Installation

Create the desired install path, such as /opt/vmware/webserver or C:\VMware\WebServer, and unpack the tar.bz2 or .zip file into that directory. From this root directory, then invoke the fixrootpath script to correct the embedded paths to the current path, and finally create a symlink ‘httpd-2.4’ in parallel to the installed httpd product path, once ready to adopt this installation as the “accepted” httpd-2.4 installation. During an upgrade, restart each server instance individually and verify the correct operation of that instance’s hosts. If there is a problem resulting from an upgrade, simply restore the symlink to the previously installed httpd path, and restart the servers with the old version to avoid unnecessary interruption. When correct operation is verified the older httpd version can be expunged.

Unix users (running as root); Windows users (in a Command window ‘Run as Administrator’);

Instance Creation

This distribution of Apache HTTP Server is parameterized to allow multiple instances to be created and managed independently, without duplicating the binary files. The instance directory is typically named for a primary server hostname and contains the instance-specific directories conf, htdocs, logs and ssl (for certificates and keys).

Unix users (running as root):

```
$ mkdir -p /opt/vmware/webserver
$ cd /opt/vmware/webserver
$ tar -xjvf {path-to}/httpd-2.4.55-230207-{arch}.tar.bz2
$ httpd-2.4.55-230207/bin/fixrootpath.pl httpd-2.4.55-230207
$ ln -s httpd-2.4.55-230207 httpd-2.4
Windows users (in PowerShell 'Run as Administrator');
C:\> mkdir \VMware\WebServer
C:\> cd \VMware\WebServer
C:\[...]> unzip {path-to}\httpd-2.4.55-230207-windows-x64.zip
C:\[...]> powershell httpd-2.4.55-230207\bin\fixrootpath.ps1 httpd-2.4.55-230207
C:\[...]> mklink /d httpd-2.4 httpd-2.4.55-230207
Unix users (running as root);
$ cd /opt/vmware/webserver
$ httpd-2.4/bin/newserver.pl --server {hostname}
$ cd {hostname}
$ bin/httpdctl install
$ bin/httpdctl start
Modify the files in {hostname}/conf/ to customize the server behavior. Use the httpden
v script
in the bin directory of the instance to have access to the various tools shipped in th
e httpd-2.4
bin directory;
$ . bin/httpdenv.sh
```

Or on Windows:

```
PS C:\VMware\WebServer\example.com> bin\httpdenv.ps1
```

The httpdctl uninstall command will remove the service from automatic startup at boot time. Updating Instances In general, no special action is required when upgrading between httpd-2.4.x releases, directives should be backwards-compatible. Restarting the server with httpdctl should be sufficient. From time to time, httpdctl itself is upgraded, and to update the instance with refreshed control scripts, it is best to uninstall any system service associated with the instance, use the –update feature of newserver.ps, and finally re-install the system service (with potentially a new service name.) Unix Users (running as root):

1. Stop and uninstall the old instance:

```
$ cd /opt/vmware/webserver
$ {instance}/bin/httpdctl stop
$ {instance}/bin/httpdctl uninstall
```

1. Update the script with new features plus any revised service names:

```
$ httpd-2.4/bin/newserver.pl --server={instance} --update
```

1. Install and start the service with the new name:

```
$ {instance}/bin/httpdctl install
$ {instance}/bin/httpdctl start
```

1. Repeat steps 1-3 for each service.

RELEASE-NOTES-2-4-62-20240828181951

Updated: July 03, 2024

Build Date: July 03, 2024

What's in the Release Notes

- Package Description
- Included Components
- RHEL 7 Users
- RHEL 8 Users
- RHEL 9 Users
- Ubuntu Users
- Microsoft Windows Users
- Installation
- Instance Creation
- Updating Instances

Package Description

This package includes Apache HTTP Server (httpd), along with a number of frequently updated library components (dependencies).

This package is structured to allow parallel installation of multiple releases of Apache HTTP Server and related components. It contains one directory tree, labeled as 2.4.62-20240828181951 which represents the current version of httpd and of all components bundled in the package as of the effective date. In this case, all of the components reflect current releases as of the releases build date.

Unlike many httpd distributions, the end user instance configuration, server content, and logs are not modified in this directory tree. See the section about Instance Creation for details of creating a server instance with these user maintained files.

A tarball of the sources is provided alongside the binary release downloads, for ready reference.

Versions prior to 2.4.53 used the OpenSSL and PCRE-8.x legacy versions. As of httpd 2.4.53, OpenSSL release 3.0 and PCRE2 release 10.x are used instead. If modules were also compiled to consume OpenSSL or PCRE2 themselves, they must be rebuilt.

Downloading

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. An access token is required to download.

- httpd-ubuntu-2.4.62-20240828181951.tar.bz2
- httpd-windows-x64-2.4.62-20240828181951.zip
- httpd-rhel-2.4.62-20240828181951.tar.bz2
- httpd-sources-2.4.62-20240828181951.zip
- release-notes-2.4.62-20240828181951.md

Included Components

The following components are included in this httpd-2.4.55-230207 build; those marked (*) are not compiled on RHEL 7 and Ubuntu 18.04, but the OS Vendors' distribution packages are used instead. Links to the user change notes and vulnerability indexes are illustrated below. Packages updated since the previous release httpd-2.4.54-220722 are identified in boldface. In cases where the project does not maintain a reference to specific CVE's in an easily web accessible format, the https://www.cvedetails.com/vulnerability-list/ database link is provided; this list is not endorsed as complete or comprehensive and is offered for convenience only.

- Apache HTTPS Server 2.4.62
  - http://www.apache.org/dist/httpd/CHANGES_2.4
  - http://httpd.apache.org/security/vulnerabilities_24.html
- Apache APR library 1.7.5
  - http://www.apache.org/dist/apr/CHANGES-APR-1.7
- Apache APR-iconvlibrary 1.2.2
  - http://www.apache.org/dist/apr/CHANGES-APR-ICONV-1.2
- Apache APR-utillibrary 1.6.3
  - http://www.apache.org/dist/apr/CHANGES-APR-UTIL-1.6
- brotli compression library 1.0.9
  - https://github.com/google/brotli/releases
- Curl 8.8.0
  - https://curl.haxx.se/changes.html https://curl.haxx.se/docs/security.html
- expat 2.6.2
  - https://github.com/libexpat/libexpat/blob/master/expat/Changes
- Jansson 2.14
  - https://jansson.readthedocs.io/en/stable/changes.html
- libxml 2.11.9
  - https://www.cvedetails.com/vulnerability-list/vendor_id-1962/product_id-3311/Xmlsoft-Libxml2.html
  - http://www.xmlsoft.org/news.html (out of date)
- Lua language 5.4.7
  - https://www.cvedetails.com/vulnerability-list/vendor_id-13641/product_id-28436/LUA-LUA.html

    - https://www.lua.org/bugs.html
- nghttp2 library 1.63.0
    - https://github.com/nghttp2/nghttp2/releases
- OpenSSL library openssl-3.3.1
    - https://www.openssl.org/news/vulnerabilities.html
    - https://www.openssl.org/news/changelog.html
- PCRE2 library 10.44
    - https://www.cvedetails.com/vulnerability-list/vendor_id-3265/product_id-33513/Pcre-cre2.html
    - https://www.pcre.org/changelog.txt
- Zlib compression librar 1.3.1
    - https://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-1820/GNU-Zlib.html
    - https://zlib.net/ChangeLog.txt

RHEL 7 Users

The RHEL 7 package requires several commonly installed packages to be available. These can be provisioned with the following command:

```
$ yum install expat jansson libuuid libxml2 lua pcre2 zlib
```

Note the addition of the jansson package to this list since the 2.4.29-171109 release, and the change to pcre2 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/ oss-httpd-build README page.

RHEL 8 Users

The RHEL 7 package is compatible with RHEL 8 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

RHEL 9 Users

The RHEL 7 package is compatible with RHEL 9 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib libxcrypt-compat
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages

are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

Ubuntu 20.04 and 22.04 Users

The Ubuntu package requires several commonly installed packages to be available, these may be provisioned with the following command:

```
$ apt-get -y install libexpat1 libjansson4 liblua5.3-0 libpcre2-8-0 libxml2 zlib1g bzip2
```

Note the addition of the libjansson4 package and corrected liblua5.3-0 and libxml2 package names to this list since the 2.4.29-171109 release, and the change to libpcre2-8-0 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/oss-httpd-build README page.

Users can still build this package from source for the Ubuntu 16.04 operating system, see https://github.com/vmware-tanzu/oss-httpd-build#readme, however, that OS is no longer supported.

Microsoft Windows Users

This package is built using Visual C++ 19 and C Runtime version 14, components of Microsoft Visual Studio 2022. Windows Server 2022 and Windows Server 2019 are both suitable for deployment. Windows 10 Desktop and Windows 11 Desktop are suitable for developer evaluation, but are not suitable for server deployment, as Microsoft restricts the Windows Desktop license, limiting aspects of the operating system behavior, including the Windows Sockets API, and tunes the process scheduler to deliver a better desktop experience. Users must obtain and install the “Microsoft Visual C++ Redistributable for Visual Studio 2022”, x64 edition from https://visualstudio.microsoft.com/downloads/ (currently this is listed under Other Tools and Frameworks, and provides support for Visual Studio 2015, 2017 and 2019 as well.) Install the x64 flavor, and observe the prerequisites noted for that package. Installing this package from Microsoft ensures that this runtime is updated by the Windows Update service for security vulnerabilities within the Universal C Runtime itself.

Note that VMware convenience packages prior to httpd 2.4.53 were built with Visual Studio 2017 or 2019. This may cause issues for users who have compiled third-party modules. Users are advised to rebuild any such modules before combining them with these newer packages. This package relies upon Windows PowerShell to execute the httpd control scripts on Windows computers. All supported Windows versions have PowerShell installed by default, but specific installations of Windows may not. To check whether your version of Windows has PowerShell installed, go to **Start** > **All Programs** > **Accessories** and check for Windows PowerShell in the list.

If Windows PowerShell is not already installed, install it as directed at; https://docs.microsoft.com/en-us/powershell/scripting/setup/setup-reference.

If necessary, enable Windows PowerShell for script processing; script processing may be disabled by default:

1. Start PowerShell from the Start Menu as an Administrator by opening **Start** > **All Programs**

Accessories > **Windows PowerShell**, then right-clicking on Windows PowerShell and selecting **Run as Administrator**. A PowerShell window starts.

1. Check the current PowerShell setting by executing the following command: PS prompt> Get-ExecutionPolicy.
2. If the command returns Restricted, it means that PowerShell is not yet enabled. Enable it to allow local script processing at a minimum by executing the following command: PS prompt> Set-ExecutionPolicy RemoteSigned
3. You can choose a different execution policy for your organization, and enable PowerShell using Group and User policies. Typically, only the Administrator will be using the server control scripts, so the RemoteSigned execution policy should be adequate in most cases.

Windows users must take note that extracting the zip file contents using the File Explorer from a remote drive or volume, or from an untrusted “blocked” file will result in untrusted and non-executable files and scripts. For the windows binary package, copy the .zip file to a local drive before using the File Explorer extraction tool. If the .zip file was downloaded, then using the Windows File Explorer examine the .zip file properties, and under **Security** below the **Attributes** item, check the **Unblock** check box to mark the zip file contents as trusted. If the **Security** item is not present, the file is already unblocked.

In the command-line example given below, unzip is provided by info-zip, while mklink is an intrinsic cmd.exe command which is not provided by PowerShell.

Installation

Create the desired install path, such as /opt/vmware/webserver or C:\VMware\WebServer, and unpack the tar.bz2 or .zip file into that directory. From this root directory, then invoke the fixrootpath script to correct the embedded paths to the current path, and finally create a symlink ‘httpd-2.4’ in parallel to the installed httpd product path, once ready to adopt this installation as the “accepted” httpd-2.4 installation. During an upgrade, restart each server instance individually and verify the correct operation of that instance’s hosts. If there is a problem resulting from an upgrade, simply restore the symlink to the previously installed httpd path, and restart the servers with the old version to avoid unnecessary interruption. When correct operation is verified the older httpd version can be expunged.

Unix users (running as root); Windows users (in a Command window ‘Run as Administrator’);

Instance Creation

This distribution of Apache HTTP Server is parameterized to allow multiple instances to be created and managed independently, without duplicating the binary files. The instance directory is typically named for a primary server hostname and contains the instance-specific directories conf, htdocs, logs and ssl (for certificates and keys).

Unix users (running as root):

```
$ mkdir -p /opt/vmware/webserver
$ cd /opt/vmware/webserver
$ tar -xjvf {path-to}/httpd-2.4.55-230207-{arch}.tar.bz2
$ httpd-2.4.55-230207/bin/fixrootpath.pl httpd-2.4.55-230207
$ ln -s httpd-2.4.55-230207 httpd-2.4
Windows users (in PowerShell 'Run as Administrator');
C:\> mkdir \VMware\WebServer
```

```
C:\> cd \VMware\WebServer
C:\[...]> unzip {path-to}\httpd-2.4.55-230207-windows-x64.zip
C:\[...]> powershell httpd-2.4.55-230207\bin\fixrootpath.ps1 httpd-2.4.55-230207
C:\[...]> mklink /d httpd-2.4 httpd-2.4.55-230207
Unix users (running as root);
$ cd /opt/vmware/webserver
$ httpd-2.4/bin/newserver.pl --server {hostname}
$ cd {hostname}
$ bin/httpdctl install
$ bin/httpdctl start
Modify the files in {hostname}/conf/ to customize the server behavior. Use the httpden
v script
in the bin directory of the instance to have access to the various tools shipped in th
e httpd-2.4
bin directory;
$ . bin/httpdenv.sh
```

Or on Windows:

```
PS C:\VMware\WebServer\example.com> bin\httpdenv.ps1
```

The httpdctl uninstall command will remove the service from automatic startup at boot time. Updating Instances In general, no special action is required when upgrading between httpd-2.4.x releases, directives should be backwards-compatible. Restarting the server with httpdctl should be sufficient. From time to time, httpdctl itself is upgraded, and to update the instance with refreshed control scripts, it is best to uninstall any system service associated with the instance, use the –update feature of newserver.ps, and finally re-install the system service (with potentially a new service name.) Unix Users (running as root):

1. Stop and uninstall the old instance:

```
$ cd /opt/vmware/webserver
$ {instance}/bin/httpdctl stop
$ {instance}/bin/httpdctl uninstall
```

1. Update the script with new features plus any revised service names:

```
$ httpd-2.4/bin/newserver.pl --server={instance} --update
```

1. Install and start the service with the new name:

```
$ {instance}/bin/httpdctl install
$ {instance}/bin/httpdctl start
```

1. Repeat steps 1-3 for each service.

RELEASE-NOTES-2-4-62-20240904201630

Updated: November 06, 2024

Build Date: September 04, 2024

What’ in the Release Notes

- Package Description
- Included Components

- RHEL 7 Users
- RHEL 8 Users
- RHEL 9 Users
- Ubuntu Users
- Microsoft Windows Users
- Installation
- Instance Creation
- Updating Instances

Package Description

This package includes Apache HTTP Server (httpd), along with a number of frequently updated library components (dependencies).

This package is structured to allow parallel installation of multiple releases of Apache HTTP Server and related components. It contains one directory tree, labeled as 2.4.62-20240904201630 which represents the current version of httpd and of all components bundled in the package as of the effective date. In this case, all of the components reflect current releases as of the releases build date.

Unlike many httpd distributions, the end user instance configuration, server content, and logs are not modified in this directory tree. See the section about Instance Creation for details of creating a server instance with these user maintained files.

A tarball of the sources is provided alongside the binary release downloads, for ready reference.

Versions prior to 2.4.53 used the OpenSSL and PCRE-8.x legacy versions. As of httpd 2.4.53, OpenSSL release 3.0 and PCRE2 release 10.x are used instead. If modules were also compiled to consume OpenSSL or PCRE2 themselves, they must be rebuilt.

Downloading

Apache HTTP built by VMware is distributed as part of Spring Enterprise Subscription. An access token is required to download.

- httpd-ubuntu-2.4.62-20240904201630.tar.bz2
- httpd-windows-x64-2.4.62-20240904201630.zip
- httpd-rhel-2.4.62-20240904201630.tar.bz2
- httpd-sources-ubuntu-2.4.62-20240904201630.zip
- release-notes-2.4.62-20240904201630.md

Included Components

The following components are included in this httpd-2.4.55-230207 build; those marked (*) are not compiled on RHEL 7 and Ubuntu 18.04, but the OS Vendors' distribution packages are used instead. Links to the user change notes and vulnerability indexes are illustrated below. Packages updated since the previous release httpd-2.4.54-220722 are identified in boldface. In cases where the project does not maintain a

reference to specific CVE's in an easily web accessible format, the https://www.cvedetails.com/vulnerability-list/ database link is provided; this list is not endorsed as complete or comprehensive and is offered for convenience only.

- Apache HTTPS Server 2.4.62
  - http://www.apache.org/dist/httpd/CHANGES_2.4
  - http://httpd.apache.org/security/vulnerabilities_24.html
- Apache APR library 1.7.5
  - http://www.apache.org/dist/apr/CHANGES-APR-1.7
- Apache APR-iconvlibrary 1.2.2
  - http://www.apache.org/dist/apr/CHANGES-APR-ICONV-1.2
- Apache APR-utillibrary 1.6.3
  - http://www.apache.org/dist/apr/CHANGES-APR-UTIL-1.6
- brotli compression library 1.0.9
  - https://github.com/google/brotli/releases
- Curl 8.8.0
  - https://curl.haxx.se/changes.html https://curl.haxx.se/docs/security.html
- expat 2.6.3
  - https://github.com/libexpat/libexpat/blob/master/expat/Changes
- Jansson 2.14
  - https://jansson.readthedocs.io/en/stable/changes.html
- libxml 2.11.9
  - https://www.cvedetails.com/vulnerability-list/vendor_id-1962/product_id-3311/Xmlsoft-Libxml2.html
  - http://www.xmlsoft.org/news.html (out of date)
- Lua language 5.4.7
  - https://www.cvedetails.com/vulnerability-list/vendor_id-13641/product_id-28436/LUA-LUA.html
  - https://www.lua.org/bugs.html
- nghttp2 library 1.63.0
  - https://github.com/nghttp2/nghttp2/releases
- OpenSSL library openssl-3.3.2
  - https://www.openssl.org/news/vulnerabilities.html
  - https://www.openssl.org/news/changelog.html
- PCRE2 library 10.44
  - https://www.cvedetails.com/vulnerability-list/vendor_id-3265/product_id-33513/Pcre-cre2.html
  - https://www.pcre.org/changelog.txt
- Zlib compression librar 1.3.1

- https://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-1820/GNU-Zlib.html
- https://zlib.net/ChangeLog.txt

RHEL 7 Users

The RHEL 7 package requires several commonly installed packages to be available. These can be provisioned with the following command:

```
$ yum install expat jansson libuuid libxml2 lua pcre2 zlib
```

Note the addition of the jansson package to this list since the 2.4.29-171109 release, and the change to pcre2 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/ oss-httpd-build README page.

RHEL 8 Users

The RHEL 7 package is compatible with RHEL 8 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

RHEL 9 Users

The RHEL 7 package is compatible with RHEL 9 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib libxcrypt-compat
```

Note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. To use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

Ubuntu 20.04 and 22.04 Users

The Ubuntu package requires several commonly installed packages to be available, these may be provisioned with the following command:

```
$ apt-get -y install libexpat1 libjansson4 liblua5.3-0 libpcre2-8-0 libxml2 zlib1g bzi
p2
```

Note the addition of the libjansson4 package and corrected liblua5.3-0 and libxml2 package names to this list since the 2.4.29-171109 release, and the change to libpcre2-8-0 since the 2.4.51-211007 release. To use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/oss-httpd-build README page.

Users can still build this package from source for the Ubuntu 16.04 operating system, see https://github.com/vmware-tanzu/oss-httpd-build#readme, however, that OS is no longer supported.

Microsoft Windows Users

This package is built using Visual C++ 19 and C Runtime version 14, components of Microsoft Visual Studio 2022. Windows Server 2022 and Windows Server 2019 are both suitable for deployment. Windows 10 Desktop and Windows 11 Desktop are suitable for developer evaluation, but are not suitable for server deployment, as Microsoft restricts the Windows Desktop license, limiting aspects of the operating system behavior, including the Windows Sockets API, and tunes the process scheduler to deliver a better desktop experience. Users must obtain and install the “Microsoft Visual C++ Redistributable for Visual Studio 2022”, x64 edition from https://visualstudio.microsoft.com/downloads/ (currently this is listed under Other Tools and Frameworks, and provides support for Visual Studio 2015, 2017 and 2019 as well.) Install the x64 flavor, and observe the prerequisites noted for that package. Installing this package from Microsoft ensures that this runtime is updated by the Windows Update service for security vulnerabilities within the Universal C Runtime itself.

Note that VMware convenience packages prior to httpd 2.4.53 were built with Visual Studio 2017 or 2019. This may cause issues for users who have compiled third-party modules. Users are advised to rebuild any such modules before combining them with these newer packages. This package relies upon Windows PowerShell to execute the httpd control scripts on Windows computers. All supported Windows versions have PowerShell installed by default, but specific installations of Windows may not. To check whether your version of Windows has PowerShell installed, go to **Start** > **All Programs** > **Accessories** and check for Windows PowerShell in the list.

If Windows PowerShell is not already installed, install it as directed at; https://docs.microsoft.com/en-us/powershell/scripting/setup/setup-reference.

If necessary, enable Windows PowerShell for script processing; script processing may be disabled by default:

1. Start PowerShell from the Start Menu as an Administrator by opening **Start** > **All Programs**

Accessories > **Windows PowerShell**, then right-clicking on Windows PowerShell and selecting **Run as Administrator**. A PowerShell window starts.

1. Check the current PowerShell setting by executing the following command: PS prompt> Get-ExecutionPolicy.
2. If the command returns Restricted, it means that PowerShell is not yet enabled. Enable it to allow local script processing at a minimum by executing the following command: PS prompt> Set-ExecutionPolicy RemoteSigned
3. You can choose a different execution policy for your organization, and enable PowerShell using Group and User policies. Typically, only the Administrator will be using the server control scripts, so the RemoteSigned execution policy should be adequate in most cases.

Windows users must take note that extracting the zip file contents using the File Explorer from a remote drive or volume, or from an untrusted “blocked” file will result in untrusted and non-executable files and scripts. For the windows binary package, copy the .zip file to a local drive before using the File Explorer

extraction tool. If the .zip file was downloaded, then using the Windows File Explorer examine the .zip file properties, and under **Security** below the **Attributes** item, check the **Unblock** check box to mark the zip file contents as trusted. If the **Security** item is not present, the file is already unblocked.

In the command-line example given below, unzip is provided by info-zip, while mklink is an intrinsic cmd.exe command which is not provided by PowerShell.

Installation

Create the desired install path, such as /opt/vmware/webserver or C:\VMware\WebServer, and unpack the tar.bz2 or .zip file into that directory. From this root directory, then invoke the fixrootpath script to correct the embedded paths to the current path, and finally create a symlink ‘httpd-2.4’ in parallel to the installed httpd product path, once ready to adopt this installation as the “accepted” httpd-2.4 installation.

During an upgrade, restart each server instance individually and verify the correct operation of that instance’s hosts. If there is a problem resulting from an upgrade, simply restore the symlink to the previously installed httpd path, and restart the servers with the old version to avoid unnecessary interruption. When correct operation is verified the older httpd version can be expunged.

Unix users (running as root); Windows users (in a Command window ‘Run as Administrator’);

Instance Creation

This distribution of Apache HTTP Server is parameterized to allow multiple instances to be created and managed independently, without duplicating the binary files. The instance directory is typically named for a primary server hostname and contains the instance-specific directories conf, htdocs, logs and ssl (for certificates and keys).

Unix users (running as root):

```
$ mkdir -p /opt/vmware/webserver
$ cd /opt/vmware/webserver
$ tar -xjvf {path-to}/httpd-2.4.55-230207-{arch}.tar.bz2
$ httpd-2.4.55-230207/bin/fixrootpath.pl httpd-2.4.55-230207
$ ln -s httpd-2.4.55-230207 httpd-2.4
Windows users (in PowerShell 'Run as Administrator');
C:\> mkdir \VMware\WebServer
C:\> cd \VMware\WebServer
C:\[...]> unzip {path-to}\httpd-2.4.55-230207-windows-x64.zip
C:\[...]> powershell httpd-2.4.55-230207\bin\fixrootpath.ps1 httpd-2.4.55-230207
C:\[...]> mklink /d httpd-2.4 httpd-2.4.55-230207
Unix users (running as root);
$ cd /opt/vmware/webserver
$ httpd-2.4/bin/newserver.pl --server {hostname}
$ cd {hostname}
$ bin/httpdctl install
$ bin/httpdctl start
Modify the files in {hostname}/conf/ to customize the server behavior. Use the httpden
v script
in the bin directory of the instance to have access to the various tools shipped in th
e httpd-2.4
bin directory;
$ . bin/httpdenv.sh
```

Or on Windows:

```
PS C:\VMware\WebServer\example.com> bin\httpdenv.ps1
```

The httpdctl uninstall command will remove the service from automatic startup at boot time. Updating Instances In general, no special action is required when upgrading between httpd-2.4.x releases, directives should be backwards-compatible. Restarting the server with httpdctl should be sufficient. From time to time, httpdctl itself is upgraded, and to update the instance with refreshed control scripts, it is best to uninstall any system service associated with the instance, use the –update feature of newserver.ps, and finally re-install the system service (with potentially a new service name.) Unix Users (running as root):

1. Stop and uninstall the old instance:

```
$ cd /opt/vmware/webserver
$ {instance}/bin/httpdctl stop
$ {instance}/bin/httpdctl uninstall
```

1. Update the script with new features plus any revised service names:

```
$ httpd-2.4/bin/newserver.pl --server={instance} --update
```

1. Install and start the service with the new name:

```
$ {instance}/bin/httpdctl install
$ {instance}/bin/httpdctl start
```

1. Repeat steps 1-3 for each service.

RELEASE-NOTES 2.4.63-20250218195700

Apache HTTP Built by VMware 2.4.63-20250218195700 Release Notes

What’s in the Release Notes

- Package Description
- Included Components
- RHEL 7 Users
- RHEL 8 Users
- RHEL 9 Users
- Ubuntu Users
- Microsoft Windows Users
- Installation
- Instance Creation
- Updating Instances

Package Description

This package includes Apache HTTP Server (httpd), along with a number of frequently updated library components (dependencies).

This package is structured to allow parallel installation of multiple releases of Apache HTTP Server and related components. It contains one directory tree, labeled as 2.4.63-20250218195700 which represents the current version of httpd and of all components bundled in the package as of the effective date, in this case, all of the components reflect current releases as of the releases build date

Unlike many httpd distributions, the end user instance configuration, server content and logs are not modified in this directory tree. See the section on Instance Creation for details of creating a server instance with these user maintained files.

A tarball of the sources is provided alongside the binary release downloads, for ready reference.

Versions prior to 2.4.53 used the OpenSSL and PCRE-8.x legacy versions. As of httpd 2.4.53, OpenSSL release 3.0 and PCRE2 release 10.x are used instead. If modules were also compiled to consume OpenSSL or PCRE2 themselves, they must be rebuilt.

Downloading

Apache HTTP built by VMware is distributed as part of [Spring Enterprise Subscription]) [https://docs.vmware.com/en/Tanzu-Spring-Runtime/Commercial/Tanzu-Spring-Runtime/spring-enterprise-subscription.html). An access token is required to download.

- httpd-ubuntu-2.4.63-20250218195700.tar.bz2
- httpd-windows-x64-2.4.63-20250218195700.zip
- httpd-rhel-2.4.63-20250218195700.tar.bz2
- httpd-sources-2.4.63-20250218195700.zip
- release-notes-2.4.63-20250218195700.md

Included Components

The following components are included in this httpd-2.4.55-230207 build; those marked (*) are not compiled on RHEL 7 and Ubuntu 18.04, but the OS Vendors' distribution packages are used instead. Links to the user change notes and vulnerability indexes are illustrated below. Packages updated since the previous release httpd-2.4.54-220722 are identified in boldface. In cases where the project does not maintain a reference to specific CVE's in an easily web accessible format the https://www.cvedetails.com/vulnerability-list/ database link is provided; this list is not endorsed as complete or comprehensive and is offered for convenience only.

- Apache HTTPS Server 2.4.63
    - http://www.apache.org/dist/httpd/CHANGES_2.4
    - http://httpd.apache.org/security/vulnerabilities_24.html
- Apache APR library 1.7.5
    - http://www.apache.org/dist/apr/CHANGES-APR-1.7
- Apache APR-iconvlibrary 1.2.2
    - http://www.apache.org/dist/apr/CHANGES-APR-ICONV-1.2

- Apache APR-utillibrary 1.6.3
    - http://www.apache.org/dist/apr/CHANGES-APR-UTIL-1.6
- brotli compression library 1.0.9
    - https://github.com/google/brotli/releases
- Curl 8.11.1
    - https://curl.haxx.se/changes.html https://curl.haxx.se/docs/security.html
- expat 2.6.4
    - https://github.com/libexpat/libexpat/blob/master/expat/Changes
- Jansson 2.14
    - https://jansson.readthedocs.io/en/stable/changes.html
- libxml 2.13.6
    - https://www.cvedetails.com/vulnerability-list/vendor_id-1962/product_id-3311/Xmlsoft-Libxml2.html
    - http://www.xmlsoft.org/news.html (out of date)
- Lua language 5.4.7
    - https://www.cvedetails.com/vulnerability-list/vendor_id-13641/product_id-28436/LUA-LUA.html
    - https://www.lua.org/bugs.html
- nghttp2 library 1.64.0
    - https://github.com/nghttp2/nghttp2/releases
- OpenSSL library openssl-3.4.1
    - https://www.openssl.org/news/vulnerabilities.html
    - https://www.openssl.org/news/changelog.html
- PCRE2 library 10.45
    - https://www.cvedetails.com/vulnerability-list/vendor_id-3265/product_id-33513/Pcre-cre2.html
    - https://www.pcre.org/changelog.txt
- Zlib compression librar 1.3.1
    - https://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-1820/GNU-Zlib.html
    - https://zlib.net/ChangeLog.txt

RHEL 7 Users

The RHEL 7 package requires several commonly installed packages to be available, these may be provisioned with the following command;

```
$ yum install expat jansson libuuid libxml2 lua pcre2 zlib
```

Please note the addition of the jansson package to this list since the 2.4.29-171109 release, and the change to pcre2 since the 2.4.51-211007 release. In order to use the provided apxs utility, additional packages are

required as indicated at the https://github.com/vmware-tanzu/ oss-httpd-build README page.

RHEL 8 Users

The RHEL 7 package is compatible with RHEL 8 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib
```

Please note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. In order to use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

RHEL 9 Users

The RHEL 7 package is compatible with RHEL 9 and Fedora 30+, and requires some less commonly installed packages to be available. These may all be provisioned with the following command;

```
$ dnf install expat jansson libuuid libxcrypt libxml2 pcre2 zlib libxcrypt-compat
```

Please note the change to pcre2 since the 2.4.51-211007 release. On some later flavors of linux, libxcrypt may go by the package designation libxcrypt-compat instead. In order to use the provided apxs utility, additional packages are required as indicated at the https:// github.com/vmware-tanzu/oss-httpd-build README page.

Ubuntu 20.04 and 22.04 Users

The Ubuntu package requires several commonly installed packages to be available, these may be provisioned with the following command;

```
$ apt-get -y install libexpat1 libjansson4 liblua5.3-0 libpcre2-8-0 libxml2 zlib1g bzi
p2
```

Please note the addition of the libjansson4 package and corrected liblua5.3-0 and libxml2 package names to this list since the 2.4.29-171109 release, and the change to libpcre2-8-0 since the 2.4.51-211007 release. In order to use the provided apxs utility, additional packages are required as indicated at the https://github.com/vmware-tanzu/oss-httpd-build README page.

NOTICE: Users may still build this package from source for the Ubuntu 16.04 operating system, see https://github.com/vmware-tanzu/oss-httpd-build#readme however that OS is no longer supported.

Microsoft Windows Users

This package is built using Visual C++ 19 and C Runtime version 14, components of Microsoft Visual Studio 2022. Windows Server 2022 and Windows Server 2019 are both suitable for deployment. Windows 10 Desktop and Windows 11 Desktop are suitable for developer evaluation but are not suitable for server deployment, as Microsoft restricts the Windows Desktop license, limiting aspects of the operating system behavior including the Windows Sockets API, and tunes the process scheduler to deliver a better desktop

experience. Users must obtain and install the “Microsoft Visual C++ Redistributable for Visual Studio 2022”, x64 edition; from https://visualstudio.microsoft.com/downloads/ (currently this is listed under Other Tools and Frameworks, and provides support for Visual Studio 2015, 2017 and 2019 as well.) Install the x64 flavor, and observe the prerequisites noted for that package. Installing this package from Microsoft ensures that this runtime is updated by the Windows Update service for security vulnerabilities within the Universal C Runtime itself.

Note that VMware convenience packages prior to httpd 2.4.53 were built with Visual Studio 2017 or 2019. This may cause issues for users who have compiled third-party modules. Users are advised to rebuild any such modules before combining them with these newer packages. This package relies upon Windows PowerShell to execute the httpd control scripts on Windows computers. All supported Windows versions have PowerShell installed by default, but specific installations of Windows may not. To check whether your version of Windows has PowerShell installed, go to Start > All Programs > Accessories and check for Windows PowerShell in the list.

If Windows PowerShell is not already installed, install it as directed at; https://docs.microsoft.com/en-us/powershell/scripting/setup/setup-reference

If necessary, enable Windows PowerShell for script processing; script processing may be disabled by default; 1. Start PowerShell from the Start Menu as an Administrator by opening Start > All Programs

Accessories > Windows PowerShell, then right-clicking on Windows PowerShell and selecting Run as Administrator. A PowerShell window starts.

1. Check the current PowerShell setting by executing the following command: PS prompt> Get-ExecutionPolicy.
2. If the command returns Restricted, it means that PowerShell is not yet enabled. Enable it to allow local script processing at a minimum by executing the following command: PS prompt> Set-ExecutionPolicy RemoteSigned
3. You can choose a different execution policy for your organization if you want, as well as enable PowerShell using Group and User policies. Typically, only the Administrator will be using the server control scripts, so the RemoteSigned execution policy should be adequate in most cases.

Windows users must take note that extracting the zip file contents using the File Explorer from a remote drive or volume, or from an untrusted “blocked” file will result in untrusted and non- executable files and scripts. For the windows binary package, copy the .zip file to a local drive before using the File Explorer extraction tool. If the .zip file was downloaded, then using the Windows File Explorer examine the .zip file properties, and under ‘Security’ below the ‘Attributes’ item, check the “Unblock” checkbox to mark the zip file contents as trusted. If the ‘Security’ item is not present, the file is already unblocked.

In the command-line example given below, unzip is provided by info-zip, while mklink is an intrinsic cmd.exe command which is not provided by PowerShell.

Installation

Create the desired install path, such as /opt/vmware/webserver or C:\VMware\WebServer, and unpack the tar.bz2 or .zip file into that directory. From this root directory, then invoke the fixrootpath script to correct the embedded paths to the current path, and finally create a symlink ‘httpd-2.4’ in parallel to the installed httpd product path, once ready to adopt this installation as the “accepted” httpd-2.4 installation. During an upgrade, restart each server instance individually and verify the correct operation of that instance’s hosts. If

there is a problem resulting from an upgrade, simply restore the symlink to the previously installed httpd path, and restart the servers with the old version to avoid unnecessary interruption. When correct operation is verified the older httpd version can be expunged.

Unix users (running as root); Windows users (in a Command window ‘Run as Administrator’);

Instance Creation

This distribution of Apache HTTP Server is parameterized to allow multiple instances to be created and managed independently, without duplicating the binary files. The instance directory is typically named for a primary server hostname and contains the instance-specific directories conf, htdocs, logs and ssl (for certificates and keys). Unix users (running as root);

```
$ mkdir -p /opt/vmware/webserver
$ cd /opt/vmware/webserver
$ tar -xjvf {path-to}/httpd-2.4.55-230207-{arch}.tar.bz2
$ httpd-2.4.55-230207/bin/fixrootpath.pl httpd-2.4.55-230207
$ ln -s httpd-2.4.55-230207 httpd-2.4
```

Windows users (in PowerShell ‘Run as Administrator’);

```
C:\> mkdir \VMware\WebServer
C:\> cd \VMware\WebServer
C:\[...]> unzip {path-to}\httpd-2.4.55-230207-windows-x64.zip
C:\[...]> powershell httpd-2.4.55-230207\bin\fixrootpath.ps1 httpd-2.4.55-230207
C:\[...]> mklink /d httpd-2.4 httpd-2.4.55-230207
```

Unix users (running as root);

```
$ cd /opt/vmware/webserver
$ httpd-2.4/bin/newserver.pl --server {hostname}
$ cd {hostname}
$ bin/httpdctl install
$ bin/httpdctl start
```

Modify the files in {hostname}/conf/ to customize the server behavior. Use the httpdenv script in the bin directory of the instance to have access to the various tools shipped in the httpd-2.4 bin directory;

```
$ . bin/httpdenv.sh
```

Or on Windows;

```
PS C:\VMware\WebServer\example.com> bin\httpdenv.ps1
```

The httpdctl uninstall command will remove the service from automatic startup at boot time. Updating Instances

In general, no special action is required when upgrading between httpd-2.4.x releases, directives should be backwards-compatible. Restarting the server with httpdctl should be sufficient.

From time to time, httpdctl itself is upgraded, and to update the instance with refreshed control scripts, it is best to uninstall any system service associated with the instance, use the –update feature of newserver.ps, and finally re-install the system service (with potentially a new service name.)

Unix Users (running as root);

1. Stop and uninstall the old instance:

```
$ cd /opt/vmware/webserver
$ {instance}/bin/httpdctl stop
$ {instance}/bin/httpdctl uninstall
```

1. Update the script with new features plus any revised service names:

```
$ httpd-2.4/bin/newserver.pl --server={instance} --update
```

1. Install and start the service with the new name:

```
$ {instance}/bin/httpdctl start
```

1. Repeat steps 1-3 for each service

Spring Application Advisor

Spring Application Advisor is a set of tools for continuously and incrementally upgrading Spring application dependencies, source code, and configuration across all your Git repositories. The Spring Application Advisor CLI can be integrated into Continuous Integration pipelines to generate source code updates and merge requests for specific upgrade steps.

- Troubleshooting Spring Application Advisor
- Spring Application Advisor CLI Reference

Release Notes

These are the Release Notes for Spring Application Advisor.

1.1.2

Release Date: January 14, 2025

- Adds support for creating upgrade plans for Spring Cloud Commons, Spring Cloud Config, and Spring Cloud Netflix 4.2.x.
- Adds support for creating upgrade plans for Spring Cloud Consul, Spring Cloud Kubernetes, Spring Cloud Gateway, Spring Cloud Circuit Breaker, and Spring Cloud Bus.
- Fixes connectivity issue when generating upgrade mappings using the experimental `advisor mapping` command.
- Enables publishing the build configuration with the Tanzu Platform UI (SaaS) using an OAuth application.
- Adds support for defining and downloading upgrade mappings from a Maven repository.
- Adds support for creating upgrade plans for Spring Data Geode.
- Fixes an issue with generating the build configuration when there is a CycloneDXBom task defined.

1.1.1

Release Date: December 20, 2024

- Adds support for creating upgrade plans for Spring Boot 3.4.x.
- Integrates Spring commercial recipes for Spring Framework 6.2.x.
- Integrates Spring commercial recipes for Spring Data Commons 3.0.x.
- Integrates Spring commercial recipes for Spring Data JPA 3.0.x.
- Integrates Spring commercial recipes for Spring Data Redis 3.0.x.
- Adds support for loading custom upgrade mappings from Git.
- Adds the experimental `advisor mapping build` command, which is not exposed in the CLI.
- Adds support for loading custom upgrade mappings from HTTP.
- Adds hot-reloading of custom upgrade mappings.
- Adds integration with the SaaS version of the Tanzu Platform UI.
- Adds support for creating upgrade plans for Spring Data Redis and Jedis.
- Adds support for creating upgrade plans for Spring LDAP.
- Adds support for creating upgrade plans for Spring Data Commons.

- Adds support for creating upgrade plans for Spring Webflow.
- Adds support for creating upgrade plans for Spring Hateoas.
- Adds support for creating upgrade plans for Reactor Pool, Hibernate, RxJava, R2DBC projects, Servicetalk, Redisson, AWS SDK, Tiles Autotag and Tiles Request.
- Adds explanation of the blocking dependencies to continue with the upgrade.
- Adds explanation of the possible target versions of a blocking project/dependency.
- Fixes issues with resolving upgrade plans related with transitive dependencies.
- Updates the environment variables to connect to Tanzu Platform UI.

1.1.0

- Fixes issue with calculating Maven application modules when the build configuration is created.
- Fixes issue with applying upgrade plans in Maven multi-modules
- Fixes issue with resolving the upgrade plan when semantic versioning needs to be applied.
- Fixes issue with resolving upgrade plans when custom upgrade plans are integrated.
- Adds support for creating upgrade plans for Spring Data MongoDB.
- Adds support for creating upgrade plans for Reactor, Reactor Netty and Reactor Netty Incubator.
- Adds support for creating upgrade plans for Spring Cloud Services Starters, Spring Cloud Open Service Broker, and Spring Cloud App Broker.
- Adds support for creating upgrade plans for SpringFox
- Upgrades the commercial recipes to integrate a complete Spring Boot 3.0.x recipe.
- Adds Maven and Gradle debug messages when log files are generated.
- Adds integration with the Tanzu Platform UI Self Managed via Tanzu Spring Server 1.0.x when `advisor build-config publish` is executed.

1.0.4

- Fixes issue with calculating the upgrade plan when Spring projects are consumed from different dependencies
- Integrates Spring commercial recipes for complete upgrades to Spring Framework 6.1.x
- Integrates Spring commercial recipes for complete upgrades to SpringDoc 2.x
- Fixes issue with regenerating changes that were introduced in previous upgrades of Spring Security and Spring Boot 3.x

1.0.3

- Adds a `--force` option in the upgrade-plan get/apply commands to preview the upgrade when there are external dependencies that require Spring with no upgrade plans configured
- Adds support for creating upgrade plans for Spring Cloud Netflix

- Allows configuration of upgrade plans for custom projects
- Fixes issue with creating pull requests in GitHub.com
- Fixes issue with resolving the submodules of a Maven repository
- Adds support for creating upgrade plans for Pre-Liquibase, which uses Spring

1.0.2

- Disables the remaining automatic additions of Governance Starter Spring Boot extension
- Provides support for creating pull requests in GitHub Enterprise
- Prevents running the Java 17 upgrade in Spring Boot 3.x projects
- Adds support for creating upgrade plans for Spring Cloud Dataflow, Spring Data Commons, and Spring Cloud App Broker
- Adds support for creating upgrade plans for Spring Cloud Vault, Spring Cloud Task and Spring Vault
- Adds support for creating upgrade plans for SpringDoc
- Adds support for creating upgrade plans for Spring LDAP
- Adds support for creating upgrade plans for Spring Data JPA 2.1.x
- Fixes issue that occurred upgrading using Spring Data JPA 2.2.x
- Upgrades managed dependencies for Spring Boot 3.x upgrades
- Removes the list of executed recipes in the CLI output when there is an execution error during the upgrade

1.0.1

- Adds support for creating upgrade plans for Spring AI.
- Adds support for creating upgrade plans for Resilience4j.
- Adds support for creating upgrade plans for Cloud Foundry Java Client.
- Adds support for creating upgrade plans for Spring Cloud Sleuth, Spring Cloud Alibaba, Spring Cloud Stream.
- Adds support for creating upgrade plans for Wavefront.
- Adds tracing for the upgrade plan resolution in the Application Advisor CLI and server.
- Fixes issue that occurred in the resolution of the upgrade plan when there are dependency cycles.
- Disables the automatic addition of Governance Starter Spring Boot extension.

1.0.0

- Integration with Spring Commercial recipes 1.0.0
- Adds support for creating upgrade plans for Spring Retry.

- Adds support for creating upgrade plans for Spring Cloud, Spring Cloud Commons, and Spring Cloud Connectors.

0.0.9

- Adds support for creating upgrade plans for Cloud Foundry dependencies.
- Adds support for upgrading Java versions just right before they are unsupported by Spring projects.
- Fixes issue that occurred when upgrading Spring Boot applications using Governance Starter Enterprise extension.
- Adds support for creating upgrade plans for Spring Cloud Azure dependencies.
- Includes several enhancements in the CLI messages.

0.0.8

- Report of upgrade blockers.
- Report of Maven and Gradle modules in the build-config file.

0.0.7

- Integration with Spring commercial recipes for:
    - Spring Boot 3.1.x and 3.0.x
    - Spring Framework 6.0.x
    - Spring Security 5.8.x and 6.0.x

This means that you might need to configure/adapt your Maven settings. Refer to Running commercial recipes using OpenRewrite tools for more details.

0.0.6

- Updates for independent upgrade steps for Java and Gradle

What is Spring Application Advisor?

Spring Application Advisor is a VMware Tanzu Spring capability for continuously and incrementally upgrading Spring dependencies in all your Git repositories.

Spring Application Advisor creates an upgrade pull request every time it detects that there is a new version available for your Spring dependencies. These pull requests include changes in your build configuration and Java files. With the new pull request, the corresponding developer team can easily review the code changes and validate their correctness using their CI/CD engine.

How is Spring Application Advisor Different From Other Solutions?

Spring Boot Migrator

Spring Application Advisor is designed to replace Spring Boot Migrator. Spring Application Advisor supports only the use case for upgrading your Spring applications, but provides higher upgrade coverage than Spring Boot Migrator through use of Spring Commercial OpenRewrite recipes.

OpenRewrite

Spring Application Advisor runs on OpenRewrite, but there are some key differences.

1. Developers do not need to understand, search, and compose the recipes they need to upgrade their Spring applications. Spring Application Advisor selects the recipes based on the project setup without exposing any OpenRewrite contract.
2. Includes increased coverage for Spring project upgrade paths with Tanzu Spring-provided recipes.
3. Prevents invalid upgrades when Spring is consumed as a transitive dependency if there are no associated OpenRewrite recipes. You will know when new recipes need to be created before upgrading. For instance, if your organization has a custom Spring starter located in a different repository than your application, Spring Application Advisor will not upgrade applications using that starter if there are no recipes configured for it.

How Spring Application Advisor Works

Spring Application Advisor is a package that is composed of:

- the native CLI
- the Server (requires Java 17)

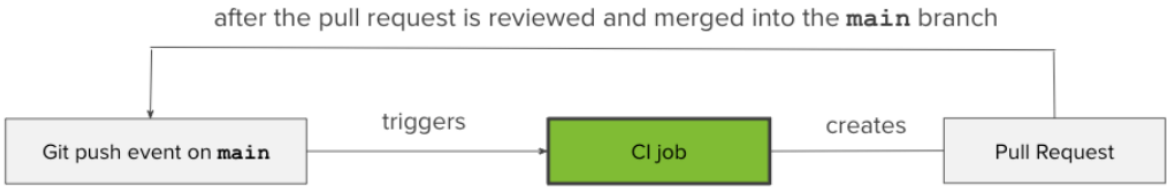
Both components are required.

The native CLI

This is a required component, and is currently available on Linux. This component is responsible for:

- Generating the dependency tree and the build tool versions of a Git repository.
- Running the refactors that apply the corresponding dependency version changes and Java API upgrades, if needed, using the OpenRewrite recipes reported by the server.
- Creating pull requests with the refactors. The CLI needs a Git access token with write access to the repository.

It is assumed that the CLI is integrated into the CI/CD environment so that the Git repositories are continuously analyzed and upgraded to the next version, if necessary. The CI/CD environment is already configured to have access to internal Maven repositories, and to be able to resolve all the dependencies and compile the sources.



The Server

Requirements: Java 17 or higher

This is a required component. The Server is responsible for computing the upgrade plan, which is the list of Spring dependencies or tools that must be upgraded together (using OpenRewrite recipes) to the next release.

The Server also stores the dependency trees and build tool versions that have been inferred from each build. By default, this information is stored in memory, but it can be stored in a SQL database.

Spring Application Advisor How-to Guides

You can use the following “How-to” topics to help you understand how to use Spring Application Advisor:

- Upgrade Spring Boot from 2.7 to 3.4
- Upgrade an Spring application that uses a custom Spring Boot Starter

Upgrade Spring Boot from 2.7 to 3.4

This is a classic exercise that upgrades a Spring Boot 2.7.x application with Java 8 to the latest version of Spring Boot.

For this example, we are using a detached commit of an existing OSS repository called Spring Petclinic, a basic application that uses Spring Boot.

The main branch of this repository is already up to date with the latest Spring Boot version. For the sake of this example, we use a detached branch when the application was using Spring Boot 2.7.

```
git clone https://github.com/spring-projects/spring-petclinic
cd spring-petclinic
git branch advisor-demo 9ecdc1111e3da388a750ace41a125287d9620534
git checkout -f advisor-demo
```

The requirements for upgrading this repository are:

- The Spring Application Advisor server component is ready to accept connections. See Install App Advisor.
- The CLI is available in your `$PATH`. See Run App Advisor.
- Minimum requirement: Java SDK 17 or higher is available. Recommended: Java SDK 8, 11, and 17 are available.
- You have a tool to manage multiple different Java versions. In this guide, we use sdkman, but you can use any tool available.

The first step is to generate the build configuration of your application, which means to generate all the information required to build it: the dependency tree (SBOM), the Java version, the build tool version, and the application modules. Run the following command:

```
advisor build-config get
```

The result of the command is:

```
Resolving the build configuration of spring-petclinic.

  [ 1 / 3 ] Resolving dependencies with mvnw [00m 04s] ok
  [ 2 / 3 ] Resolving JDK version [00m 02s] ok
  [ 3 / 3 ] Resolving build tool [00m 01s] ok

  The build-configuration has been generated in target/.advisor/build-config.json
```

The build configuration is a file required to resolve the upgrade plan of an application.

To resolve the upgrade plan, you must use the server. For convenience, VMware recommends that you define the URL location of the server in the following environment variable.

```
export ADVISOR_SERVER=http://YOUR_ADVISOR_SERVER_LOCATION
```

Now you can run the following command:

```
advisor upgrade-plan get
```

This command prints the Spring Petclinic upgrade plan as follows:

```
  Fetching and processing upgrade plan details [00m 01s] ok
- Step 1:
    * Upgrade java from 8 to 11
- Step 2:
    * Upgrade java from 11 to 17
- Step 3:
    * Upgrade spring-data-jpa from 2.7.x to 3.0.x
    * Upgrade hibernate-orm from 5.6.x to 6.1.x
    * Upgrade spring-framework from 5.3.x to 6.0.x
    * Upgrade spring-boot from 2.7.x to 3.0.x
    * Upgrade spring-data-commons from 2.7.x to 3.0.x
    * Upgrade micrometer from 1.9.x to 1.10.x
- Step 4:
    * Upgrade spring-data-jpa from 3.0.x to 3.1.x
    * Upgrade hibernate-orm from 6.1.x to 6.2.x
    * Upgrade spring-boot from 3.0.x to 3.1.x
    * Upgrade spring-data-commons from 3.0.x to 3.1.x
    * Upgrade micrometer from 1.10.x to 1.11.x
- Step 5:
    * Upgrade spring-data-jpa from 3.1.x to 3.2.x
    * Upgrade hibernate-orm from 6.2.x to 6.4.x
    * Upgrade spring-framework from 6.0.x to 6.1.x
    * Upgrade spring-boot from 3.1.x to 3.2.x
    * Upgrade spring-data-commons from 3.1.x to 3.2.x
    * Upgrade micrometer from 1.11.x to 1.12.x
- Step 6:
    * Upgrade spring-data-jpa from 3.2.x to 3.3.x
    * Upgrade hibernate-orm from 6.4.x to 6.5.x
    * Upgrade spring-boot from 3.2.x to 3.3.x
    * Upgrade spring-data-commons from 3.2.x to 3.3.x
    * Upgrade micrometer from 1.12.x to 1.13.x
- Step 7:
     * Upgrade spring-data-jpa from 3.3.x to 3.4.x
     * Upgrade hibernate-orm from 6.5.x to 6.6.x
     * Upgrade spring-framework from 6.1.x to 6.2.x
     * Upgrade spring-boot from 3.3.x to 3.4.x
```

```
     * Upgrade spring-data-commons from 3.3.x to 3.4.x
     * Upgrade micrometer from 1.13.x to 1.14.x
```

Next, apply the upgrade plan. Run the following command to apply the first step, which is `Upgrade java from 8 to 11`. Before running the command, ensure that you have already configured the Spring Enterprise Maven repository. See Guide for Artifact Repository Developers on your developer workstation, because this step requires it.

```
advisor upgrade-plan apply
```

and you should see the following output:

```
  [ 1 / 2 ] Fetching and processing upgrade plan details [00m 01s] ok

Projects to upgrade:
        * java from 8 to 11

  [ 2 / 2 ] Upgrading sources... [00m 25s] ok

  Successfully applied upgrade.
```

This produces a very small change. However, that is the value of the tool; it lets developers upgrade as much as they can without imposing an upgrade to the latest version of Spring. To review the changes, run the following command:

```
git diff
```

The output should look similar to the following:

```
diff --git a/pom.xml b/pom.xml
index d29355c..29b736e 100644
--- a/pom.xml
+++ b/pom.xml
@@ -15,7 +15,7 @@
   <properties>

     <!-- Generic properties -->
-    <java.version>1.8</java.version>
+    <java.version>11</java.version>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

@@ -23,7 +23,7 @@
     <webjars-bootstrap.version>5.1.3</webjars-bootstrap.version>
     <webjars-font-awesome.version>4.7.0</webjars-font-awesome.version>

-    <jacoco.version>0.8.7</jacoco.version>
+    <jacoco.version>0.8.12</jacoco.version>
     <nohttp-checkstyle.version>0.0.10</nohttp-checkstyle.version>
     <spring-format.version>0.0.31</spring-format.version>
```

As you can see, this upgrade is also upgrading Jacoco. This is because Jacoco traditionally advertises full backwards compatibility for older Java versions.

You can check that your application is still working using Java 11. Run the following commands:

```
sdk install java 11.0.25-tem
sdk use java 11.0.25-tem
./mvnw test
```

The end of the output should be:

```
[INFO]
[INFO] Results:
[INFO]
[WARNING] Tests run: 41, Failures: 0, Errors: 0, Skipped: 1
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  17.765 s
[INFO] Finished at: 2024-11-06T11:11:17+01:00
[INFO] ------------------------------------------------------------------------
```

To continue applying the steps, consider committing the changes produced at each step. Run the following command to include the changes in the `advisor-demo` branch.

```
git add .
git commit -m "Upgrade java from 8 to 11"
```

To proceed with the upgrade to Java 17, repeat the same steps because the Java version has changed, and therefore the build configuration must be regenerated:

```
advisor build-config get && advisor upgrade-plan apply
```

Now, after you run `git diff`, you see this change:

```
diff --git a/pom.xml b/pom.xml
index 29b736e..2a448a2 100644
--- a/pom.xml
+++ b/pom.xml
@@ -15,7 +15,7 @@
   <properties>

     <!-- Generic properties -->
-    <java.version>11</java.version>
+    <java.version>17</java.version>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

@@ -126,7 +126,7 @@
       <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-checkstyle-plugin</artifactId>
-        <version>3.1.2</version>
+        <version>3.6.0</version>
         <dependencies>
           <dependency>
             <groupId>com.puppycrawl.tools</groupId>
```

The upgrade to Java 17 affects some build plug-ins, so `maven-checkstyle-plugin` upgrade is enforced because old versions might have issues with the introduction of text blocks in Java 17.

Now, to evaluate the changes, continue with the following commands:

```
sdk install java 17.0.13-tem
sdk use java 17.0.13-tem
./mvnw test
```

You get the same result as in the previous upgrade, so you can commit the changes again.

```
git add .
git commit -m "Upgrade java from 11 to 17"
```

Now, you are ready to start upgrading to Spring Boot 3.0.x, which is an important upgrade because it replaces the `javax` packages with Jakarta.

```
advisor build-config get && advisor upgrade-plan apply
```

In this case, after you run `git status`, you see the following changes:

```
On branch advisor-demo
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   pom.xml
        modified:   src/main/java/org/springframework/samples/petclinic/model/BaseEnti
ty.java
        modified:   src/main/java/org/springframework/samples/petclinic/model/NamedEnt
ity.java
        modified:   src/main/java/org/springframework/samples/petclinic/model/Person.j
ava
        modified:   src/main/java/org/springframework/samples/petclinic/owner/Owner.ja
va
        modified:   src/main/java/org/springframework/samples/petclinic/owner/OwnerCon
troller.java
        modified:   src/main/java/org/springframework/samples/petclinic/owner/Pet.java
        modified:   src/main/java/org/springframework/samples/petclinic/owner/PetContr
oller.java
        modified:   src/main/java/org/springframework/samples/petclinic/owner/PetType.
java
        modified:   src/main/java/org/springframework/samples/petclinic/owner/Visit.ja
va
        modified:   src/main/java/org/springframework/samples/petclinic/owner/VisitCon
troller.java
        modified:   src/main/java/org/springframework/samples/petclinic/system/CrashCo
ntroller.java
        modified:   src/main/java/org/springframework/samples/petclinic/vet/Specialty.
java
        modified:   src/main/java/org/springframework/samples/petclinic/vet/Vet.java
        modified:   src/main/java/org/springframework/samples/petclinic/vet/VetControl
ler.java
        modified:   src/main/java/org/springframework/samples/petclinic/vet/Vets.java
        modified:   src/main/resources/application.properties
        modified:   src/test/java/org/springframework/samples/petclinic/model/Validato
rTests.java
        modified:   src/test/java/org/springframework/samples/petclinic/vet/VetTests.j
```

```
ava

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        sql-error-codes.xml
        src/main/resources/META-INF/

no changes added to commit (use "git add" and/or "git commit -a")
```

Interesting output is produced by looking at the changes introduced in `src/main/java/org/springframework/samples/petclinic/owner/OwnerController.java`.

If you run:

```
git diff src/main/java/org/springframework/samples/petclinic/owner/OwnerController.jav
a
```

You see changes like these:

```
diff --git a/src/main/java/org/springframework/samples/petclinic/owner/OwnerControlle
r.java b/src/main/java/org/springframework/samples/petclinic/owner/OwnerController.jav
a
index 3c96327..7415003 100644
--- a/src/main/java/org/springframework/samples/petclinic/owner/OwnerController.java
+++ b/src/main/java/org/springframework/samples/petclinic/owner/OwnerController.java
@@ -17,7 +17,7 @@ package org.springframework.samples.petclinic.owner;

 import java.util.List;
 import java.util.Map;
-import javax.validation.Valid;
+import jakarta.validation.Valid;
 import org.springframework.data.domain.Page;
 import org.springframework.data.domain.PageRequest;
 import org.springframework.data.domain.Pageable;
@@ -60,14 +60,14 @@ class OwnerController {
                return ownerId == null ? new Owner() : this.owners.findById(ownerId);
        }

-       @GetMapping("/owners/new")
+       @GetMapping({"/owners/new", "/owners/new/"})
        public String initCreationForm(Map<String, Object> model) {
                Owner owner = new Owner();
                model.put("owner", owner);
                return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
        }

-       @PostMapping("/owners/new")
+       @PostMapping({"/owners/new", "/owners/new/"})
        public String processCreationForm(@Valid Owner owner, BindingResult result) {
                if (result.hasErrors()) {
                        return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
@@ -77,13 +77,13 @@ class OwnerController {
                return "redirect:/owners/" + owner.getId();
```

As you can see, `javax` imports have been replaced by the equivalent in Jakarta. Also, Spring Framework 6 introduces a runtime breaking change. See The trailing slash matching configuration option has been deprecated and its default value set to false.

With this Spring Framework change, `GET /owners/new/` no longer matches by default and results in an HTTP 404 error. To prevent this breaking change in your client applications, Spring Application Advisor is designed to apply changes to ensure that existing applications behave as before and new code can incrementally adopt the best practices.

This concludes the second step of the upgrade plan. By running the command `advisor build-config get && advisor upgrade-plan apply` for each of the remaining steps and following the same pattern of git commands, the application is fully upgraded to the latest version of Spring Boot.

Upgrade an Spring application that uses a custom Spring Boot Starter

This is a classic exercise that upgrades a Spring Boot application called `acme-bookings-app` with a dependency called `acme-boot-starter`, which is a custom Spring Boot starter.

Notice that in this exercise, both the application (`acme-bookings-app`) and the starter (`acme-boot-starteracme-bookings-app do the following:`

- `Upgrade the corresponding Java sources to consume the new Spring APIs.`
- `Upgrade the defined Spring dependencies that appears in the pom.xml or the build.gradle file.`

`However, the recipe will NEVER bump the version of the acme-boot-starter dependency because OpenRewrite has no knowledge about the following:`

- `What dependencies (specially those that are internal) are using Spring?`
- `What versions of Spring are available in every version of every dependency?`
- `What is the correct version to upgrade the application given that some dependencies have not been released for every version of Spring?`

By using Spring Application Advisor, we will learn that:

- Spring Application Advisor prevents invalid dependency changes in `acme-bookings-app`. There will not be an upgrade plan for `acme-bookings-app` unless we publish the custom upgrade mappings for `acme-boot-starter`.
- By continuously publishing your custom upgrade mappings of a project like `acme-boot-starter` in the Spring Application Advisor server, all the downstream dependencies can be automatically upgraded.

The requirements for upgrading `acme-bookings-app` are:

- Having the Spring Application Advisor server component ready to accept connections. See Install App Advisor.
- The CLI available in your `$PATH`. See Run App Advisor.

- Minimum requirement: Java SDK 17 or higher is available. Recommended: Java SDK 8, 11, and 17 are available.
- A tool to manage multiple different Java versions. In this guide, we use sdkman, but you can use any tool available.

To follow the example in this guide, clone `acme-bookings-app` and `acme-boot-starter`.

```
git clone https://github.com/rpau/acme-bookings-app
git clone https://github.com/rpau/acme-boot-starter
```

The next step is to build the artifacts of the different available versions of `acme-boot-starter` in your local machine. To do this, run the following commands from the `acme-boot-starter` directory:

```
git checkout 1.0.0
./mvnw install

git checkout 2.0.0
./mvnw install
```

These commands build the `com.acme.boot:acme-boot-starter:1.0.0` and `com.acme.boot:acme-boot-starter:2.0.0` artifacts and make them available into your local Maven repository, so they can be resolved to build `acme-bookings-app`. In a real world scenario, this step is not required because these artifacts are already available in a public or internal Maven repository.

Now, if you open the `pom.xml` of `acme-bookings-app`, you see the following dependency:

```
<dependency>
    <groupId>com.acme.boot</groupId>
    <artifactId>acme-boot-starter</artifactId>
    <version>1.0.0</version>
</dependency>
```

Verify that the dependency can be resolved by running the following command from the `acme-booking-app` directory:

```
./mvnw package
```

This should produce an output that finishes with:

```
[INFO] --- spring-boot:2.7.3:repackage (repackage) @ acme-bookings-app ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.384 s
[INFO] Finished at: 2025-01-15T12:54:24+01:00
[INFO] ------------------------------------------------------------------------
```

The `acme-booking-app` is using Spring Boot 2.7.x and we want to upgrade it to use the latest version of Spring Boot using Spring Application Advisor.

Start by running the following commands from the `acme-booking-app`:

```
advisor build-config get
advisor upgrade-plan get --url=$ADVISOR_SERVER
```

Notice that the last command prints this output:

```
The projects ["spring-framework", "spring-boot"] could not be included in the Upgrade
Plan because they are used as transitive dependencies for other projects, and no upgra
des are configured for them.
Please request your administrator to configure the projects of the following dependenc
ies:

        - com.acme.boot:acme-boot-starter
                uses:
                        - spring-framework
                        - spring-boot
                blocking upgrades for:
                        - spring-boot

In order to learn more about publishing upgrade mappings, visit https://techdocs.broad
com.com/us/en/vmware-tanzu/spring/tanzu-spring/commercial/spring-tanzu/app-advisor-cus
tom-upgrades.html

No upgrade plans available - your project seems to be up to date.
```

Spring Application Advisor is notifying you that is not safe to upgrade this application until you provide information about how to upgrade `com.acme.boot:acme-boot-starter`; because otherwise there might be inconsistent Spring Boot versions between the one that is consumed by the `acme-boot-starter` and the `acme-booking-app`.

To tell Spring Application Advisor how to upgrade `acme-boot-starter` from one version to another, you need to provide a custom upgrade mapping file. This file can initially be filled manually, according to the specification described, or using an experimental Spring Application Advisor command. To try the experimental command, run the following from any empty directory.

```
advisor mapping build -r https://github.com/rpau/acme-boot-starter --offline --url=$AD
VISOR_SERVER
```

This command calculates the dependencies, the minimum Java version, and the submodules for each of the `acme-boot-starter` Git tags that have been released.

Note that you must use the `--offline` option. This is because we want to enforce looking for the versions available in the local Maven repository that were previously built. Otherwise, the command looks in the remote Maven repository.

Sometimes, the tags available in the Git repository and the versions released might not be identical, so the commands performs some pattern matching based on real projects.

After running the command, the output should be:

```
** Downloading project acme-boot-starter from: https://github.com/rpau/acme-boot-start
er

** Detecting available versions for project: acme-boot-starter
- Versions found: [1.0, 2.0]
```

```
  Checking out version 1.0 for project acme-boot-starter:
** Successfully checked out tag '1.0.0'
** Generating Sbom for version 1.0 of project acme-boot-starter
- Re-detecting BuildTool
- Get Dependencies
- Get JavaRuntime
- Get Modules
 New mapping for version: 1.0

  Checking out version 2.0 for project acme-boot-starter:
** Successfully checked out tag '2.0.0'
** Generating Sbom for version 2.0 of project acme-boot-starter
- Re-detecting BuildTool
- Get Dependencies
- Get JavaRuntime
- Get Modules
 New mapping for version: 2.0

  Mapping file created at: .advisor/mappings/acme-boot-starter.json

  PROCESS HAS FINISHED
```

If you look at the generated file, in the `supportedGenerations` properties, you will see the following conditions:

- `acme-boot-starter:1.0.0` requires `spring-boot:2.7.x`
- `acme-boot-starter:2.0.0` requires `spring-boot:3.0.x`

You will also notice that there is an empty list for the `recipes` properties. By default, when no recipes are defined, Spring Application Advisor dynamically generates the recipes to bump the artifact versions.

To instruct the Spring Application Advisor server to load this configuration, run the following command:

```
curl -X POST -H "Content-Type: application/json" -d @./.advisor/mappings/acme-boot-sta
rter.json  $ADVISOR_SERVER/mapping/upload
```

This command sends the upgrade mappings to the Spring Application Advisor server component. The logs contain this entry:

```
2025-01-15T13:50:29.037+01:00  INFO 30017 --- [Tanzu Spring Server] [omcat-handler-4]
c.v.t.s.a.s.mapping.MappingController    : Custom Mapping file added, refreshing scope
```

The mappings are now available. If you request the upgrade plan for `acme-booking-app` (running `advisor upgrade-plan get`), the output is:

```
  Fetching and processing upgrade plan details [00m 01s] ok
- Step 1:
    * Upgrade acme-boot-starter from 1.0.x to 2.0.x
    * Upgrade spring-framework from 5.3.x to 6.0.x
    * Upgrade spring-boot from 2.7.x to 3.0.x
- Step 2:
    * Upgrade spring-boot from 3.0.x to 3.1.x
- Step 3:
    * Upgrade spring-framework from 6.0.x to 6.1.x
    * Upgrade spring-boot from 3.1.x to 3.2.x
- Step 4:
```

```
    * Upgrade spring-boot from 3.2.x to 3.3.x
- Step 5:
    * Upgrade spring-framework from 6.1.x to 6.2.x
    * Upgrade spring-boot from 3.3.x to 3.4.x
```

Note that Spring Boot can be upgraded in all the steps without requiring upgrade of `acme-boot-starter`. This is because Spring Application Advisor assumes semantic versioning, which means that the Spring libraries, `acme-boot-starter` and `acme-booking-app`, should be compatible.

And after running the following command to see the changes to upgrade to Spring Boot 3.0.x:

```
advisor upgrade-plan apply --url=$ADVISOR_SERVER
git diff
```

You will see the dependency version change for `spring-boot-starter-parent` and `acme-boot-starter` and other minor configuration changes required to upgrade this application to Spring Boot 3.0.x

```
diff --git a/pom.xml b/pom.xml
index a2c32d1..1e62a23 100644
--- a/pom.xml
+++ b/pom.xml
@@ -5,7 +5,7 @@
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
-               <version>2.7.3</version>
+               <version>3.0.18</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.acme.boot</groupId>
@@ -37,7 +37,7 @@
                 <dependency>
                         <groupId>com.acme.boot</groupId>
                         <artifactId>acme-boot-starter</artifactId>
-                        <version>1.0.0</version>
+                        <version>2.0.0</version>
                 </dependency>

                <dependency>
diff --git a/src/main/resources/application.properties b/src/main/resources/applicatio
n.properties
index 695b435..ea70a0f 100644
--- a/src/main/resources/application.properties
+++ b/src/main/resources/application.properties
@@ -1 +1,2 @@
+logging.pattern.dateformat=yyyy-MM-dd HH:mm:ss.SSS
 spring.application.name=acme-bookings-app
```

To understand how Spring Application Advisor works in all the scenarios, you can tune the `acme-boot-starter.json` that you generated previously.

Open the `acme-boot-starter.json` file and replace the contents with the following text. These new contents add a new artificial version `3.0.x` that uses `spring-boot:3.4.x`.

```
{
  "slug" : "acme-boot-starter",
  "coordinates" : [
    "com.acme.boot:acme-boot-starter"
  ],
  "repositoryUrl" : "https://github.gwd.broadcom.net/TNZ/acme-boot-starter",
  "rewrite" : {
    "1.0.x" : {
      "recipes" : [ ],
      "nextRewrite" : "2.0.x",
      "requirements" : {
        "supportedJavaVersions" : {
          "major" : 11,
          "minor" : 11
        },
        "supportedGenerations" : {
          "spring-boot" : "2.7.x"
        },
        "excludedArtifacts" : [ ]
      }
    },
    "2.0.x" : {
      "recipes" : [ ],
      "nextRewrite" : "3.0.x",
      "requirements" : {
        "supportedJavaVersions" : {
          "major" : 17,
          "minor" : 17
        },
        "supportedGenerations" : {
          "spring-boot" : "3.0.x"
        },
        "excludedArtifacts" : [ ]
      }
    },
    "3.0.x" : {
     "recipes" : [ ],
     "nextRewrite" : null,
     "requirements" : {
       "supportedJavaVersions" : {
         "major" : 17,
         "minor" : 17
       },
       "supportedGenerations" : {
         "spring-boot" : "3.4.x"
       },
       "excludedArtifacts" : [ ]
     }
    }
  }
}
```

After you replace the contents, publish the new version to the server by running the `curl` command again.

```
curl -X POST -H "Content-Type: application/json" -d @./.advisor/mappings/acme-boot-sta
rter.json  $ADVISOR_SERVER/mapping/upload
```

At this point, calculate the upgrade plan for `acme-boot-starter` again with the following commands:

```
advisor build-config get
advisor upgrade plan get --url=$ADVISOR_SERVER
```

The following output is generated, which shows upgrades of the Spring Boot version from `3.0.x` to `3.4.x` instead of having a step for each of the versions: `3.1.x`, `3.2.x`, `3.3.x`, and `3.4.x`. This is because Spring Application Advisor is trying to align the higher number of projects together, because it means that their versions have been tested together. In other words, Spring Application Advisor is trying to create coherence between different project upgrades that might have been released with a different cadence.

```
 Fetching and processing upgrade plan details [00m 01s] ok
- Step 1:
   * Upgrade acme-boot-starter from 2.0.x to 3.0.x
   * Upgrade spring-framework from 6.0.x to 6.2.x
   * Upgrade spring-boot from 3.0.x to 3.4.x
```

Spring Application Advisor Architecture

To help you understand how the Spring Application Advisor works and how it interacts with your environment and services, this topic:

- Explains how Spring Application Advisor fits into your software delivery lifecycle (SDLC)
- Provides an architecture diagram that shows how data flows through the Spring Application advisor components and your system

How Spring Application Advisor fits into your software delivery lifecycle (SDLC)

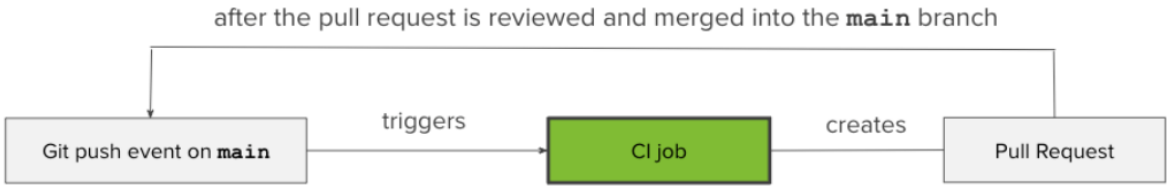
Spring Application Advisor is designed to create automatic pull requests that incrementally upgrade your Spring applications. Pull requests are requests to review a new contribution to a repository. This is where the information is shared among reviewers, and where multiple manual and automatic checks are executed to prevent causing a broken application.

Spring Application Advisor creates a new branch in the Git repository every time a new upgrade opportunity is detected, so engineering team members with write access in the repository can review and adapt the requested changes before integrating them into the main branch.

Spring Application Advisor is designed to run in a CI/CD environment with a native CLI every time new code changes have been integrated into the main branch, so there is continuous checking for available incremental Spring upgrades.

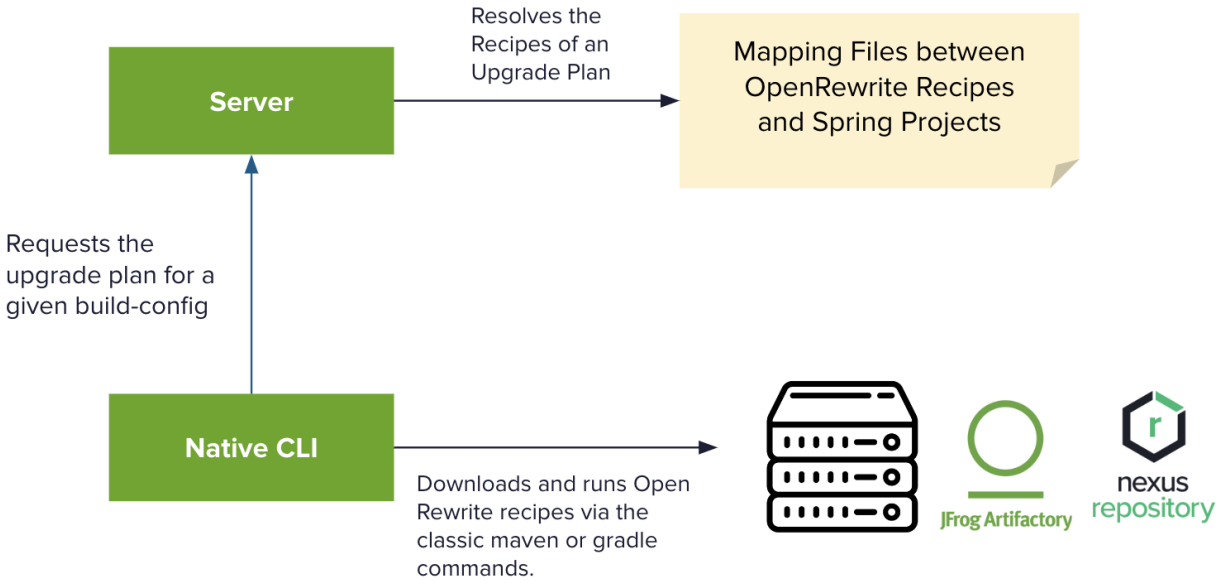
To resolve whether there are incremental upgrades available (for example, from Spring Boot 2.6 to Spring Boot 2.7), Spring Application Advisor checks for the current version of your dependencies and build tools. To retrieve this information with accuracy and to prevent CI failures after an upgrade, Spring Application Advisor must run in a development environment that always has access to your enterprise Maven repositories. This environment is usually the CI/CD environment.

The described flow appears in this diagram:



Architecture diagram

The graphic shown in this section is a high-level architecture diagram that shows the flow of data between Spring Application Advisor and a typical customer environment. Arrows indicate communication between components.



Spring Application Advisor has two main components:

- The Server, which uses a set of mapping files to resolve the OpenRewrite recipes to be applied in an upgrade plan.
  The Server does not require any external Internet connectivity or storage.
- The Native CLI, which requests upgrade plans to the server and runs the OpenRewrite recipes associated to a plan using the classic Maven and Gradle plugins.
  The Native CLI needs connection to the server and the preferred artifact manager tool (Nexus or Artifactory, for example) that is able to resolve the artifacts that contain the recipes returned by the server.

Spring Application Advisor upgrades the source code running OpenRewrite recipes from the CLI, so no source code is transferred from the CLI location to the server. The Server resolves which OpenRewrite recipes need to be executed, but the artifacts in which those recipes are stored are downloaded from the Maven repositories that you have configured in your environment.

For more information:

- To install the server, see How to install Spring Application Advisor.

- To connect the CLI to the server, see How to run the CLI.
- To understand how to integrate the Maven repository into your environment, see Spring Enterprise Subscription for Artifact Repository Administrators. Spring Application Advisor executes commercial recipes that are available in the Spring Commercial repository.

Installing Spring Application Advisor

This topic provides the steps for installing Spring Application Advisor.

Download and Start the Spring Application Advisor Server

The server component of Spring Application Advisor requires Java 17 or higher.

To get the server component, download the following artifact from the Spring Enterprise Maven repository.

Run:

```
curl -L -H "Authorization: Bearer $ARTIFACTORY_TOKEN" -o spring-server.jar -X GET http
s://packages.broadcom.com/artifactory/spring-enterprise/com/vmware/tanzu/spring/tanzu-
spring-server/1.1.2/tanzu-spring-server-1.1.2.jar
```

For information about syncing your internal repository with the Spring Enterprise Maven repository, see Spring Enterprise Subscription.

After the server is downloaded and renamed, you can start the server by running the following command, which by default, will start the service listening on port 8080:

```
java -jar -Dserver.port=9003 spring-server.jar
```

After the process starts, you can check the status in `http://localhost:9003/actuator/health`.

This solution does not require Internet connectivity.

If you need to expose the endpoints in a particular route, you can run the server with the property `spring.advisor.server.prefix` set to a specific path.
Remember to include this path as part of the server URL (referenced in this documentation as `ADVISOR_SERVER`) when executing the Spring Application Advisor CLI.

```
java -jar spring-server.jar --spring.advisor.server.prefix="/api/adviso
r"
```

Running Spring Application Advisor CLI

The Spring Application Advisor CLI is a native CLI that supports the following commands:

- `build-config`
- `upgrade-plan`

```
Usage: advisor [COMMAND]
Spring Application Advisor CLI
Commands:
  build-config  Project build dependencies and tools
  upgrade-plan  Retrieves or applies upgrade plan(s) to project
```

Download the CLI

The CLI is currently available only for Linux and MacOs.

To download the CLI, run:

For Linux:

```
curl -L -H "Authorization: Bearer $ARTIFACTORY_TOKEN" -o advisor-cli.tar -X GET http
s://packages.broadcom.com/artifactory/spring-enterprise/com/vmware/tanzu/spring/applic
ation-advisor-cli-linux/1.1.3/application-advisor-cli-linux-1.1.3.tar
tar -xf advisor-cli.tar --strip-components=1 --exclude=./META-INF
```

For MacOS Intel:

```
curl -L -H "Authorization: Bearer $ARTIFACTORY_TOKEN" -o advisor-cli.tar -X GET http
s://packages.broadcom.com/artifactory/spring-enterprise/com/vmware/tanzu/spring/applic
ation-advisor-cli-macos/1.1.3/application-advisor-cli-macos-1.1.3.tar
tar -xf advisor-cli.tar --strip-components=1 --exclude=./META-INF
```

For MacOS ARM64:

```
curl -L -H "Authorization: Bearer $ARTIFACTORY_TOKEN" -o advisor-cli.tar -X GET http
s://packages.broadcom.com/artifactory/spring-enterprise/com/vmware/tanzu/spring/applic
ation-advisor-cli-macos-arm64/1.1.3/application-advisor-cli-macos-arm64-1.1.3.tar
tar -xf advisor-cli.tar --strip-components=1 --exclude=./META-INF
```

Configure the Maven settings to download the commercial recipes

For you to be able to upgrade your Spring Applications, the Application Advisor CLI must be able to download artifacts from the Spring Maven Enterprise repository. Ensure that your Maven repositories are configured correctly. See Running commercial recipes using OpenRewrite tools.

Produce a build configuration

A build configuration contains:

- The dependency tree using the CycloneDX format
- The Java version required to compile the sources
- The build tool versions

To produce the build configuration, run:

```
advisor build-config get
```

This command produces this output:

```
Resolving the build configuration of $path.

  [ 1 / 3 ]  Resolving dependencies with “maven/gradle command” [3m 2s] ok
  [ 2 / 3 ]  Resolving JDK version  [4s]
  [ 3 / 3 ]  Resolving build tool  [1s]

  Build configuration generated at $path/.advisor/build-config.json
  Errors
- $repo failed with the following message:
The maven command failed. You can find the error in .advisor/errors/${error-id}.log
```

The build configuration is produced as a HSON file in an internal folder called `.advisor`. If the folder already contains a build configuration, it will be overwritten.

Publish a build configuration

Use this command to publish the generated build configuration to the Spring Application Advisor server:

```
advisor build-config publish --url=${ADVISOR_SERVER}
```

Where `ADVISOR-SERVER` is the URL of the Server where the Application Advisor is installed.

Generate an upgrade plan

This command provides the step-by-step upgrade plan showing the Spring projects that need to be upgraded, and to what versions.

```
advisor upgrade-plan get --url=${ADVISOR_SERVER}
```

Where `ADVISOR-SERVER` is the URL of the Server where the Application Advisor is installed.

The output looks something like this:

```
Fetching details for upgrade plan:
 - Step 1:
      * Upgrade Spring Boot from v2.6.1 to v2.7.x
      * Upgrade Spring Framework from v3.5.1 to v4.0.x
 - Step 2:
      * Upgrade Java from 8 to 11
 - Step 3:
      * Upgrade Spring Boot from v 2.7.1 to v3.0.x
```

Apply an upgrade plan from your local machine

The following command can upgrade the files locally on your machine. Then you can manually review them to decide if you want to integrate Spring Application Advisor pull requests into your repository.

```
advisor upgrade-plan apply --url=${ADVISOR_SERVER}
```

Where `ADVISOR-SERVER` is the URL of the Server where the Application Advisor is installed.

Spring Application Advisor preserves your coding style by making the minimum required changes in the source files. However, if you are using a Maven or Gradle formatter like `spring-javaformat` for your repository, add the `--after-upgrade-cmd` option to the `advisor upgrade-plan apply` command as follows:

```
advisor upgrade-plan apply --url=${ADVISOR_SERVER} --after-upgrade-cmd=${MAVEN_OR_GRAD
LE_FORMATTER_TASK}
```

Where:

- `ADVISOR-SERVER` is the URL of the Server where the Application Advisor is installed.
- `MAVEN_OR_GRADLE_FORMATTER_TASK` is the Maven or Gradle formatting task.

For example, for `spring-javaformat`, use:

```
advisor upgrade-plan apply --url=$https://appadvisorserver.company.org --build-tool-ru
n-cmd=spring-javaformat:apply
```

Increasing memory limit

The Advisor CLI runs Gradle to get Build Configuration and apply recipes.

Gradle will run a separate process, `daemon`, to use the local configuration, depending on the project. The Java VM used by the daemon limits memory to 512 MegaBytes by default. However, it can also provide other default options.

When the target project to upgrade is large, it may be necessary to increase the default memory limit. For a Gradle build, use `org.gradle.vmargs`.

For example, if you want to increase the memory limit to 1 GigaByte, run:

```
upgrade-plan apply --url <url>
--build-tool-jvm-args="-Dorg.gradle.jvmargs=-Xmx1g"
```

You can also change the Garbage Collector:

```
upgrade-plan apply --url <url>
--build-tool-jvm-args="-Dorg.gradle.jvmargs=-Xmx2g -XX:+UseParallelGC"
```

Enable continuous and incremental upgrades

To enable continuous and incremental upgrades with automatic pull requests:

1. If you are using GitLab Enterprise, GitHub Enterprise, or Jenkins, check that your pipelines are executing the following command:

   ```
   advisor upgrade-plan apply --push --from-yml --url=${ADVISOR_SERVER}
   ```

   Where `ADVISOR-SERVER` is the URL of the Server where the Application Advisor is installed.

   If not, integrate the following commands in your CI/CD pipeline:

```
advisor build-config get
advisor build-config publish --url=${ADVISOR_SERVER}
advisor upgrade-plan apply --push --from-yml --url=${ADVISOR_SERVER}
```

Where `ADVISOR-SERVER` is the URL of the Server where the Application Advisor is installed.

2. Verify or create the `GIT_TOKEN_FOR_PRS` environment variable for your CI/CD build. The value should be an access token with write access to the repository. Spring Application Advisor creates a branch in the repository and a makes a new pull request against the branch.
3. Add a file named `.spring-app-advisor.yml` in the root directory of your repository with the following contents:

```
enabled: true
```

Integrating Spring Application Advisor with CI/CD

These topics provide the steps for integrating Spring Application Advisor:

- with GitLab Enterprise
- with GitHub Enterprise
- with Jenkins
- with other SaaS CI/CD tools

Integrating with Spring Application Advisor in GitLab Enterprise

This topic provides the steps for integrating Spring Application Advisor with your CI/CD pipelines in GitLab Enterprise. It explains how to automatically integrate Spring Advisor after every build so that manual changes are not required in every pipeline.

Step 1: Create a Custom GitLab Runner using GKE

There are multiple GitLab runners. This section explains the easiest way to integrate the Spring Application Advisor CLI without having to edit the CI/CD pipelines: the Custom GitLab Runner Executor.

This topic illustrates the required steps for Google Cloud, but it can be configured in any environment.

1. Create a new Virtual Machine in GKE: **Compute Engine - Virtual Machines** using an Ubuntu image (available under the **Boot disk** section).
2. Edit the `/etc/hosts` to reference the GitLab Instance, if it's not public:

```
<IP VALUE> gitlab.acme.com
```

3. Install the `gitlab-runner` utility for Ubuntu at `/home/` folder:

```
cd /home/
curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/
script.deb.sh" | sudo bash
sudo apt-get install gitlab-runner
```

4. Go to your GitLab instance as an administrator, and scroll to the **Admin** section at the bottom of the screen. Create a new Runner at: **CI/CD - Runners - New Instance Runner**.
5. Click **Linux** and **Run untagged jobs**.
6. Get the token and register the runner with type `custom` and assign a name at the machine by running:

   ```
   sudo gitlab-runner register --url https://gitlab.acme.com  --token MY_SECRET_TO
   KEN --tls-ca-file gitlab.acme.com.crt
   ```

   The certificate is available to export from the GitLab instance using any web browser at: https://gitlab.acme.com. You can upload it to the machine by running:

   ```
   gcloud compute scp LOCAL-DIRECTORY/gitlab.acme.com.crt root@"MACHINE-NAME":/hom
   e/ --zone "us-central1-a" --project "app-advisor"
   ```

This generates a config file: `/etc/gitlab-runner/config.toml`.

Step 2: Invoke the Advisor CLI from the Custom GitLab Runner

Now that you have created a custom GitLab runner, you need to configure it to run Spring App Advisor.

Follow these steps:

1. Edit the file generated in the previous step: `/etc/gitlab-runner/config.toml`. Use the following content to configure the custom runner to execute a script.

   ```
   concurrent = 1
   check_interval = 0
   shutdown_timeout = 0

   [session_server]
   session_timeout = 1800

   [[runners]]
   name = "instance-for-gitlab-runner"
   url = "https://gitlab.acme.com"
   token = "" # From GitLab Runner Instance config
   tls-ca-file = "gitlab.acme.com.crt"
   executor = "custom"
   builds_dir = "/home/gitlab/builds"
   cache_dir = "/home/gitlab/cache"
   [runners.cache]
       MaxUploadedArchiveSize = 0
   [runners.custom]
       run_exec = "/home/gitlab/advisor_exec.sh"
   ```

2. Create the folders to be used in the script:

```
sudo mkdir /home/gitlab/builds
sudo chmod -R 777 /home/gitlab/builds

sudo mkdir /home/gitlab/cache
sudo chmod -R 777 /home/gitlab/cache
```

3. Add a Maven settings file to let Spring Application Advisor connect to the Spring Maven repositories and run the commercial Spring recipes. The Maven settings file should be located in `/home/gitlab/.m2/settings.xml`. Use the example provided in Running commercial recipes using OpenRewrite tools
4. Copy and upload the next script (`advisor_exec.sh`) and the CLI binary (for Linux).

```
#!/bin/bash

readonly SCRIPT="$1"
readonly ACTION="$2"

cp /home/gitlab/.m2/settings.xml /root/.m2/settings.xml
export GIT_TOKEN_FOR_PRS="${GIT_TOKEN_FOR_PRS:-undefined}"

run_advisor() {
    echo "Project downloaded from git at: $CUSTOM_ENV_CI_PROJECT_DIR"
    echo "Running Spring Advisor CLI"
    /home/gitlab/advisor build-config get
    /home/gitlab/advisor build-config publish --url=${SERVER}
    /home/gitlab/advisor upgrade-plan apply --push --from-yml --url=$SERVER
}

case "${ACTION}" in
"cleanup_file_variables")
run_advisor
;;
*)
. "$SCRIPT" "$ACTION"
;;
esac
```

5. Next, assign permissions:

```
gcloud compute scp LOCAL-DIRECTORY/advisor_exec.sh root@"MACHINE-NAME":/home/gi
tlab --zone "us-central1-a" --project "app-advisor"
gcloud compute scp LOCAL-DIRECTORY/advisor root@"MACHINE-NAME":/home/gitlab --z
one "us-central1-a" --project "app-advisor"
```

```
sudo chmod +x /home/gitlab/advisor_exec.sh
sudo chmod +x /home/gitlab/advisor
```

The script detects the phase of the custom runner execution and during the cleanup phase, it executes the CLI.

6. Make the runner available for the GitLab instance:

```
sudo gitlab-runner run NAME-OF-THE-RUNNER
```

Step 3: Check that your GitLab pipelines run Spring Application Advisor at the end

1. Go to the GitLab Instance and run a job to check that everything works. This step assumes that the repository containing the job already has a pipeline configured (`.gitlab-ci.yml`).

Integrating with Spring Application Advisor in GitHub Enterprise

This topic provides the steps for integrating Spring Application Advisor with your CI/CD pipelines in GitHub Enterprise.

You can automatically execute scripts on a self-hosted runner, either before a job runs, or after a job finishes running. For instructions for creating a self-hosted runner, see the official GitHub documentation.

1. Modify the `ACTIONS_RUNNER_HOOK_JOB_COMPLETED` environment variable. There are two ways to set this environment variable:
    - Add it to the operating system:

      ```
      ACTIONS_RUNNER_HOOK_JOB_COMPLETED=/opt/runner/advisor_script.sh
      ```

    - Add it to a file named `.env` in the self-hosted runner application directory.

      Create the `advisor_script.sh` file with the following contents:

      ```
      #!/bin/bash
      # This script assumes that advisor CLI is in the $PATH

      export GIT_TOKEN_FOR_PRS = **WRITE_GIT_ACCESS_TOKEN**

      # Check that the $HOME/.m2/settings is using the Spring Commercial reposi
      tory

      advisor build-config get
      advisor build-config publish --url=${ADVISOR_SERVER}
      advisor upgrade-plan apply --push --from-yml --url=${ADVISOR_SERVER}
      ```

      Ensure that the script has execution permissions.

      ```
      chmod u+x /opt/runner/advisor_script.sh
      ```

Integrating with Spring Application Advisor in Jenkins

This topic provides the steps for integrating Spring Application Advisor with your CI/CD pipelines in Jenkins.

Before integrating Spring Application Advisor, check that Jenkins is configured to use the Spring commercial repository in a shared Maven `settings.xml` file.

See the CloudBees official guide for details about how to provide a shared Maven settings file.

See Running commercial recipes using OpenRewrite tools for a working Maven `settings.xml` file to connect to the Spring commercial repository.

Using Pipeline Templates

In Jenkins, Pipeline Templates help ensure that pipeline builds conform to organizational standards. Central or platform teams can create their own standards using a Pipeline Template. For information about how pipeline templates work, refer to the CloudBees documentation.

- If you are already using CloudBees Pipeline Templates, you can adapt your existing template to include the Spring Application Advisor CLI commands. For example:

```
pipeline {
    agent any

    environment {
    ADVISOR_SERVER = 'http://advisor.acme.com'
    GIT_TOKEN_FOR_PRS = credentials('advisor_git_token_for_prs')
    }
    stages {
        stage(my-stage) {
            steps {
                …
            }
        }
        …
        stage('spring-app-advisor') {
            steps {
                sh 'advisor build-config get'
                sh 'advisor build-config publish --url=$ADVISOR_SERVER'
                sh 'advisor upgrade-plan apply --push --from-yml --url=$ADVISOR
_SERVER' --token=$GIT_TOKEN_FOR_PRS
            }
        }
}
}
```

- If you are not using a CloudBees Pipeline Template, create a new Template, and then create a new job for each of the repositories.

Integrating with Other SaaS CI/CD Tools

For SaaS tools, there is no way to embed a binary in all the builds without altering references in the CI/CD pipeline. Because every CI/CD engine has its own syntax and vocabulary, this topic explains a script-based approach that you must adapt for your solution.

Set up for script execution

The goal is to execute these CLI commands at the end of the build of the `default` or `main` branch:

```
...download and extract the advisor CLI…
export GIT_TOKEN_FOR_PRS = **WRITE_GIT_ACCESS_TOKEN**

advisor build-config get
advisor build-config publish --url=${ADVISOR_SERVER}
advisor upgrade-plan apply --push --from-yml --url=${ADVISOR_SERVER}
```

To set this up:

1. Configure the `GIT_TOKEN_FOR_PRS` environment variable. This must be an access token with write access to the analyzed repository. This is to allow creation of automatic pull requests for upgrading your Spring dependencies, if needed. Developers decide if they want to receive these pull requests by adding a file named `.spring-app-advisor.yml` in the root directory. For more information, see Enable continuous and incremental upgrades with automatic pull requests.
2. Replace `${ADVISOR_SERVER}` with the full URL of your server. For example: `https://advisor.acme.com`.

GitHub Actions

This section shows how to apply the script-based approach in the context of GitHub Actions. Note that there is no concrete Java version requirement for running Application Advisor. It just needs to be consistent with the project requirements.

```
name: Spring App Advisor Workflow

on:
  push:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest
    permissions:
      contents: read

    steps:
    - uses: actions/checkout@v4
    - name: Set up JDK
      uses: actions/setup-java@v4
      with:
        java-version: '17'
        distribution: 'temurin'
    - name: Generates Maven Settings
      uses: 's4u/maven-settings-action@v3.0.0'
      with:
        servers: '[{"id": "tanzu-spring-release", "username":  "${{ secrets.BC_USER
}}", "password": "${{ secrets.$BC_PWD }}"}]'
        repositories: '[{"id":"tanzu-spring-release", "name":"Spring Enterprise Suppor
ted Releases","url":"https://packages.broadcom.com/artifactory/spring-enterprise","sna
pshots":{"enabled":false}}]'
    - name: Runs Spring Application Advisor
          env:
           GIT_TOKEN_FOR_PRS: ${{ secrets.advisor_git_token_for_prs }}
           ADVISOR_SERVER: ${{ secrets.advisor_server }}
           ARTIFACTORY_TOKEN: ${{ secrets.advisor_artifactory_token }}
          run: |
            curl -L -H "Authorization: Bearer $ARTIFACTORY_TOKEN"  -o advisor-linux.ta
r -X GET https://packages.broadcom.com/artifactory/spring-enterprise/com/vmware/tanzu/
spring/application-advisor-cli-linux/1.1.3/application-advisor-cli-linux-1.1.3.tar
            tar -xf advisor-linux.tar --strip-components=1 --exclude=./META-INF
            ./advisor build-config get
```

```
            ./advisor build-config publish --url=$ADVISOR_SERVER
            ./advisor upgrade-plan apply --push --from-yml --url=$ADVISOR_SERVER --tok
en=$GIT_TOKEN_FOR_PRS
    - name: Get errors if exist
      if: ${{ hashFiles('.advisor/errors/') != '' }}
      run: |
        cat .advisor/errors/*
```

Custom upgrades using Spring Application Advisor

Most organizations have shared Java libraries and components across multiple Spring applications. If these shared components use Spring libraries, Spring Application Advisor prevents, by default, upgrading these applications to prevent the introduction of incompatible Spring versions in the classpath.

For example, if you have an application that depends on an internal library called `acme-spring-commons` allocated in a different Git repository and that library uses `spring-boot` 2.7.x, this application cannot upgrade to `spring-boot` 3.0.x until after that library has been upgraded and released with `spring-boot` 3.0.x.

To allow Spring applications to upgrade the Spring libraries when you upgrade your shared libraries, you must configure the upgrade mappings for those shared libraries in the server.

Configure the upgrade plan for shared libraries

To start, create a `custom-upgrades-mappings.json` file. Copy and adapt the following example:

```
{
  "rewriteArtifacts": [
      {
        "coordinates": "com.acme.recipes:acme-spring-recipes:5.22.0",
        "minimalJavaVersion": "8"
      }
   ],
  "projects": [
    {
      "slug": "project-name",
      "coordinates": [
        "com.acme.project:project-module1",
        "com.acme.project:project-module2",
        ...
      ],
      "repositoryUrl": "https://github.com/acme/project-name",
      "rewrite": {
        "1.0.x": {
          "recipes": [
            {
              "name": "com.acme.recipes.project.UpgradeProject2_0"
            }
          ],
          "requirements": {
            "supportedJavaVersions": {
              "major": 21,
              "minor": 17
            }
            "supportedGenerations": {
```

```
                "spring-boot": "3.0.x",
                "spring-security": "6.0.x",
                "spring-security-rsa": "1.0.x",
                "spring-integration": "6.0.x",
                "spring-retry": "2.0.x"
              }
            },
            "nextRewrite": "2.0.x"
          },
          "2.0.x": {
            "recipes": [],
            "requirements": {
              "supportedJavaVersions": {
                "major": 21,
                "minor": 17
              },
              "supportedGenerations": {
                "spring-boot": "3.2.x",
                "spring-security": "6.2.x",
                "spring-security-rsa": "1.1.x",
                "spring-integration": "6.2.x",
                "spring-retry": "2.0.x"
              }
              "excludedArtifacts": [
                "com.acme.project:project-module2"
              ]
            },
            "nextRewrite": null
          }
        }
      }
    ]
}
```

In this example, the configuration contains the following properties.

Property	Function
`rewriteArtifacts[*].coordinates`	Required. The Maven identifier for the artifact that contains the OpenRewrite recipes. If you need OSS OpenRewrite recipes, you do not need to define those coordinates.
`rewriteArtifacts[*].minimalJavaVersion`	Required. The required minimalJavaVersion to run the recipes. Spring Application advisor ignores the coordinates that cannot be applied in a repository that uses an older Java version.
`projects[*].slug`	Required. The unique project name. Usually corresponds to the name of the Git repository that contains the shared libraries
`projects[*].coordinates`	Required. The list of `groupId:artifactId` of the coordinates used to reference the Java modules of the same Git repository
`projects[*].repositoryUrl`	Optional. The URL pointing to the Git repository of the shared libraries.
`projects[*].rewrite`	Required. Contains the requirements and the OpenRewrite recipes to upgrade from a specific version, specified as a JSON object key, to the target version, specified as `nextRewrite`.

Property	Function
`projects[*].rewrite.$version.recipes`	Required. Array of OpenRewrite recipes that need to be executed simultaneously to upgrade. If the array is empty, by default, the versions of the selected coordinates are upgraded to the selected `nextRewrite` version.
`projects[*].rewrite.$version.requirements.supportedJavaVersions.major`	Required. The major Java version required to run the coordinates in the selected `$version`
`projects[*].rewrite.$version.requirements.supportedJavaVersions.minor`	Required. The minor Java version required to run the coordinates in the selected `$version`
`projects[*].rewrite.$version.requirements.supportedGenerations`	Required. The list of projects and versions that this project, under the selected `$version`, is consuming
`projects[*].rewrite.$version.requirements.excludedArtifacts`	Optional. The list of coordinates that are no longer available in the selected `$version`. An application cannot be upgraded if these are consumed.

Alternatively, especially because it can be a very tedious task to resolve all the requirements of each version, you can start using the experimental command `advisor mapping build` that is currently available for non-Spring projects whose libraries for each of the versions are available in Maven central or offline, which means that those libraries are in the local folder `${HOME}/.m2/repository`.

The command to generate the mappings for a repository is executed as follows:

```
advisor mapping build --repository=${REPO_URL} --url=${ADVISOR_SERVER} [--offline]
```

Update the server configuration

There are three options for updating the upgrade mappings in the server.

Providing upgrade mappings stored in the file system

This option is only useful if you want to test the upgrade plans and code changes introduced after adding specific upgrade mappings without impacting developer teams.

To configure the server with specific upgrade mappings for your shared libraries/components:

1. Create a new environment variable called `SPRING_ADVISOR_MAPPING_CUSTOM_0_FILEPATH` with the path of your mapping file relative to the location where the server has started. For example:

   ```
   export SPRING_ADVISOR_MAPPING_CUSTOM_0_FILEPATH=relative/path/mapping.json
   ```

2. Restart the server to allow it to read the environment variable.

Providing upgrade mappings located in a Git repository

This option is useful for maintaining the upgrade mappings of OSS projects for which your organization does not own the release process, but which your Spring applications are consuming. The server reloads the mappings on a regular basis (daily, by default). So modifications in those Git repositories are reloaded without requiring any explicit request.

To configure the server with specific upgrade mappings located in a Git repository, follow the next steps:

1. Create these two environment variables for each of the Git repositories you want to configure. If you have multiple Git repositories to configure, use the number that appears in the environment variable name as an index for each of the mappings.

   ```
   export SPRING_ADVISOR_MAPPING_CUSTOM_0_GIT_URI=https://github.com/org/repo.git
   export SPRING_ADVISOR_MAPPING_CUSTOM_0_GIT_FILEPATH=.advisor/mappings/project.j
   son
   ```

   If the repository is private, a token can be specified.

   ```
   export SPRING_ADVISOR_MAPPING_CUSTOM_0_GIT_TOKEN=${MY_GIT_TOKEN}
   ```

   Optionally, a branch can be set.

   ```
   export SPRING_ADVISOR_MAPPING_CUSTOM_0_GIT_BRANCH=notmain
   ```

2. If you want to modify the frequency at which the mappings are reloaded (the default is daily), set a Cron expression in the `SPRING_ADVISOR_MAPPING_COORDINATES_RELOAD_SCHEDULE` environment variable.

   ```
   export SPRING_ADVISOR_MAPPING_COORDINATES_RELOAD_SCHEDULE=0 0 0 * * *
   ```

3. Restart the server to allow it to read the environment variable.

Providing upgrade mappings located in JFrog Artifactory

App Advisor supports storing and retrieving custom upgrade mappings from JFrog Artifactory generic repositories. This approach enables centralized management of upgrade mappings across your organization.

1. Organize your mapping files using the following structure, where each dependency version has its own mapping file

   ```
   acme-mappings
   ├── com.test.acme
   │   └── weather-service
   │       ├── 1.0.0
   │       │   └── weather-service.json
   │       └── 1.0.1
   │           └── weather-service.json
   └── com.test.acme
       └── booking-service
           └── 1.0.0
               └── booking-service.json
   ```

   Mapping files must use SemVer versioning.

   Create the following environment variables:

```
export ARTIFACTORY_TOKEN=mysecrettoken
export ARTIFACTORY_URI=https://internal.packages.acme.com
export ARTIFACTORY_REPOSITORY=acme-mappings-generic-local
```

Use the following curl command to upload each mapping file:

```
curl -i -H "Authorization: Bearer $ARTIFACTORY_TOKEN" \
-XPUT "${ARTIFACTORY_URI}/${ARTIFACTORY_REPOSITORY}/acme-mappings/com.test.acm
e/weather-service/1.0.0/weather-service.json" \
-d @/myuser/acme-mappings/com.test.acme/weather-service/1.0.0/weather-service.j
son
```

Uploads can be tested by running:

```
curl -H "Authorization: Bearer $ARTIFACTORY_TOKEN" "${ARTIFACTORY_URI}/artifact
ory/api/storage/${ARTIFACTORY_REPOSITORY}/acme-mappings/com.test.acme/weather-s
ervice\?list\&deep\=1"
```

2. Set the following environment variables to enable custom mapping retrieval:

```
export SPRING_ADVISOR_MAPPING_CUSTOM_0_ARTIFACTORY_URI=https://internal.package
s.acme.com
export SPRING_ADVISOR_MAPPING_CUSTOM_0_ARTIFACTORY_TOKEN=${ARTIFACTORY_TOKEN}
export SPRING_ADVISOR_MAPPING_CUSTOM_0_ARTIFACTORY_REPOSITORY=acme-mappings-gen
eric-local
export SPRING_ADVISOR_MAPPING_CUSTOM_0_ARTIFACTORY_GAV=com.test.acme:weather-se
rvice
```

Providing upgrade mappings using HTTP

If you want to update the mappings without having to restart the sever, follow these steps:

1. Extract the specific project you want to update/add in a JSON file. The default upgrade mappings for all versions of a project can be automatically generated with the experimental command `advisor mapping build`.

```
advisor mapping build --repository=${REPO_URL} --url=${ADVISOR_SERVER} [--offli
ne]
```

Every time this command is executed, all of the upgrade mappings for each of the versions are generated from scratch. The contents of the JSON file produced include only a project JSON object. For example,

```
{
  "slug": "project-name",
  "coordinates": [
    "com.acme.project:project-module1",
    "com.acme.project:project-module2",
    ...
  ],
  "repositoryUrl": "https://github.com/acme/project-name",
  "rewrite": {
    "1.0.x": {
      "recipes": [
```

```
      ],
      "requirements": {
        "supportedJavaVersions": {
          "major": 21,
          "minor": 17
        }
        "supportedGenerations": {
            "spring-boot": "3.0.x",
            "spring-security": "6.0.x",
            "spring-security-rsa": "1.0.x",
            "spring-integration": "6.0.x",
            "spring-retry": "2.0.x"
        }
      },
      "nextRewrite": null
    }
  }
}
```

2. To continuously upgrade these in your CI/CD process when a new release is available, VMware recommends that you create a pull request with the generated contents and manually merge the new additions. Consider testing the upgrade plans with the new configuration before integrating them into production.
3. Optionally, add your custom OpenRewrite recipes for upgrading any of the identified versions, add those in the corresponding versions in a different JSON file in a Git repository, and ask your administrator to configure it in the server. However, if you are using basic Java or text recipes (for example, to change a package, rename a class, and so on), this step is not required.

```
{
  "rewriteArtifacts": [
      {
        "coordinates": "com.acme.recipes:acme-spring-recipes:5.22.0",
        "minimalJavaVersion": "8"
      }
   ]
}
```

4. After you are happy with the mappings, send the mappings using HTTP, using the `curl` command, for example.

```
curl -X POST -H "Content-Type: application/json" -d @./.advisor/mappings/my-pro
ject.json  ${ADVISOR_SERVER}/mapping/upload
```

   This command automatically stores the upgrade mappings in the file system of the server and is reloaded everytime is restarted.

Running commercial recipes using OpenRewrite tools

OpenRewrite is an Open Source Software (OSS) application used for automatically refactoring source code. Spring Application Advisor combines OSS recipes with commercial recipes built by the Spring team. These commercial recipes are available only in the Spring Commercial repository. To check how to configure your environment to run the commercial recipes, we recommend following one of the options described in Spring Enterprise Subscription for Application Developers.

This topic provides instructions for running the Spring Commercial OpenRewrite recipes to upgrade Spring applications. The published recipes use `org.openrewrite.recipe:rewrite-recipe-bom`: 2.22.0.

There are several options for running OpenRewrite recipes. For simplicity, instructions are provided only for the OpenRewrite Maven Plugin.

Upgrade to Spring Boot 3.0.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot30.UpgradeSpringBoot_3_0
```

The commercial recipes included are described in Recipes for Spring Boot 3.0.x.

To apply the Spring Boot release train, use the following command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot30.BootReleaseTrain_3_0
```

Upgrade to Spring Boot 3.1.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot31.UpgradeSpringBoot_3_1
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Boot 3.1.x

To apply the Spring Boot release train, use the following command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot31.BootReleaseTrain_3_1
```

Upgrade to Spring Boot 3.2.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot32.UpgradeSpringBoot_3_2
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Boot 3.2.x

To apply the Spring Boot release train, use the following command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
```

```
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot32.BootReleaseTrain_3_2
```

Upgrade to Spring Boot 3.3.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot33.UpgradeSpringBoot_3_3
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Boot 3.3.x

To apply the Spring Boot release train, use the following command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot33.BootReleaseTrain_3_3
```

Upgrade to Spring Boot 3.4.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot34.UpgradeSpringBoot_3_4
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Boot 3.4.x

To apply the Spring Boot release train, use the following command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.boot34.BootReleaseTrain_3_4
```

Upgrade to Spring Security 5.8.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.security58.UpgradeSpringSecu
rity_5_8
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Security 5.8.x

Upgrade to Spring Security 6.0.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
```

```
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.security60.UpgradeSpringSecu
rity_6_0
```

This recipe does not include additional commercial recipes.

Upgrade to Spring Security 6.1.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.security60.UpgradeSpringSecu
rity_6_1
```

This recipe does not include additional commercial recipes.

Upgrade to Spring Security 6.2.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.security60.UpgradeSpringSecu
rity_6_2
```

This recipe does not include additional commercial recipes.

Upgrade to Spring Security 6.3.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.security60.UpgradeSpringSecu
rity_6_3
```

This recipe does not include additional commercial recipes.

Upgrade to Spring Data 3.0.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.data30.UpgradeSpringData_3_0
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Data 3.0.x

Upgrade to Spring Framework 6.0.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.framework60.UpgradeSpringFra
mework_6_0
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Framework 6.0.x

Upgrade to Spring Framework 6.1.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.framework61.UpgradeSpringFra
mework_6_1
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Framework 6.1.x

Upgrade to Spring Framework 6.2.x

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:spring-boot-3-upgrade-recipes:1.2.
3 -Drewrite.activeRecipes=com.vmware.tanzu.spring.recipes.framework62.UpgradeSpringFra
mework_6_2
```

The commercial recipes that are part of the upgrade are described in Recipes for Spring Framework 6.2.x

Migrate from JAXRS to Spring Boot 3.3

Use the following Maven command:

```
./mvnw -B org.openrewrite.maven:rewrite-maven-plugin:5.45.1:runNoFork -Drewrite.recipe
ArtifactCoordinates=com.vmware.tanzu.spring.recipes:javaee-boot-recipes:1.2.3 -Drewrit
e.activeRecipes=com.vmware.tanzu.spring.recipes.javaee.jaxrs.MigrateJaxRs
```

Design Principles

Commercial Spring Recipes follow a couple of design principles that are different from the OSS Spring recipes to avoid duplicated changes and to avoid executing unnecessary recipes. These principles are:

- Recipes do not perform steps to upgrade previous steps. For instance, the recipe to upgrade to Spring Boot 3.1.x does not invoke the recipe to upgrade to Spring Boot 3.0.x. It assumes that the user knows that the repository uses Spring Boot 3.0.x.
- Recipes do not upgrade downstream projects. The Spring Framework recipes do not upgrade Spring Security. VMware recommends using Spring Application Advisor if you don't want to have to remember what combination of recipes need to be executed in your repository.

Spring Boot 3.0.x Recipes

ID	Description
`com.vmware.tanzu.spring.recipes.common.generated.boot30.Upgrade_3_0_AllManagedDependencies`	Upgrades all the managed dependencies associated to Spring Boot 3.0.x
`com.vmware.tanzu.spring.recipes.boot30.ActuatorEndpointExposure`	Enables all Actuator endpoints.
`com.vmware.tanzu.spring.recipes.boot30.AddFormerLoggingDateProperty`	Adds property to keep former date format in logback messages if a project has a logback dependency.
`com.vmware.tanzu.spring.recipes.boot30.MetricsMigration`	Replaces the `WebMvcMetricsFilter` and `MetricsRestTemplateCustomizer` classes with `ServerHttpObservationFilter` and `ObservationRestTemplateCustomizer` respectively.
`com.vmware.tanzu.spring.recipes.boot30.Saml2IdentityProviderToAssertingPartyYamlMigration`	Replaces `spring.security.saml2.relyingparty.registration.{id}.identityprovider` with `spring.security.saml2.relyingparty.registration.{id}.assertingparty`.
`com.vmware.tanzu.spring.recipes.boot30.UpdateOverriddenR2dbcVersions`	Replaces `r2dbc-bom.version` with actuals driver version.
`com.vmware.tanzu.spring.recipes.boot30.DeprecatedPropertiesSpringBoot_3_0`	Adds inline comment to all properties deprecated in Spring Boot 3.0.x.
`com.vmware.tanzu.spring.recipes.boot30.UpgradeSpringBoot_3_0`	Main recipe that upgrades applications to the latest Spring Boot 3.0.x release.

Spring Boot 3.1.x Recipes

ID	Description
`com.vmware.tanzu.spring.recipes.common.generated.boot31.Upgrade_3_1_AllManagedDependencies`	Upgrades all the managed dependencies associated to Spring Boot 3.1.x
`com.vmware.tanzu.spring.recipes.boot31.UpdateBootMavenPluginWhenMavenCompilerPropertiesAreNull`	Replaces `${maven.compiler.source}` and/or `${maven.compiler.target}` if their values are defined as either `${maven.compiler.release}` or `${java.version}`.
`com.vmware.tanzu.spring.recipes.boot31.UseBootMavenCompilerRelease`	The `spring-boot-starter-parent` now uses `maven.compiler.release` to configure the Java version instead of `maven.compiler.source` and `maven.compiler.target`. If you use these in your build, migrate to `maven.compiler.release`.
`com.vmware.tanzu.spring.recipes.boot31.AdaptGitCommitIdPluginForBoot_3_1`	Replaces property `git-commit-id-plugin.version` with `git-commit-id-maven-plugin.version`.
`com.vmware.tanzu.spring.recipes.boot31.IgnoreRegistrationFailureForBoot_3_1`	Adds `setIgnoreRegistrationFailure(true)` when `ServletRegistrationBean` and `FilterRegistrationBean` are registered.

ID	Description
com.vmware.tanzu.spring.recipes.boot31.DisableHealthGroupMembershipValidationForBoot_3_1	Disables Health Group Membership Validation for Spring Boot 3.1.x
com.vmware.tanzu.spring.recipes.boot31.UpgradeSpringBoot_3_1	Main recipe that upgrades applications to the latest Spring Boot 3.1.x.

Spring Boot 3.2.x Recipes

ID	Description
com.vmware.tanzu.spring.recipes.common.generated.boot32.Upgrade_3_2_AllManagedDependencies	Upgrades all the managed dependencies associated with Spring Boot 3.2.x
com.vmware.tanzu.spring.recipes.boot32.UpgradeSpringBoot_3_2	Main recipe that upgrades applications to the latest Spring Boot 3.2.x.

Spring Boot 3.3.x Recipes

ID	Description
com.vmware.tanzu.spring.recipes.common.generated.boot33.Upgrade_3_3_AllManagedDependencies	Upgrades all the managed dependencies associated with Spring Boot 3.3.x
com.vmware.tanzu.spring.recipes.boot33.UpgradeSpringBoot_3_3	Main recipe that upgrades applications to the latest Spring Boot 3.3.x.

Spring Boot 3.4.x Recipes

ID	Description
com.vmware.tanzu.spring.recipes.boot34.ConfigurationPropertiesValidation	Adds `@Valid` annotation to activate validation of nested properties to support the lack of validation of nested configuration properties.
com.vmware.tanzu.spring.recipes.boot34.OltpTracingConnectionDetailsUrl	Replaces `OtlpTracingConnectionDetails.getUrl()` method with the same call, but adding `org.springframework.boot.actuate.autoconfigure.tracing.otlp.TransportHTTP` as a parameter.
com.vmware.tanzu.spring.recipes.boot34.ResourceBannerApplicationVersion	Overwrites the `ResourceBanner.getApplicationVersion(Class)` implementation of `ResourceBanner` subclasses by resolving the package version.
com.vmware.tanzu.spring.recipes.boot34.DeprecatedPropertiesSpringBoot_3_4	Either renames or adds inline comments to all properties removed or deprecated in Spring Boot 3.4.
com.vmware.tanzu.spring.recipes.common.generated.boot34.Upgrade_3_4_AllManagedDependencies	Upgrades all managed dependencies associated with Spring Boot 3.4.x

ID	Description
`com.vmware.tanzu.spring.recipes.boot34.UpgradeSpringBoot_3_4`	Main recipe that upgrades applications to the latest Spring Boot 3.4.x.

Spring Data 3.0.x Recipes

ID	Description
`com.vmware.tanzu.spring.recipes.data30.AddCheckForNotExistingEntityByDelete`	Restores the Spring Data 2.7.x semantics by throwing an `org.springframework.dao.EmptyResultDataAccessException` exception if the user requests deletion of a non-existent entity.
`com.vmware.tanzu.spring.recipes.data30.ReplaceAuditingHandlerConstructors`	Adapts the arguments of the `org.springframework.data.auditing.AuditingHandler` and `org.springframework.data.auditing.IsNewAwareAuditingHandler` constructors.
`com.vmware.tanzu.spring.recipes.data30.ReplaceInstantiationAwarePropertyAccessorConstructor`	Adapts the arguments of the `org.springframework.data.mapping.model.InstantiationAwarePropertyAccessor` constructor.
`com.vmware.tanzu.spring.recipes.data30.ReplaceKotlinReflectionMethods`	Replaces deprecated Kotlin reflection methods in Spring Data Commons.
`com.vmware.tanzu.spring.recipes.data30.ReplaceLazyConstructor`	Replaces the `org.springframework.data.util.Lazy` constructor with `Lazy.of()`.
`com.vmware.tanzu.spring.recipes.data30.SortingRepoWithoutCrudRepo`	Adapts subtypes of `org.springframework.data.repository.PagingAndSortingRepository` to extend `org.springframework.data.repository.CrudRepository`
`com.vmware.tanzu.spring.recipes.data30.UpgradeSpringDataRedis_3_0`	Renames the methods, types and constants according the new APIs for Spring Data Redis 3.0.x.
`com.vmware.tanzu.spring.recipes.data30.UpgradeSpringData_3_0`	Main recipe that upgrades applications to the latest Spring Data 3.0.x.

Spring Framework 6.0.x Recipes

.

ID	Description
`com.vmware.tanzu.spring.recipes.framework60.AsyncAnnotationMethodReturnType`	Replaces methods to return `Future` or `void` for methods/types annotated with `@Async`.
`com.vmware.tanzu.spring.recipes.framework60.DeprecateSerializationUtilsForFramework`	`SerializationUtils.deserialize(...)` is deprecated due to vulnerabilities. Typical use of it is to clone an object via serialize/deserialize. Replaces this use with a `SerializationUtils.clone(obj)` call.

ID	Description
com.vmware.tanzu.spring.recipes.framework60.EnableFullBeanIntrospector	Enables java.beans.Introspector for Spring 5.x backward compatibility.
com.vmware.tanzu.spring.recipes.framework60.LegacySqlJdbcErrorCodesTranslator	Adds Legacy SQL JDBC error codes translator XML file if spring-jdbc is on the classpath.
com.vmware.tanzu.spring.recipes.framework60.MigrateAddCallBackFromListenableFuture	Replaces ListenableFuture.addCallBack(ListenableFutureCallback) with CompletableFuture.whenComplete(BiConsumer).
com.vmware.tanzu.spring.recipes.framework60.MigrateListenableFutureCallback	Replaces ListenableFutureCallback with BiConsumer.
com.vmware.tanzu.spring.recipes.framework60.RemoveRequiredAnnotation	Removes @Required from setters and adds corresponding property assertions in afterPropertySet() or @PostConstruct
com.vmware.tanzu.spring.recipes.framework60.ReplaceAsyncTaskExecutorCFPattern	Replaces supplyAsync(..) pattern with executor.submitCompletable(task) for AsyncListenableTaskExecutor case.
com.vmware.tanzu.spring.recipes.framework60.ReplaceCommonsMultipartFile	Replaces CommonsMultipartFile with MultipartFile, and constructor invocation with the generated MultipartFileDiskImpl.
com.vmware.tanzu.spring.recipes.framework60.ReplaceCommonsMultipartResolver	Replaces CommonsMultipartResolver with StandardServletMultipartResolver.
com.vmware.tanzu.spring.recipes.framework60.ReplaceIntStatusCodeWithHttpStatusCode	Replaces int status code with HttpStatusCode.valueOf(int) in method invocations.

ID	Description
com.vmware.tanzu.spring.recipes.framework60.ReplaceMergeAnnotationsEnclosingClassesStrategy	Replaces `MergeAnnotations.TYPE_HIERARCHY_AND_ENCLOSING_CLASSES`.
com.vmware.tanzu.spring.recipes.framework60.ReplaceRawStatusInClientHttpResponse	Replaces uses of `ClientHttpResponse#getRawStatusCode()`.
com.vmware.tanzu.spring.recipes.framework60.ReplaceRawStatusInClientResponse	Replaces uses of `ClientResponse#rawStatusCode()`.
com.vmware.tanzu.spring.recipes.framework60.ReplaceReturnedHttpStatus	Replaces `HttpStatus` with `HttpStatusCode` for method invocations.
com.vmware.tanzu.spring.recipes.framework60.HttpMethodAsClassForFramework_6_0	Replaces `EnumSet<>` with `Set` and replaces `switch` blocks with `if/else`.
com.vmware.tanzu.spring.recipes.framework60.HttpStatusCode_6_0	Migrates APIs from `HttpStatus` to `HttpStatusCode`.
com.vmware.tanzu.spring.recipes.framework60.RemoveOutdatedServletIntegrationsForFramework_6_0	Several outdated Servlet-based integrations have been dropped: e.g. Apache Commons FileUpload (`org.springframework.web.multipart.commons.CommonsMultipartResolver`), Apache Tiles, and FreeMarker JSP support in the corresponding `org.springframework.web.servlet.view` subpackages. VMware recommends `org.springframework.web.multipart.support.StandardServletMultipartResolver` for multipart file uploads and regular FreeMarker template views if needed, and a general focus on REST-oriented web architectures.
com.vmware.tanzu.spring.recipes.framework60.RenameTransactionExceptionClassesForFramework_6_0	Replaces `org.springframework.dao.CannotSerializeTransactionException` and `org.springframework.dao.DeadlockLoserDataAccessException` with `org.springframework.dao.PessimisticLockingFailureException`.
com.vmware.tanzu.spring.recipes.framework60.ReplaceListenableFutureForFramework_6_0	Replaces `ListenableFuture`.

ID	Description
com.vmware.tanzu.spring.recipes.framework60.RemoveRequiredAnnotationForFramework_6_0	Removes `@Required` from setter methods and adds corresponding property validation in `afterPropertySet()` or `@PostConstruct`.
com.vmware.tanzu.spring.recipes.framework60.UpgradeSpringFramework_6_0	Main recipe that upgrades applications to the latest Spring Framework 6.0.x.

Spring Framework 6.1.x Recipes

ID	Description
com.vmware.tanzu.spring.recipes.framework61.AutowireCapableBeanFactoryCreateBean	Replaces `AutowireCapableBeanFactory.createBean(Class,int,boolean)` by `AutowireCapableBeanFactory.createBean(Class)` if the values of the 2nd parameter is a 0 (`AUTOWIRE_NO`) or 3 (`AUTOWIRE_CONSTRUCTOR`).
com.vmware.tanzu.spring.recipes.framework61.AutowireCapableBeanFactoryCreateBean	Replaces `AutowireCapableBeanFactory.createBean(Class,int,boolean)` to `AutowireCapableBeanFactory.createBean(Class)` if the values of the 2nd parameter is a 0 (`AUTOWIRE_NO`) or 3 (`AUTOWIRE_CONSTRUCTOR`).
com.vmware.tanzu.spring.recipes.framework61.AutowireCapableBeanFactoryCreateBean	Replaces `AutowireCapableBeanFactory.createBean(Class,int,boolean)` to `AutowireCapableBeanFactory.createBean(Class)` if the values of the 2nd parameter is a 0 (`AUTOWIRE_NO`) or 3 (`AUTOWIRE_CONSTRUCTOR`).
com.vmware.tanzu.spring.recipes.framework61.CommentOverClientHttpRequestFactory	Comments over new `ClientHttpRequestFactory` instances explaining buffering is not available
com.vmware.tanzu.spring.recipes.framework61.DeprecationsInAssert	Adds an `String` message to the deprecated methods from `org.springframework.util.Assert` to use a non-deprecated API.
com.vmware.tanzu.spring.recipes.framework61.MaybeRemoveValidatedAnnotationOnController	Removes `@Validated` at the controller class level in controllers whose method parameters contain `@Constraint` annotations.
com.vmware.tanzu.spring.recipes.framework61.NoStreamingDefaultPartHttpMessageReader	Replaces `DefaultPartHttpMessageReader.setStreaming(boolean)` with `PartEventHttpMessageReader`.
com.vmware.tanzu.spring.recipes.framework61.RemoveSetThrowExceptionIfNoHandlerFoundSetToTrue	Sets the `throwExceptionIfNoHandlerFound` property of `DispatcherHandler` to `true` by default.
com.vmware.tanzu.spring.recipes.framework61.RenameTransactionSystemException	Replaces `TransactionSystemException` to `JpaSystemException` in catch blocks.
com.vmware.tanzu.spring.recipes.framework61.TransactionEventListenerUsage	Flags incorrect usage of `@TransactionalEventListener`.

ID	Description
com.vmware.tanzu.spring.recipes.framework61.WrapWithBufferingClientHttpRequestFactory	Wraps `ClientHttpRequestFactory` in `BufferingClientHttpRequestFactory`.
com.vmware.tanzu.spring.recipes.framework61.UpgradeSpringFramework_6_1	Main recipe that upgrades applications to the latest Spring Framework 6.1.x.

Spring Framework 6.2.x Recipes

ID	Description
com.vmware.tanzu.spring.recipes.framework62.Base64UtilsRemoved	Replaces `org.springframework.util.Base64Utils` with `java.util.Base64`.
com.vmware.tanzu.spring.recipes.framework62.BodySpecDeprecations	Replaces `org.springframework.test.web.reactive.server.WebTestClient.BodyContentSpec.jsonPath(String, Object...)` with `org.springframework.test.web.reactive.server.WebTestClient.BodyContentSpec.jsonPath(String)`.
com.vmware.tanzu.spring.recipes.framework62.ClientHttpResponseApiRemovals	Replaces `org.springframework.http.client.ClientHttpResponse.getRawStatusCode()` with `org.springframework.http.client.ClientHttpResponse.getStatusCode().value()`
com.vmware.tanzu.spring.recipes.framework62.HandlerResultApiRemovals	Replaces `org.springframework.web.reactive.HandlerResult` with the appropriate expressions using `getExceptionHandler` and `setExceptionHandler`.
com.vmware.tanzu.spring.recipes.framework62.HttpHeadersDeprecations	Replaces `org.springframework.http.HttpHeaders.writableHttpHeaders(HttpHeaders)` with `new HttpHeaders(HttpHeaders)`
com.vmware.tanzu.spring.recipes.framework62.HttpRequestValuesRemovedApi	Adapts the existing implementations of `org.springframework.web.service.invoker.HttpRequestValues`.
com.vmware.tanzu.spring.recipes.framework62.HttpServiceProxyFactoryApiRemovals	Adapts the existing implementations of `org.springframework.web.service.invoker.HttpServiceProxyFactory`
com.vmware.tanzu.spring.recipes.framework62.JsonPathExpectationsHelperDeprecations	Replaces `org.springframework.test.util.JsonPathExpectationsHelper(String, Object...)` with `org.springframework.test.util.JsonPathExpectationsHelper(String)`.
com.vmware.tanzu.spring.recipes.framework62.MethodArgumentNotValidExceptionApiRemovals	Replaces usage of `org.springframework.web.bind.MethodArgumentNotValidException` with the equivalent `BindErrorUtils` expressions.
com.vmware.tanzu.spring.recipes.framework62.MockMvcRequestDeprecation	Replaces `MockMvcRequestBuilders.request(String, URI)` with `MockMvcRequestBuilders.request(HttpMethod, URI)`.

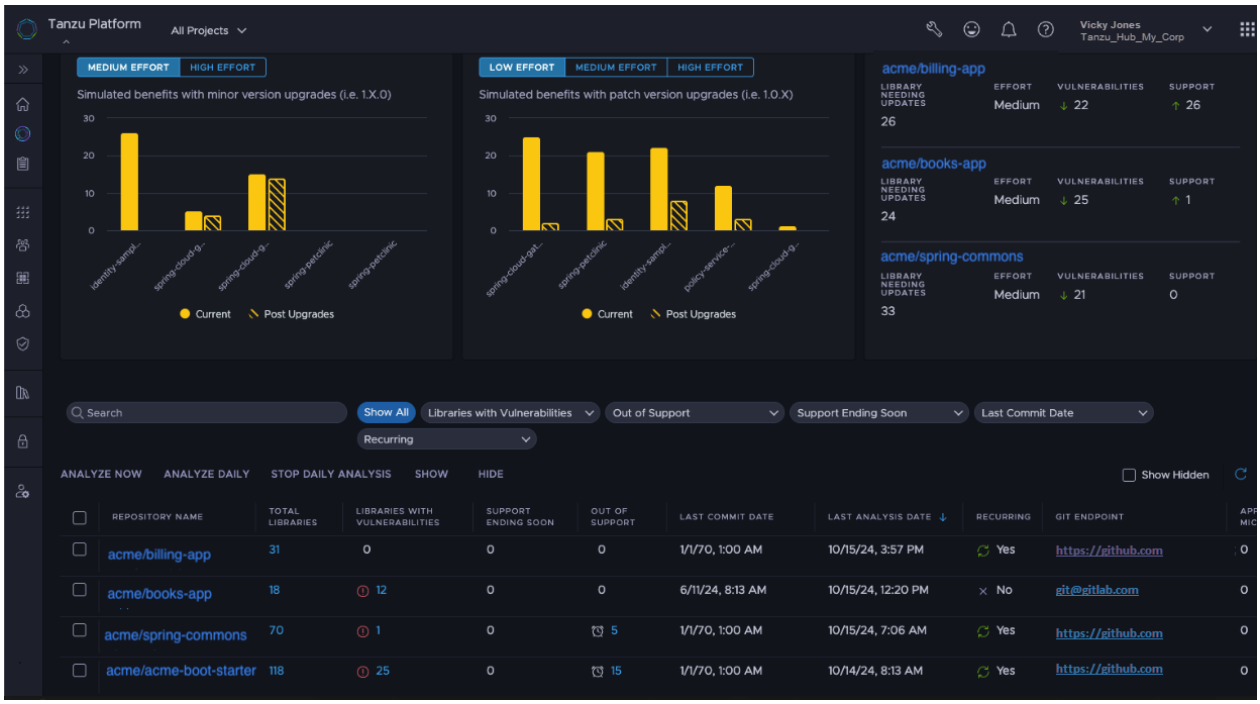
ID	Description
com.vmware.tanzu.spring.recipes.framework62.SingletonBeanRegistryDeprecations	Adds a TODO comment to replace the use of org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingletonMutex()
com.vmware.tanzu.spring.recipes.framework62.UriComponentsBuilderApiRemovals	Replaces org.springframework.web.util.UriComponentsBuilder.parseForwardedFor(HttpRequest, InetSocketAddress) with the same method with additional parameters.
com.vmware.tanzu.spring.recipes.framework62.WebSocketMessageBrokerStatsDeprecations	Replaces org.springframework.web.socket.config.WebSocketMessageBrokerStats with the equivalent String expressions with a previous null validation.
com.vmware.tanzu.spring.recipes.framework62.UriComponentsBuilderDeprecations	Replaces methods in org.springframework.web.util.UriComponentsBuilder with the corresponding ForwardedHeaderUtils.adaptFromForwardedHeaders expressions.
com.vmware.tanzu.spring.recipes.framework62.FreeMarkerViewApiRemovals	Replaces org.springframework.web.reactive.result.view.freemarker.getTemplate(Locale) with lookupTemplate(locale).block().
com.vmware.tanzu.spring.recipes.framework62.ResourceHttpMessageWriterApiRemovals	Replaces org.springframework.http.codec.ResourceHttpMessageWriter.addHeaders(ReactiveHttpOutputMessage, Resource, MediaType, Map) with addDefaultHeaders(...).block()
com.vmware.tanzu.spring.recipes.framework62.WebClientAdapterApiRemovals	Replaces org.springframework.web.reactive.function.client.support.WebClientAdapter.forClient(WebClient) with org.springframework.web.reactive.function.client.support.WebClientAdapter.create(WebClient).
com.vmware.tanzu.spring.recipes.framework62.ServerWebExchangeContextFilterApiRemovals	Replaces org.springframework.web.filter.reactive.ServerWebExchangeContextFilter.get(Context) with org.springframework.web.filter.reactive.ServerWebExchangeContextFilter.getExchange(getExchange).
com.vmware.tanzu.spring.recipes.framework62.HighCardinalityKeyNamesApiRemovals	Replaces HighCardinalityKeyNames.CLIENT_NAME with LowCardinalityKeyNames.CLIENT_NAME.
com.vmware.tanzu.spring.recipes.framework62.UpgradeSpringFramework_6_2	Main recipe that upgrades applications to the latest Spring Framework 6.2.x.

Spring Security 5.8.x Recipes

ID	Description
com.vmware.tanzu.spring.recipes.security58.UseNewRequestMatchers	Replaces the deprecated antMatchers, mvcMatchers, and regexMatchers methods for the new requestMatchers methods. Refer to the Spring Security docs for more information.
com.vmware.tanzu.spring.recipes.security58.UpgradeSpringSecurity_5_8	Main recipe that upgrades applications to the latest Spring Security 5.8.x.

Portfolio Analysis with the Tanzu Platform UI

Spring Application Advisor can be integrated with the Tanzu Platform UI to allow you to understand the support status and vulnerabilities of your Spring dependencies across all your Git repositories.



Connect the server to the Tanzu Platform UI

This section describes how to connect the Spring Application Advisor server component to Tanzu Platform and how to publish the dependencies of your repositories.

Using Tanzu Platform UI SaaS

If you use the Tanzu Platform UI from `https://platform.tanzu.broadcom.com`, before starting the server component of Spring Application Advisor, you must configure the following environment variables:

```
export TANZU_PLATFORM_INTEGRATION_ENABLED=true
export TANZU_PLATFORM_URL=https://data.platform.tanzu.broadcom.com
export TANZU_PLATFORM_CSP_URL=https://console.tanzu.broadcom.com
export TANZU_PLATFORM_ORG_ID=<YOUR_ORG_ID>
export TANZU_PLATFORM_APP_ID=<YOUR_APP_ID>
export TANZU_PLATFORM_APP_SECRET=<YOUR_APP_ID>
```

These are the functions of the environment variables:

- `TANZU_PLATFORM_INTEGRATION_ENABLED` enables (when set to `true`) sending data from the server component to Tanzu Platform.
- `TANZU_PLATFORM_URL` is the URL location of your Tanzu Platform data instance. In this case, the location of the public service must be `https://data.platform.tanzu.broadcom.com`.
- `TANZU_PLATFORM_CSP_URL` is the URL location to generate a temporal access token for each request. In this case, the location of the public service must be `https://console.tanzu.broadcom.com`.

- `TANZU_PLATFORM_ORG_ID` is the UUID of the organization registered in the Tanzu Platform to which your repositories belong; for example, `ee04bfae-a665-4f20-a5b9-d8b043180252`. To get the value for your organization, click the top right menu where you see your user name, and click the copy icon.
- `TANZU_PLATFORM_APP_ID` is the OAuth application identifier of the added OAuth application used to send the data.
- `TANZU_PLATFORM_APP_SECRET` is the OAuth application secret of the added OAuth application used to send the data.

TO learn how to create an OAuth application, see the Tanzu Platform documentation

Using Tanzu Platform UI Self-Managed

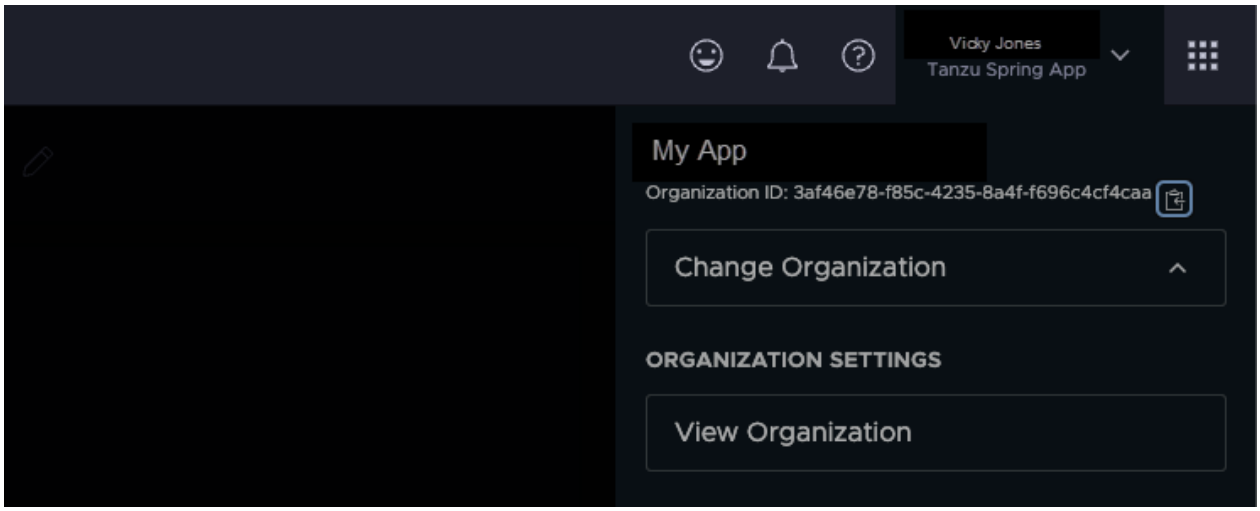
If you have your own installation of Tanzu Platform UI (self-managed), before starting the server component of Spring Application Advisor, you must configure the following environment variables:

```
export TANZU_PLATFORM_INTEGRATION_ENABLED=true
export TANZU_PLATFORM_URL=<YOUR_TANZU_HUB_HOST>
export TANZU_PLATFORM_ORG_ID=<YOUR_ORG_ID>
export TANZU_PLATFORM_API_KEY=<API_KEY>
```

These are the functions of the environment variables:

- `TANZU_PLATFORM_INTEGRATION_ENABLED` enables (when set to `true`) sending data from the server component to Tanzu Platform.
- `TANZU_PLATFORM_URL` is the URL location of your Tanzu Platform data instance. For example, the location of the public service is `https://data.platform.tanzu.broadcom.com`.
- `TANZU_PLATFORM_ORG_ID` is the UUID of the organization registered in the Tanzu Platform to which your repositories belong; for example, `ee04bfae-a665-4f20-a5b9-d8b043180252`. To get the value for your organization, click the top right menu where you see your user name, and click the copy icon.
- `TANZU_PLATFORM_API_KEY` is a key designed to connect third-party components to the Tanzu platform. This key must be generated by the administrator of the Tanzu Platform. See the Tanzu Platform documentation.

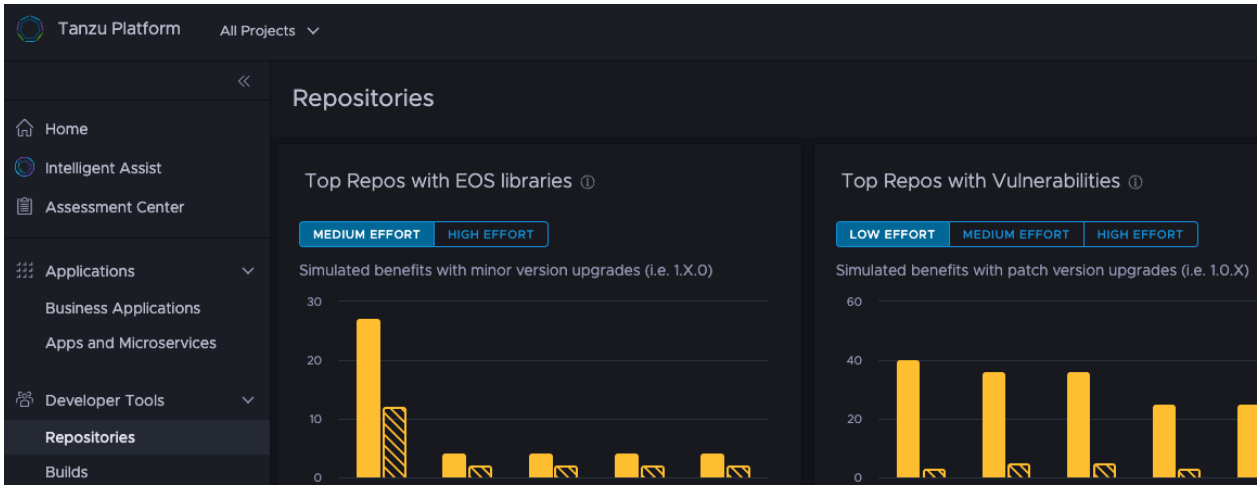
The following graphic shows how to copy the the organization ID from the Tanzu Platform UI.



After the server is configured, you can upload your Git repository build configuration. Use the following command from the local folder where the Git repository is located:

```
advisor build-config publish
```

After the command is executed, you can see the support status of your Spring dependencies and the associated vulnerabilities under **Developer Tools** > **Repositories**.



Troubleshooting Spring Application Advisor

This topic provides steps for troubleshooting common Spring Application Advisor problems.

Why does the apply command report that there are no upgrade plans if there are outdated Spring dependencies?

This is usually the case when your application is using components/libraries from other repositories that are using Spring. To find out what third-party components depend on Spring, and should be upgraded first, run:

```
advisor upgrade-plan get --url=http://localhost:9003
```

This will produce an output like this:

```
The projects ["spring-framework"] could not be included in the Upgrade Plan because th
ey are used as transitive dependencies for other projects, and no upgrades are configu
red for them.
Ask your administrator to configure the projects of the following dependencies:

        - org.flowable:flowable-engine-common uses: ["spring-framework"]
        - org.flowable:flowable-engine uses: ["spring-framework"]
        - org.flowable:flowable-event-registry-spring uses: ["spring-framework"]
        - org.flowable:flowable-spring-common uses: ["spring-framework"]
        - org.flowable:flowable-identitylink-service uses: ["spring-framework"]
        ...

No upgrade plans available - your project seems to be up to date.
```

If the components listed in the output belong to your repositories, you need to do the following:

1. Upgrade these components using Spring Application Advisor.
2. Create an OpenRewrite recipe to upgrade the components.
3. Define the mapping between the components and the recipes in Spring Application Advisor.

If the listed components do not belong to your repositories, contact Broadcom Software Support for help.

Why is my project unable to resolve the new Spring Maven Plugin?

Spring Application Advisor is resolving the latest patch version of Spring projects in the configured Maven repositories. If you have configured your Maven repositories to use Application Advisor, it will update the Spring Maven Plugin, and in this case, it might include a commercial version that is available only in the commercial repository.

To solve this problem, VMware recommends ensuring that `https://packages.broadcom.com/artifactory/spring-enterprise` is accessible as a `pluginRepository` in your Maven settings file; that is, in `$HOME/.m2/settings.xml`.

See Running commercial recipes using OpenRewrite tools for a detailed example.

Why is Spring Application Advisor unable to resolve the bom.json file?

If the command `advisor build-config get` is failing with the following message, this is usually because you have a conflicting configuration for the CycloneDX plug-in and the output directory for `bom.json` file is not in the default location.

```
java.io.UncheckedIOException: Could not read file: YOUR_REPO_DIR/build/reports/bom.jso
n
```

Spring Application Advisor expects to find the file in:

- For Maven, `target/classes/META-INF/sbom`
- For Gradle, `build/reports`

To resolve the problem, replace the output directory with the default value for your system, or move the file.

Why am I seeing the “Blocked mirror for repositories” error when applying the upgrade plan?

If there are errors in the Maven settings (that is, `$HOME/.m2/settings.xml`) used to download the Spring commercial recipes, the `advisor upgrade-plan apply` command fails.

If you are using mirror repositories and you see the following error in the generated log file, this indicates that there might be a rule defined in the global Maven settings file under `mirrorOf` that is blocking the download. Refer to the Maven documentation for information about adapting the patterns.

```
[ERROR] Failed to execute goal org.openrewrite.maven:rewrite-maven-plugin:5.35.0:runNo
Fork (default-cli) on project demo: Failed to resolve requested artifacts transitive d
ependencies. Failed to collect dependencies at com.vmware.tanzu.spring.recipes:spring-
2-7-upgrade-recipes:jar:0.0.1-M4: Failed to read artifact descriptor for com.vmware.ta
nzu.spring.recipes:spring-2-7-upgrade-recipes:jar:0.0.1-M4: Could not transfer artifac
t com.vmware.tanzu.spring.recipes:spring-2-7-upgrade-recipes:pom:0.0.1-M4 from/to mave
n-default-http-blocker (http://0.0.0.0/): Blocked mirror for repositories: [spring-ent
erprise (http://20.15.236.104:8081/repository/spring-enterprise/, default, releases+sn
apshots)]
```

The following is is an example showing how to combine a proxy for Maven central and another for the Spring Enterprise repository in a single Maven settings file:

```
<settings>
  <mirrors>
    <mirror>
      <id>internal-repository</id>
      <name>Maven Repository Manager running on repo.mycompany.com</name>
      <url>http://repo.mycompany.com/proxy</url>
      <mirrorOf>*,!spring-enterprise-subscription</mirrorOf>
    </mirror>
    <mirror>
      <id>mirror-spring-enterprise</id>
      <name>Mirror for Spring Enterprise</name>
      <url>http://repo.mycompany.com/spring-enterprise-repo</url>
      <mirrorOf>spring-enterprise-subscription</mirrorOf>
    </mirror>
  </mirrors>
  <activeProfiles>
      <activeProfile>org-profile</activeProfile>
  </activeProfiles>
  <profiles>
    <profile>
        <id>org-profile</id>
        <repositories>
          <repository>
            <id>spring-enterprise-subscription</id>
            <url>https://packages.broadcom.com/artifactory/spring-enterprise</url>
          </repository>
          <repository>
            <id>central</id>
            <url>http://central</url>
            <releases>
              <enabled>true</enabled>
            </releases>
            <snapshots>
```

```
              <enabled>true</enabled>
            </snapshots>
          </repository>
        </repositories>
    </profile>
  </profiles>
  <servers>
    <server>
      <id>mirror-spring-enterprise</id>
      <username>USERNAME</username>
      <password>PWD<password>
    </server>
    <server>
      <id>internal-repository</id>
      <username>USERNAME</username>
      <password>PWD<password>
    </server>
    <server>
      <id>spring-enterprise-subscription</id>
      <username>USERNAME</username>
      <password>PWD<password>
    </server>
  </servers>
</settings>
```

Why can’t I see my repository in the Tanzu Platform?

If you are not seeing your repository in the Tanzu Platform UI, check the logs of the Spring Application Advisor server component.

If you see an error like the one shown here, ensure that you have specified the correct value for the `TANZU_PLATFORM_URL` environment variable. This needs to be the URL location of your Tanzu Platform data instance (e.g., https://data.platform.tanzu.broadcom.com), not the the URL for Tanzu Platform UI.

```
org.springframework.web.client.HttpClientErrorException$BadRequest: 400 Bad Request: "
<?xml version='1.0' encoding='UTF-8'?><Error><Code>InvalidArgument</Code><Message>Inva
lid argument.</Message><Details>POST object expects Content-Type multipart/form-data</
Details></Error>"
```

Spring Application Advisor CLI Reference

This topic provides the list of CLI commands for Spring Application Advisor, along with supported options and examples.

advisor build-config get

Generates the project build configuration compile-time dependencies and developer tools versions to compile the Java sources of a repository.

Usage

```
advisor build-config get [-dh] [-b=<buildTool>] [-p=<path>]
```

Generates the project build configuration, including compile-time dependencies and developer tools versions to compile the Java sources of a repository.

The build-configuration file is named `.advisor/build-config.json`. This file is generated in the build directory of the project:

- For Maven: `/target` folder
- For Gradle: `/build` folder

The build-configuration file is required to generate the upgrade plan of the repository.

Supported options

Options	Function
-b, --build-tool=*buildTool*	Selects the build tool used to resolve the project dependencies options: `mvnw`, `mvn`, `gradlew`, `gradle` (default: `mvnw` when there are multiple wrappers, and `mvn` when there are no wrappers)
-d, --debug	Prints out debug messages
-h, --help	Prints the help for the command options
-p, --path=*path*	Selects the root directory of the source code repository (default: current directory)

Examples

Example	Result
`advisor build-config get`	Generates the upgrade plan of the repository when the root folder is in the current directory
`advisor build-config get --path=/home/user/foo`	Generates the upgrade plan of the repository when the root folder is in the `/home/user/foo` directory

advisor build-config publish

Publishes an existing build configuration of the source code repository into the Application Advisor server.

Usage

```
advisor build-config publish [-h] [-p=<path>] -u=<appAdvisorServerUrl>
```

Supported options

Options	Function
-h, --help	Prints the help for the command options
-p, --path=*path*	Selects the root directory of the source code repository (default: current directory)

Options	Function
-u, --url=*appAdvisorServerUrl*	Selects the URL location of the Application Advisor server (default: `$ADVISOR_SERVER`)

Examples

Example	Result
`advisor build-config publish --url=http://localhost:9003`	Publishes the build configuration associated with the repository located in the current directory

advisor upgrade-plan get

Prints out the upgrade plan of the source code repository. An upgrade plan is the list of incremental Spring related upgrades that can be performed in isolation.

Usage

```
advisor upgrade-plan get [-dfh] [-p=<path>] -u=<appAdvisorServerUrl>
```

Supported options

Options	Function
-d, --debug	Prints out debug messages
-h, --help	Prints the help for the command options
-f, --force	Forces the resolution of the upgrade plan excluding intermediate dependencies that belong to unknown projects that use Spring.
-p, --path=*path*	Selects the root directory of the source code repository (default: current directory)
-u, --url=*appAdvisorServerUrl*	Selects the URL location of the Application Advisor server (default: `$ADVISOR_SERVER`)

Examples

Example	Result
advisor upgrade-plan get --url=http://localhost:9003	Prints the upgrade plan associated to the repository located in the current directory.
advisor upgrade-plan get --force --url=http://localhost:9003	If there are unrecognized dependencies that use Spring, the upgrade plan is empty. The `--force` option resolves the upgrade plan by ignoring the unrecognized dependencies.

advisor upgrade-plan apply

Incrementally applies an upgrade plan to the source code repository. This command applies the first step of the upgrade plan to the source code repository. You must first generate the build configuration using `advisor build-config get`.

If you want to preview a list of the next versions to upgrade your dependencies to, use `advisor upgrade-plan get`.

Usage

```
advisor upgrade-plan apply [-dfh] [--from-yml] [--push]
                           [--after-upgrade-cmd=<afterUpgradeRunCommand>] [-b=<buildTo
ol>]
                           [--build-tool-jvm-args=<buildToolJvmArgs>] [-p=<path>]
                           -u=<appAdvisorServerUrl>
```

Supported options

Options	Function
--after-upgrade-cmd=*afterUpgradeRunCommand*	Executes a Maven or Gradle task after applying the upgrade plan
-b, --build-tool=*buildTool*	Selects the build tool used to compile the sources options: `mvnw`, `mvn`, `gradlew`, `gradle` (default: `mvnw` when there are multiple wrappers, and `mvn` when there are no wrappers)
--build-tool-jvm-args=*buildToolJvmArgs*	Adds JVM arguments to pass into the build tool (default: `$BUILD_TOOL_JVM_ARGS`)
-d, --debug	Prints out debug messages
-f, --force	Forces execution of full upgrade plan, including intermediate dependencies
--from-yml	Enables the upgrade plan based on the contents of the `.spring-app-advisor.yml` file in the selected path.
-h, --help	Prints the help for the command options.
-p, --path=*path*	Selects the root directory of the source code repository (default: current directory)
--push	Generates a pull request with the code upgrades. The environment variable `GIT_TOKEN_FOR_PRS` is mandatory. It should contain the value of a token with permissions for creating pull requests in the repository
--push	Generates a pull request with the code upgrades. The environment variable `GIT_TOKEN_FOR_PRS` is mandatory. It should contain the value of a token with permissions for creating pull requests in the repository
-u, --url=*appAdvisorServerUrl*	Selects the URL location of the Application Advisor server (default: `$ADVISOR_SERVER`)

Examples

Example	Result
advisor upgrade-plan apply	Upgrades the repository in the current directory

Example	Result
advisor upgrade-plan apply --push	Upgrades the repository and creates a pull request with the changes
advisor upgrade-plan apply --push --from-yml	Upgrades the repository and creates a pull request with the changes if developers have explicitly enabled automatic updates
advisor upgrade-plan apply --force	Upgrades the repository in the current directory, ignoring version upgrades in intermediate dependencies that use Spring. This can potentially break the build, but allows you to preview the Spring-related changes.

advisor mapping build (Experimental)

Generates an upgrade mapping file for a project given its repository.

Usage

```
advisor mapping build [-dho] [-b=<buildTool>] [--build-tool-options=<buildToolOptions
>] [-c=<coordinate>]
                      -r=<repositoryUrl> [-s=<slug>] [-t=<accessToken>] -u=<appAdvisor
ServerUrl>
```

Supported options

Options	Function
-b, --build-tool=*buildTool*	Selects the build tool used to compile the sources options: `mvnw`, `mvn`, `gradlew`, `gradle` (default: `mvnw` when there are multiple wrappers, and `mvn` when there are no wrappers)
--build-tool-options=*buildToolOptions*	Build arguments to pass to the build tool
-c, --coordinate=*coordinate*	Main coordinate of the project to check available versions in the format `groupId:artifactId`
-d, --debug	Prints out debug messages
-f, --force	Forces execution of full upgrade plan, including intermediate dependencies
-h, --help	Prints the help for the command options
-o, --offline	Resolves the versions offline, using the local Maven repository
-r, --repository-url=*repositoryUrl*	Selects the Git repository URL of the project
-s, --slug=*slug*	Name of the project to include into the mapping result
-t, --accessToken=*accessToken*	Personal Access Token for the git repository if needed
-u, --url=*appAdvisorServerUrl*	Selects the URL location of the Application Advisor server (default: `$ADVISOR_SERVER`)

Examples

Example	Result
advisor mapping build -r='https://github.com/spring-cloud/spring-cloud-cli'	Generates the upgrade mappings for spring-cloud-cli

advisor

Base syntax, requires a command.

Usage

```
Usage: advisor [-v] [?] [COMMAND]
```

Supported options

COMMAND	Explanation
build-config	Generates or publishes build dependencies and tools
upgrade-plan	Generates or applies upgrade plan(s) to upgrade the repository code base with the latest versions of Spring components
--version	Prints version of Spring Application Advisor CLI

Enterprise Spring Boot Governance Starter

The Enterprise Spring Boot Governance Starter library enforces cipher and TLS security based on the industry standard, and empowers Spring developers to auto-generate compliance and governance reporting information for their applications.

Spring Boot Governance Starter Release Notes

These are the release notes for Enterprise Spring Boot Governance Starter.

v1.3.0

Release Date: September 12, 2024

- BasicAuthenticationFilter now satisfies TNZSPEC-0013
- Set status of expired CMVP certificates to “Historical”

v1.2.0

Release Date: August 7, 2024

- Upgrades Bouncy Castle FIPS Java API dependencies to version 2.0

The CMVP certificate for the 1.0.x line of the Bouncy Castle FIPS Java API (4616), used in previous versions of the Enterprise Spring Boot Governance Starter, has an expiration date of August 22nd, 2024. We strongly recommend upgrading to this version to ensure you remain FIPS compliant.

v1.1.0

Release Date: July 10, 2024

- Add support for Spring Boot 3.3.0
- Adds the following tags: PCIv4, DHS-4300B, CNSSI-1253J
- Add TNZSPEC-0013 verify the presence of an authenticaiton filter
- Add TNZSPEC-0014 to verify session cookies are secure
- Add TNZSPEC-0015 to validate the max number of sessions
- Add TNZSPEC-0016 to validate that passwords are not being encoded in Plain-text
- Add TNZSPEC-0017 to check if input validation/sanitization is available
- Add TNZSPEC-0018 to check for session timeout
- Add TNZSPEC-0019 to check if the CSRF filter is present
- Add TNZSPEC-0020 to check that the auditing actuator is activated

v1.0.0

Release Date: May 23, 2024

This is the first release.

Overview

The Enterprise Spring Boot Governance Starter library enforces cipher and TLS security based on the industry standard, and empowers Spring developers to auto-generate compliance and governance reporting information for their applications. This is done using the leading FIPS-approved security provider BouncyCastle and auto-configurations to enforce their compliance stance. In addition, the library conducts analysis at application startup, and provides a rich set of predefined regulatory compliance specification tests, out of the box. The results are accessible through a custom Spring Boot actuator endpoint in JSON format. This report can be easily consumed by clients to aggregate findings and generate targeted presentations.

This library also provides support for adding custom governance specifications, so you can extend its functionality beyond predefined compliance standards rules. After your custom governance specifications are added, the library runs these validations against your application at startup. This feature allows you to incorporate organization-specific regulations, industry guidelines, or proprietary standards into your validation processes.

Minimum Requirements

Component	Version
Java Virtual Machine	17+
Spring Boot	2.7.x+

Predefined Validations

Server TLS Validation

	Protocols	Ciphers	RSA/EC Key Size
Tomcat	Yes	Yes	Yes
Netty	Yes	Yes	Yes

Client TLS Validation

	Hostname Verification	Encryption Validation
HTTPS *		
RestTemplateBeans**	Yes	Yes
RestClientBeans**	Yes	Yes
Declarative RestClientBeans**	Yes	Yes
WebClientBeans	Yes	Yes
Declarative WebClientBeans	Yes	Yes
JDBC		
MySQL	Yes	Yes
MariaDB	Yes	Yes
PostgreSQL	Yes	Yes
Authentication/Authorization		
LDAP AuthenticationManager beans	Yes	Yes

- *For more details about HTTPS, see https://docs.spring.io/spring-framework/reference/integration/rest-clients.html.
- **Supports the following HTTP request factories: SimpleHttpClient, Apache, JDK

OIDC Clients

Validates the usage of TLS on the OIDC Endpoints for the supplied providers

Endpoint	TLS
Issuer	Yes
Authorization	Yes
Token	Yes
User Info	Yes
JWKS	Yes

Getting Started

This topic provides the prerequisites and instructions for running Enterprise Spring Boot Governance Starter library with your app for the first time.

Prerequisites

- Before adding the Spring Boot Governance Starter to your application, you must have access to it in your project's build. The recommended approach is to sync the Spring Enterprise Subscription artifact repository to your internal artifact repository. See Spring Enterprise Subscription for Artifact Repository Administrators for details.
- Your Maven or Gradle build environment must have access to an artifact repository where the Spring Boot Governance Starter library is available, at group path `com.vmware.tanzu.spring.governance`.

Configure the Dependency

The instructions in the following sections describe how to add the Spring Boot Governance Starter dependency to your project's Gradle or Maven build file.

Gradle

In your `build.gradle` file, add the governance-starter dependency:

```
dependencies {
    ...
    implementation "com.vmware.tanzu.spring.governance:governance-starter:1.0.0"
}
```

Maven

In your `pom.xml` file, add the governance-starter dependency:

```
...
<dependency>
    <groupId>com.vmware.tanzu.spring.governance</groupId>
    <artifactId>governance-starter</artifactId>
    <version>1.0.0</version>
</dependency>
```

Run the Application

If the app does not have the necessary TLS setting in place, the app will fail to start with the following error (or similar):

```
***************************
APPLICATION FAILED TO START
***************************

Description:

FIPS validation test case(s) encountered 1 failure(s):
```

```
- Failed test for TNZSPEC-0001: Application must use TLS
  - Expected TLS to be enabled for connector at port 8080


Action:

Resolve all test failures
```

By default, the app exits on a FIPS validation error, following FIPS rules. However, you can turn that off by setting a property to allow continuous improvement in test environment:

```
tanzu.governance:
  fips:
    exit-on-failure: false
```

By default, all specs will be checked. However, you can skip specific specs by providing them as a comma-separated list to the skip property:

```
tanzu.governance:
  specs:
    skip: TNZSPEC-0001,TNZSPEC-0020
```

The `governance-starter` brings in a couple of FIPS-related Bouncy Castle libraries as transitive dependencies. By default, BouncyCastle is configured in FIPS mode and is set as the primary security provider in the application context. You can override this behavior by setting the `enforce` property:

```
tanzu.governance:
  fips:
    config:
      bouncy-castle:
        enforce: false
```

Enable TLS with a PKCS12 keystore (non-compliant)

1. Generate a certificate with keytool:

   ```
   keytool -genkeypair -alias "demo" -keyalg RSA -keysize 4096 -validity 3650 \
   -dname "CN=localhost" -keypass changeit \
   -keystore src/main/resources/keystore.p12 \
   -ext "SAN=dns:localhost,dns:hello-fips,dns:hello-fips.sample" \
   -storeType PKCS12 -storepass changeit
   ```

2. Add properties to `application.yml` to configure TLS:

   ```
   server:
       port: 8443
       ssl:
           enabled: true
           key-alias: demo
           key-store: classpath:keystore.p12
           key-store-password: changeit
           key-store-type: PKCS12
   ```

3. In this case, the app would fail to start because this key store type is not available in an approved mode of operation due to the algorithms required for PBE key generation in the PKCS12 standard. See section 7 in the Bouncy Castle documentation.

```
Caused by: java.security.NoSuchAlgorithmException: Cannot find any provider sup
porting PBES2
    at java.base/javax.crypto.Cipher.getInstance(Cipher.java:574) ~[na:na]
    at java.base/sun.security.pkcs12.PKCS12KeyStore.lambda$engineGetKey$0(PKCS1
2KeyStore.java:363) ~[na:na]
    at java.base/sun.security.pkcs12.PKCS12KeyStore$RetryWithZero.run(PKCS12Key
Store.java:257) ~[na:na]
    at java.base/sun.security.pkcs12.PKCS12KeyStore.engineGetKey(PKCS12KeyStor
e.java:361) ~[na:na]
    ... 30 common frames omitted
```

Enable TLS with a BCFKS certificate (compliant)

1. Generate a certificate with `keytool`:

```
export JAR_FILE="bc-fips-1.0.2.4.jar"
export PROVIDER="org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider"

# Download the bouncy castle jar to create the cert with approved encryption me
thod
curl -s -o "$JAR_FILE" https://downloads.bouncycastle.org/fips-java/bc-fips-1.
0.2.4.jar

cd
keytool -genkeypair -alias demo -keyalg RSA -keysize 4096 -validity 3650 \
-dname "CN=localhost" -keypass changeit \
-keystore src/main/resources/keystore.bks \
-ext "SAN=dns:localhost,dns:hello-fips,dns:hello-fips.samples" \
-storeType BCFKS -storepass changeit \
-providerPath "$JAR_FILE" \
-provider "$PROVIDER" \
-providerClass "$PROVIDER"
```

2. Add the properties to `application.yaml` to configure TLS:

```
server:
    port: 8443
    ssl:
        enabled: true
        key-alias: demo
        key-store: classpath:keystore.bks
        key-store-password: changeit
        key-password: changeit
        key-store-type: "BCFKS"
```

The app now starts up correctly.

The Governance Actuator Endpoint

The Governance Starter library adds an actuator endpoint containing a summary of the test results, the collected details, and the individual test runs. The tests are re-run on every call to the endpoint.

The endpoint can be accessed at the `governance` actuator endpoint, which returns the tests and specs for all tags. See this example: `https://localhost:8443/actuator/governance`.

The Governance Starter library brings in `spring-boot-actuator` as a transitive dependency, which causes the default actuator endpoints to be made available. For more details about the Spring Boot Actuator, including what endpoints are enabled by default and how to turn them off, see the Spring documentation.

Filter by tag

Specs have tags associated with them to allow for filtering the spec. To filter the results by a specific tag, add it as a query parameter as shown in this example: `https://localhost:8443/actuator/governance?tag=FIPS-140-3`

The following tags are available in the predefined specifications:

Tag	Description
`FIPS-140-3`	Specs related to the Federal Information Processing Standard (FIPS) Publication 140-3.
`NIST.SP.800-52r2`	Specs related to NIST.SP.800-52r2, Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations.
`NIST.SP.800-131Ar2`	Specs related to NIST Special Publication 800-131A Rev. 2, Transitioning the Use of Cryptographic Algorithms and Key Lengths
`PCIv4`	Specs related PCI Data Security Standard (PCI DSS) v4.0.
`DHS-4300B`	Specs related DHS National Security Systems: Control Guidance Instruction Number: 4300B.102
`CNSSI-1253J`	Specs related Committee on National Security Systems Instruction (CNSSI) 1253, Security Categorization and Control Selection for National Security Systems
`IRS p1075r11-2021`	Specs related IRS Publication 1075 (Rev. 11-2021): Tax Information Security GuidelinesFor Federal, State and Local Agencies
`BouncyCastle`	Specs related to the BouncyCastle configuration in the application.

If there is a space in the tag name replace it with a `%20` when using it as a query parameter;
for example: https://`localhost:8443/actuator/governance?tag=IRS%20p1075r11-2021`.

Exposing the Endpoint

Add application properties to expose the Governance Actuator Endpoint:

```
management:
  endpoints:
    web:
```

```
      exposure:
        include: "governance"
```

Viewing the Governance Actuator Endpoint

Start the application again and access `https://localhost:8443/actuator/governance?tag=FIPS-140-3`. You should see the comprehensive report containing the FIPS stance, relevant configuration details, all the test rule runs, and an overall test summary at the bottom:

```
{
 "details": ...,
 "tests": ...,
  "timestamp": "2024-03-26T14:19:43Z",
  "results": {
    "totalTestCasesRan": 16,
    "totalTestCasesPassed": 16,
    "totalTestCasesFailed": 0,
    "totalTestCasesUnknown": 0,
    "totalTestCasesSkipped": 0,
    "allTestCasesPassed": true
  }
}
```

The field `totalTestCasesRan` is the sum of the tests that passed, failed, and resulted in an unknown state. The unknown state can occur when the library encountered a problem running a test, or collecting the data for the test.

For example, the library uses reflection while collecting the configuration of the application. In the case where reflection fails, the test result is unknown. If you encounter this result when running the pre-defined tests, contact Broadcom Support with your use case.

If there are failing or unknown tests, the library forces the application to shut down. However, you can bypass the shutdown by deactivating the specific test or by deactivating the `exit-on-failure` flag.

See Library Configuration Options for further instructions.

Library Configuration Options

The Enterprise Spring Boot Governance Starter library can be configured with the following properties.

Property	Description
tanzu.governance.test-mode	Null by default. Possible values: `[once, per_request]`. When unset or set to `once`, the library runs the tests once at startup and returns the same result going forward. When set to `per_request`, the library runs tests at startup and at every call to the `/actuator/governance` endpoint.
tanzu.governance.fips.exit-on-failure	True by default. When `true`, the application runs validation tests at startup and shutdown if any tests fail; otherwise it prints a warning message and continues.
tanzu.governance.fips.config.bouncy-castle.enforce	True by default. When `true`, BouncyCastle is configured in FIPS mode and is set as the primary security provider in application context.

Property	Description
tanzu.governance.fips.config.bouncy-castle.provider-config	Null by default. Set to override the BouncyCastle FIPS provider's configuration. See the Bouncy Castle documentation, Section 2.3: Provider configuration, for accepted values.
tanzu.governance.fips.config.server-tls.enforce	True by default. When `true`, the application configures the web server with FIPS default ciphers and protocols. Supports Tomcat and Netty.
tanzu.governance.specs.skip	Empty by default. Comma-separated list of Spec IDs to skip when performing validations.

Note that even if `tanzu.governance.fips.exit-on-failure` is set to `false`, your application may still fail to start if a rule is enforced by default, but related configuration or beans are not available.

For example, to enforce server TLS, the application must have SSL configured in its application properties:

```
server:
  ssl:
    enabled: true
    bundle: my-ssl-bundle
    // Or prior to SpringBoot 3.1 without the SSL bundle:
    key-alias: my-key
    key-store: classpath:certs/my-key-store.bks
    key-store-password: changeit
    key-password: changeit
    key-store-type: "BCFKS"
```

An application without proper SSL configurations results in server startup error with `tanzu.governance.fips.exit-on-failure=false` because the enforcer cannot find available SSL settings in the application context. To bypass this error, you can set `tanzu.governance.fips.config.server-tls.enforce` to `false`.

Governance Specifications

This topic describes the Governance Specifications.

See also Preconfigured Specifications.

A governance specification is composed of the fields shown in the following tables:

Name	Description
id	A unique id for the spec, e.g., "TNZSPEC-0001"
org	Represents the organizational structure or identifier associated with the spec. It typically follows the format of a Java package name; e.g., "com.vmware.tanzu"
title	A title for the specification
description	A detailed description of the specification
reference (1)	An optional reference to the standard being referenced. See the structure in the next table (1).
tags	A list of tags associated with the spec. The tag can be used to filter the tests and specs in the governance actuator endpoint. Must not include spaces.

(1) The reference is made up of the following fields:

Name	Description
standard	Optional. The name of the standard; e.g., “NIST SP 800-52 Rev. 2”
description	A description of the standard or the section to reference; e.g., “Section 3: Minimum Requirements for TLS Servers.”
url	Optional. The URL to reference; e.g., https://doi.org/10.6028/NIST.SP.800-52r2

Preconfigured Governance Specifications

This topic describes the pre-configured Governance specifications in the `governance-starter` library.

See also Governance Specifications.

The governance-starter library is pre-configured with the specifications shown in the following tables.

TNZSPEC-0001

id	TNZSPEC-0001	
org	com.vmware.tanzu	
title	Application must use TLS	
description	Any network service that handles sensitive or valuable data, whether it is personally identifiable information (PII), financial data, or login information, needs to adequately protect that data.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 1: Introduction"
	url	https://doi.org/10.6028/NIST.SP.800-52r2
tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0002

id	TNZSPEC-0002
org	com.vmware.tanzu
title	Application must support TLS 1.2 and TLS 1.3
description	Agencies shall support TLS 1.3 by January 1, 2024. After this date, servers shall support TLS 1.3 for both government-only and citizen or business-facing applications. In general, servers that support TLS 1.3 should be configured to use TLS 1.2 as well.

reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 3: Minimum Requirements for TLS Servers."
	url	https://doi.org/10.6028/NIST.SP.800-52r2
tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0003

id	TNZSPEC-0003	
org	com.vmware.tanzu	
title	TLS must be configured with NIST-approved cipher suites	
description	The server shall be configured to only use cipher suites that are composed entirely of NIST approved algorithms.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 3.3.1: Cipher Suites."
	url	https://doi.org/10.6028/NIST.SP.800-52r2
tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0004

id	TNZSPEC-0004	
org	com.vmware.tanzu	
title	When using RSA, the key size should be 2048 bits or greater	
description	The server shall be configured to only use cipher suites that are composed entirely of NIST approved algorithms.	
reference	**standard**	NIST SP 800-131A Rev. 2
	description	"Section 3: Digital Signatures."
	url	https://doi.org/10.6028/NIST.SP.800-131Ar2

tags	• FIPS-140-3 • NIST.SP.800-131Ar2 • PCIv4 • DHS-4300B • CNSSI-1253J

TNZSPEC-0005

id	TNZSPEC-0005	
org	com.vmware.tanzu	
title	When using ECDSA and EdDSA, the key size should be 224 bits or greater	
description	The security strength provided by an elliptic-curve-based signature algorithm is no greater than 1/2 of the length of the domain parameter n. Therefore, the length of n shall be at least 224 bits to meet the minimum security-strength requirement of 112 bits for Federal Government use.	
reference	**standard**	NIST SP 800-131A Rev. 2
	description	"Section 3: Digital Signatures."
	url	https://doi.org/10.6028/NIST.SP.800-131Ar2
tags	• FIPS-140-3 • NIST.SP.800-131Ar2 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0006

id	TNZSPEC-0006	
org	com.vmware.tanzu	
title	HTTPS TLS Clients must use hostname verification	
description	Hostname verification is essential for ensuring that the server's certificate matches the domain name of the server to prevent Man-in-the-Middle (MitM) attacks and protect against spoofing and other security threats.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 3.4.1.2 Server Name Indication"
	url	https://doi.org/10.6028/NIST.SP.800-52r2

tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J

TNZSPEC-0007

id	TNZSPEC-0007	
org	com.vmware.tanzu	
title	HTTPS TLS Clients must use certificate validation	
description	Certificate validation ensures that the presented certificate is authentic and issued by a trusted authority, maintaining the security and integrity of communication channels.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 4.5 Server Authentication"
	url	https://doi.org/10.6028/NIST.SP.800-52r2
tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0008

id	TNZSPEC-0008	
org	com.vmware.tanzu	
title	JDBC TLS Clients must use hostname verification	
description	Hostname verification is essential for ensuring that the server's certificate matches the domain name of the server to prevent Man-in-the-Middle (MitM) attacks and protect against spoofing and other security threats.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 3.4.1.2 Server Name Indication"
	url	https://doi.org/10.6028/NIST.SP.800-52r2

tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J

TNZSPEC-0009

id	TNZSPEC-0009	
org	com.vmware.tanzu	
title	JDBC TLS Clients must use hostname verification	
description	Certificate validation ensures that the presented certificate is authentic and issued by a trusted authority, maintaining the security and integrity of communication channels.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 4.5 Server Authentication"
	url	https://doi.org/10.6028/NIST.SP.800-52r2
tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0010

id	TNZSPEC-0010	
org	com.vmware.tanzu	
title	LDAP TLS Clients must use hostname verification	
description	Hostname verification is essential for ensuring that the server's certificate matches the domain name of the server to prevent Man-in-the-Middle (MitM) attacks and protect against spoofing and other security threats.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 3.4.1.2 Server Name Indication"
	url	https://doi.org/10.6028/NIST.SP.800-52r2

tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J

TNZSPEC-0011

id	TNZSPEC-0011	
org	com.vmware.tanzu	
title	LDAP TLS Clients must use certificate validation	
description	Certificate validation ensures that the presented certificate is authentic and issued by a trusted authority, maintaining the security and integrity of communication channels.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 4.5 Server Authentication"
	url	https://doi.org/10.6028/NIST.SP.800-52r2
tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0012

id	TNZSPEC-0012	
org	com.vmware.tanzu	
title	Application must use TLS when connecting to OIDC Provider	
description	Any network service that handles sensitive or valuable data, whether it is personally identifiable information (PII), financial data, or login information, needs to adequately protect that data.	
reference	**standard**	NIST SP 800-52 Rev. 2
	description	"Section 1: Introduction"
	url	https://doi.org/10.6028/NIST.SP.800-52r2

tags	• FIPS-140-3 • NIST.SP.800-52r2 • PCIv4 • DHS-4300B • CNSSI-1253J

TNZSPEC-0013

id	TNZSPEC-0013	
org	com.vmware.tanzu	
title	Uniquely Identify and Authenticate Users	
description	Ensure that both organizational and non-organizational users are uniquely identified and authenticated.	
reference	**standard**	NIST SP 800-53 Rev. 5
	description	"3.7 Identification and Authentication"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0014

id	TNZSPEC-0014	
org	com.vmware.tanzu	
title	Implement cryptographic mechanisms to protect the confidentiality and integrity of remote access sessions.	
description	Encrypting remote sessions is necessary to ensure the confidentiality and integrity of data transmitted between a client and a remote server. It prevents unauthorized access and eavesdropping by encrypting the communication, making it unreadable to anyone without the decryption key.	
reference	**standard**	NIST SP 800-53 Rev. 5
	description	"Security and Privacy Controls for Information Systems and Organizations, AC-17 (2) Protection of confidentiality and integrity using encryption"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0015

id	TNZSPEC-0015	
org	com.vmware.tanzu	
title	Concurrent Session Control	
description	Limit the number of concurrent sessions to ensure only a certain number of users can access the system at any given time, preventing overload and ensuring optimal performance and security.	
reference	**standard**	NIST SP 800-53 Rev. 5
	description	"Security and Privacy Controls for Information Systems and Organizations, AC-10 Concurrent Session Control"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0016

id	TNZSPEC-0016	
org	com.vmware.tanzu	
title	Verify that no unencrypted static authenticators are embedded in applications	
description	Plain-text passwords (i.e. unencrypted static authenticators) cannot be included in applications	
reference	**standard**	NIST SP 800-53 Rev. 5
	description	"Security and Privacy Controls for Information Systems and Organizations, IA-5 (7) Ensure that unencrypted static authenticators are not embedded in applications or other forms of static storage"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0017

id	TNZSPEC-0017
org	com.vmware.tanzu
title	Verify input validation/sanitization functionality is available in application

description	Validating inputs refers to the process of checking and verifying the accuracy, completeness, and integrity of data entered into a system. It ensures that the input meets specified criteria, preventing errors, security breaches, and system malfunctions caused by incorrect or malicious data.	
reference	**standard**	NIST SP 800-53 Rev. 5
	description	"Security and Privacy Controls for Information Systems and Organizations, SI-10 Information Input Validation"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • PCIv4 • DHS-4300B • CNSSI-1253J • IRS p1075r11-2021	

TNZSPEC-0018

id	TNZSPEC-0018	
org	com.vmware.tanzu	
title	Access Control - Session Termination after a period of inactivity	
description	Automatically terminate a user session after a period of inactivity.	
reference	**standard**	NIST SP 800-53 Rev. 5
	description	"Security and Privacy Controls for Information Systems and Organizations, AC012 Session Termination"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0019

id	TNZSPEC-0019
org	com.vmware.tanzu
title	Verify replay resistant techniques are employed in the application (CSRF)
description	Replay-resistant techniques for authenticators are necessary to prevent unauthorized access or impersonation. These techniques ensure that authentication credentials cannot be intercepted and replayed by attackers, thereby enhancing the security of the authentication process.

reference	**standard**	NIST SP 800-53 Rev. 5
	description	"Security and Privacy Controls for Information Systems and Organizations, IA-2 (8) Identification and Authentication (Organizational Users) | Access to Accounts — Replay Resistant"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • PCIv4 • DHS-4300B • CNSSI-1253J	

TNZSPEC-0020

id	TNZSPEC-0020	
org	com.vmware.tanzu	
title	Verify audit events are available in application	
description	An event is an observable occurrence in a system. The types of events that require logging are those events that are significant and relevant to the security of systems and the privacy of individuals.	
reference	**standard**	NIST SP 800-53 Rev. 5
	description	"Audit and Accountability, AU-2 Event Logging"
	url	https://doi.org/10.6028/NIST.SP.800-53r5
tags	• NIST-800-53 • PCIv4 • DHS-4300B • CNSSI-1253J • IRS p1075r11-2021	

TNZSPEC-0100

id	TNZSPEC-0100	
org	com.vmware.tanzu	
title	When using Bouncy Castle FIPS Java API, the BCFIPS provider should be the first security provider	
description	To ensure the BouncyCastleFipsProvider is used as the default provider for cryptographic algorithms, it should be the first security provider.	
reference	**description**	"BC-FJA 1.0.2 (Bouncy Castle FIPS Java API) User Guide"
	url	https://downloads.bouncycastle.org/fips-java/docs/BC-FJA-UserGuide-1.0.2.pdf

tags	• FIPS-140-3 • BouncyCastle

TNZSPEC-0101

id	TNZSPEC-0101	
org	com.vmware.tanzu	
title	When using Bouncy Castle FIPS Java API, the library's SHA-256 should match the checksum provided by the vendor	
description	To ensure the Bouncy Castle FIPS module has not been tampered or corrupted during transit, the checksum of the dependency should be verified against the checksum provided by the vendor.	
reference	**description**	"Bouncy Castle JAR Checksums"
	url	https://www.bouncycastle.org/fips-java/
tags	• FIPS-140-3 • BouncyCastle	

TNZSPEC-0102

id	TNZSPEC-0102	
org	com.vmware.tanzu	
title	When using the Bouncy Castle FIPS Java API, the latest version should be used	
description	The latest release contains fixes for defects found in prior versions.	
reference	**description**	"BC Java FIPS Release Notes"
	url	https://www.bouncycastle.org/fips-java/RELEASE_NOTES.md
tags	• FIPS-140-3 • BouncyCastle	

TNZSPEC-0103

id	TNZSPEC-0103
org	com.vmware.tanzu
title	When using the Bouncy Castle FIPS Java API, the module must be operated in FIPS mode
description	The Bouncy Castle FIPS Java API must be installed, initialized and configured as specified in the Security Policy Section 8 and operated in FIPS mode.

reference	**standard**	"FIPS 140-2"
	description	"BC-FJA (Bouncy Castle FIPS Java API), Certificate 4616, Security Policy"
	url	https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp4616.pdf
tags	• FIPS-140-3 • BouncyCastle	

TNZSPEC-0104

id	TNZSPEC-0104	
org	com.vmware.tanzu	
title	When using Bouncy Castle Java (D)TLS API and JSSE Provider, the BCJSSE provider should be the second security provider	
description	The Bouncy Castle BCJSSE provider should be the second security provider after BCFIPS.	
reference	**description**	"Java (D)TLS API and JSSE Provider User Guide Version 1.0.18, Section 2.1.1 Configuring the BCJSSE Provider in FIPS mode"
	url	https://downloads.bouncycastle.org/fips-java/docs/BC-FJA-%28D%29TLSUserGuide-1.0.18.pdf
tags	• FIPS-140-3 • BouncyCastle	

TNZSPEC-0105

id	TNZSPEC-0105	
org	com.vmware.tanzu	
title	When using Bouncy Castle Java (D)TLS API and JSSE Provider, the library's SHA-256 should match the checksum provided by the vendor	
description	To ensure the Bouncy Castle Java (D)TLS API and JSSE Provider library has not been tampered or corrupted during transit, the checksum of the dependency should be verified against the checksum provided by the vendor.	
reference	**description**	"Bouncy Castle JAR Checksums"
	url	https://www.bouncycastle.org/fips-java/
tags	• FIPS-140-3 • BouncyCastle	

TNZSPEC-0106

id	TNZSPEC-0106

org	com.vmware.tanzu	
title	When using the Bouncy Castle Java (D)TLS API and JSSE Provider, the latest version should be used	
description	The latest release contains fixes for defects found in prior versions.	
reference	**description**	"BC Java FIPS Release Notes"
	url	https://www.bouncycastle.org/fips-java/RELEASE_NOTES.md
tags	• FIPS-140-3 • BouncyCastle	

TNZSPEC-0107

id	TNZSPEC-0107	
org	com.vmware.tanzu	
title	When using the Bouncy Castle Java (D)TLS API and JSSE Provider, the module must be operated in FIPS mode.	
description	The BCJSSE provider has a FIPS mode that helps restrict the provider to cipher suites and parameters that can be offered in a FIPS compliant TLS client or server setup.	
reference	**description**	"Java (D)TLS API and JSSE Provider User Guide Version 1.0.18, Section 2.1.1 Configuring the BCJSSE Provider in FIPS mode"
	url	https://downloads.bouncycastle.org/fips-java/docs/BC-FJA-%28D%29TLSUserGuide-1.0.18.pdf
tags	• FIPS-140-3 • BouncyCastle	

Custom Standards Support and Validation

The Enterprise Spring Boot Governance Starter `governance-starter` library supports adding and validating custom specifications. This feature allows you to incorporate organization-specific regulations, industry guidelines, or proprietary standards into your validation processes.

To run validation against custom specifications, follow these steps:

1. Add custom governance specs by implementing the `GovernanceSpecProvider` bean.
2. Create a class in which to store your collected information.
3. Implement a `GovernanceDetailsScanner` bean to gather details and populate the class created in the previous step.
4. Implement a `GovernanceValidator` bean to validate against the collected data.

A spec id can only be validated by a single validator. A validator can only validate a single spec id.
That is, a spec id has a 1:1 mapping to a `GovernanceValidator` bean.

Define a GovernanceSpecProvider bean to add custom specs

To add custom governance specs, see the following example:

```
package com.example.compliance;

import com.vmware.tanzu.spring.governance.spec.GovernanceSpec;
import com.vmware.tanzu.spring.governance.spec.GovernanceSpecProvider;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
public class ExampleOrgGovernanceSpecProvider implements GovernanceSpecProvider {

        @Override
        public List<GovernanceSpec> getSpecs() {
                return List.of(getSpec());
        }

        private static GovernanceSpec getSpec() {
                var spec = new GovernanceSpec();
                spec.setId("MY-ORG-0001");
                spec.setTitle("TLS must be enabled");
                spec.setDescription("As per org rules, all apps must have TLS enable
d");
                spec.setOrg("com.example");
                spec.setTags(List.of("MY-ORG"));
                return spec;
        }
}
```

Create a custom class to store application details

To create a class in which to store your collected information, see the following example of a custom class:

```
public record ExampleOrgComplianceDetails(
        boolean serverTlsEnabled,
        boolean managementTlsEnabled) { }
```

Define a GovernanceDetailsScanner bean

Gather application details and populate the class created earlier. Implement the interface methods, where:

- `getKey()` is the key for the details object. In the validator bean, a map will be provided. Use this key to fetch the details object from the map.
- `scan()` returns an instance of the collected details object.

```
package com.example.compliance;

import com.vmware.tanzu.spring.governance.GovernanceDetailsScanner;
import org.springframework.boot.actuate.autoconfigure.web.server.ManagementServerPrope
```

```
rties;
import org.springframework.boot.autoconfigure.web.ServerProperties;
import org.springframework.stereotype.Component;

@Component
public class ExampleOrgDetailsScanner implements GovernanceDetailsScanner {

        static final String KEY = "exampleDetails";
        private final ServerProperties serverProperties;
        private final ManagementServerProperties managementServerProperties;

        ExampleOrgDetailsScanner(ServerProperties serverProperties,
                ManagementServerProperties managementServerProperties) {
                this.serverProperties = serverProperties;
                this.managementServerProperties = managementServerProperties;
        }

        @Override
        public String getKey() {
                return KEY;
        }

        @Override
        public Object scan() {
                var serverTlsEnabled = serverProperties.getSsl() != null
                        && serverProperties.getSsl().isEnabled();
                var actuatorIsUsingSameTlsConfig = serverTlsEnabled
                        && managementServerProperties.getSsl() == null;
                var actuatorHasSeparateTlsConfig = managementServerProperties.getSsl()
!= null
                        && managementServerProperties.getSsl().isEnabled();
                var managementTlsEnabled = actuatorIsUsingSameTlsConfig || actuatorHas
SeparateTlsConfig;
                return new ExampleOrgComplianceDetails(serverTlsEnabled, managementTls
Enabled);
        }
}
```

Create a GovernanceValidator bean to run your validation rules

Implement the interface methods, where:

- `requiresKey()` is the key defined in the scanner where this validator can find the details.
- `appliesToSpec()` where the validator returns the `GovernanceSpec` it validates.
- `validate()` where the validator returns a `List<ValidationTestRun>` after validating the details.

```
package com.example.compliance;

import com.vmware.tanzu.spring.governance.GovernanceValidator;
import com.vmware.tanzu.governance.ValidationState;
import com.vmware.tanzu.spring.governance.ValidationTestRun;
import com.vmware.tanzu.spring.governance.spec.GovernanceSpec;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.stereotype.Component;
```

```
import java.util.List;
import java.util.Map;

@Component
public class ExampleOrgTlsEnabledValidator implements GovernanceValidator {

        private static final String ID = "MY-ORG-0001";

        @Override
        public String requiresKey() {
                return ExampleOrgDetailsScanner.KEY;
        }

        @Override
        public GovernanceSpec appliesToSpec(List<GovernanceSpec> list) {
                return list.stream()
                        .filter(spec -> ID.equals(spec.getId()))
                        .findFirst()
                        .orElse(null);
        }

        @Override
        public List<ValidationTestRun> validate(Map<String, Object> appDetails) {
                ExampleOrgComplianceDetails exampleDetails = (ExampleOrgComplianceDeta
ils) getDetails(appDetails);

                var serverTlsEnabledTest = new ValidationTestRun();
                var passedTest1 = exampleDetails.serverTlsEnabled() ? ValidationState.
PASS : ValidationState.FAIL;
                serverTlsEnabledTest.setState(passedTest1);
                serverTlsEnabledTest.setDescriptionFormat("Observed server tls enable
d: %tlsEnabled");
                serverTlsEnabledTest.setParameters(Map.of("tlsEnabled", passedTest1));

                var managementTlsEnabledTest = new ValidationTestRun();
                var passedTest2 = exampleDetails.managementTlsEnabled() ? ValidationSt
ate.PASS : ValidationState.FAIL;
                managementTlsEnabledTest.setState(passedTest2);
                managementTlsEnabledTest.setDescriptionFormat("Observed management tls
enabled: %tlsEnabled");
                managementTlsEnabledTest.setParameters(Map.of("tlsEnabled", passedTest
2));

                return List.of(serverTlsEnabledTest, managementTlsEnabledTest);
        }
}
```

Also note:

- The interface method `getDetails()` returns the details from the `appDetails` Map using the key.
- The default interface method `appliesToDetails(Map<String, Object> appDetails)` (not shown) can be overridden to configure whether the validator should be run against the details. When it returns `false`, `validate()` is not executed. The total test cases do not include this test.
- The string in `ValidationTestRun.setDescriptionFormat()` can include parameters; for example, `"Observed management tls enabled: %tlsEnabled"`. These parameters must be present as

keys in the Map for `ValidationTestRun.setParameters()`. In the output JSON, the value will be used; for example, `"Description": "Observed server tls enabled: true"`

Validation State

The test result is represented by the `ValidationState`:

- `PASS` - the test succeeded
- `FAIL` - the test failed
- `UNKNOWN` - an issue occurred while collecting data or running the test.
  For example, the library uses reflection while collecting the configuration of the application. In the case where reflection fails, the result is treated as `UNKNOWN`. If you encounter this result when running the pre-defined tests, contact Broadcom Support with your use case.

If there are failing or unknown tests, the library forces the application to shut down. However, you can bypass the shutdown by deactivating the specific test, or by deactivating the `exit-on-failure` flag.

See Library Configuration Options for further instructions.

Run the application

Run the app and access the Governance Endpoint.
Filter by the tag for your spec: `https://localhost:8443/actuator/governance?tag=MY-ORG`.

Observe the details field in the JSON that includes a field, `exampleDetails`, from the `ExampleOrgDetailsScanner`. You will see the test run result shows failure.

Set the management port on a separate port,; then the test will pass.

```
management:
  server:
    port: 9443
```

Troubleshooting

This topic describes issues you may run into while setting up Enterprise Spring Boot Governance Starter in your application dependencies, along with solutions or workarounds.

Problems running your app as a fat jar

After adding Enterprise Spring Boot Governance Starter as a dependency to your application, you may run into the following error when running the fat jar produced by the Spring Boot plug-in:

```
11:42:15.126 [main] ERROR org.springframework.boot.SpringApplication -- Application ru
n failed
org.bouncycastle.crypto.fips.FipsOperationError: Module checksum failed: unable to fin
d
        at org.bouncycastle.crypto.fips.FipsStatus.checksumValidate(Unknown Source)
        at org.bouncycastle.crypto.fips.FipsStatus.isReady(Unknown Source)
    ...
```

Cause

The current version of Bouncy Castle is not compatible with the nested jar support in Spring Boot 3.2. For more details, see bc-fips and SpringBoot 3.2 compatibility issue.

Solution

Until the new Bouncy Castle fix is approved again, the workaround is to use the Spring Boot loader fallback option to your application:

- For Gradle:

```
bootJar {
    loaderImplementation = org.springframework.boot.loader.tools.LoaderImplemen
tation.CLASSIC
}
```

- For Maven:

```
<build>
<plugins>
    <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
        <executions>
        <execution>
            <goals>
                <goal>repackage</goal>
            </goals>
            <configuration>
                <loaderImplementation>CLASSIC</loaderImplementation>
            </configuration>
        </execution>
        </executions>
    </plugin>
</plugins>
</build>
```

Tanzu Local Authorization Server

Tanzu Local Authorization Server helps with running Authorization Server locally, without relying on external services (Okta, Azure Entra, …) or a more heavyweight solutions (Keycloak). It provides sane defaults and just enough features to produce `access_token`s and `id_token`s that look like prod tokens.

Tanzu Local Authorization Server also helps with unit-and-integration testing. By leveraging the experimental Spring Boot Testjars project, or by packaging it in a Docker image, tests can rely on a fast-starting authserver.

- Release Notes
- Getting Started with Local Authorization Server
- Using Local Authorization Server in your Tests
- Reference Configuration

Local Authorization Server Release Notes

These are the release notes for VMware Tanzu Local Authorization Server.

v1.0.1

Release Date: December 19th, 2024

- Default OAuth2 client supports OAuth2 Token Exchange grant.

v1.0.0

Release Date: October 25th, 2024

- Add custom login page.
- Add OAuth2 provider name in default Spring Boot config printed to the console on startup.

v0.0.7

Release Date: August 20th, 2024

- Add support for OpenID Connect RP-initiated logout
- Add support for Cross-Origin Resource Sharing (CORS) on every domain, with `Access-Control-Allow-Origin: "*"`

0.0.6

This is the first release of VMware Tanzu Local Authorization Server.

Getting Started with Local Authorization Server

VMware Tanzu Local Authorization Server offers a simple, lightweight solution for running an OAuth2 Authorization Server / OpenID Connect Provider locally.

To install Tanzu Local Authorization Server:

1. Follow the instructions on Broadcom Support, to obtain access to the Spring Enterprise Subscription, and save the secure token for access.
2. Using the token, download the jar file for the latest release.
3. Alternatively, if your company is mirroring the Broadcom artifactory, you can download it directly using Maven. Find the current version from the release notes page, and then run:

   ```
   mvn dependency:copy \
    -Dartifact=com.vmware.tanzu.spring:tanzu-local-authorization-server:<VERSION>
   \
    -DoutputDirectory=.
   ```

   To run Tanzu Local Authorization Server, Ensure that Java 17+ is installed, and then run:

   ```
   java -jar tanzu-local-authorization-server-<VERSION>.jar
   ```

   The command line output explains how to use the app in a Spring Boot client app, and which users can be used to log in. By default, Tanzu Local Authorization Server registers one Client application, and one user that can be used to log in. When using the default Client (`client-id: default-client`), Tanzu Local Authorization Server does not validate the `redirect_uri` or the `scope` parameters when making requests: all scopes are allowed, and all `redirect_uri`s are considered valid.

   To use custom configuration and to customize Tanzu Local Authorization Server, run the following command.
   For more information, see Tanzu Local Authorization Server Reference Configuration.

   ```
   java -jar tanzu-local-authorization-server-<VERSION>.jar --config=my-configurat
   ion.yml
   ```

Role-based or attribute-based access control using OpenID claim

Users of Tanzu Local Authorization Server can be defined in a configuration file. Arbitrary user attributes can be defined, and, when the Client requests the `profile` scope, those attributes are translated into `id_token` claims. The Client can then use the additional claims to make authorization decisions.

1. First, create a configuration file; for example, `config.yml`:

   ```
   tanzu:
     local-authorization-server:
       users:
   ```

```
      - username: alice
        password: alice-password
        attributes:
          # email is a standard OpenID claim, obtained with the email scope
          email: "alice@example.com"
          # roles is a custom, application-specific claim
          roles:
            - viewer
            - editor
            - admin
      - username: bob
        password: bob-password
        attributes:
          email: bob@example.com
          roles:
            - viewer
            - editor
```

2. Then, run Tanzu Local Authorization Server:

```
java -jar tanzu-local-authorization-server-<VERSION>.jar --config=config.yml
```

3. Copy the sample configuration that the authorization server prints out in the console, and use it in your client application.
   Ensure that the `openid` and `profile` scopes are included in `spring.security.oauth2.client.registration.tanzu-local-authorization-server.scope`.

4. Finally, configure your client application to extract authorities from the custom `roles` claim, by providing an `OidcUserService` bean:

```
@Bean
OidcUserService oidcUserService() {
    var oidcUserService = new OidcUserService();
    oidcUserService.setOidcUserMapper((oidcUserRequest, oidcUserInfo) -> {
        // Will map the "roles" claim from the `id_token` into user authorities
(roles)
        var roles = oidcUserRequest.getIdToken().getClaimAsStringList("roles");
        var authorities = AuthorityUtils.createAuthorityList();
        if (roles != null) {
            roles.stream()
                    .map(r -> "ROLE_" + r)
                    .map(SimpleGrantedAuthority::new)
                    .forEach(authorities::add);
        }
        return new DefaultOidcUser(authorities, oidcUserRequest.getIdToken(), o
idcUserInfo);
    });
    return oidcUserService;
}
```

5. You can then check Roles in request or method security:

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http
```

```
            .authorizeHttpRequests(auth -> {
                auth.requestMatchers("/public/**").permitAll();
                auth.requestMatchers("/document/**").hasAnyRole("viewer", "edit
or", "admin");
                auth.requestMatchers("/admin/**").hasRole("admin");
                auth.anyRequest().authenticated();
            })
            .oauth2Login(Customizer.withDefaults())
            .build();
}
```

TLS support

Tanzu Local Authorization Server is designed to be used locally. Since there is no easy PKI-based TLS support for local development, Tanzu Local Authorization Server does not support serving traffic over TLS. However, some tools or libraries may require traffic to be served over TLS.

While TLS is not supported first-class, Tanzu Local Authorization Server is built on Spring Boot, and exposes Spring Boot configuration properties. Spring Boot can be configured to serve traffic over TLS. Update the configuration and add the following properties:

```
spring:
  ssl:
    bundle:
      pem:
        server:
          keystore:
            certificate: /path/to/certificate/localhost.pem
            private-key: /path/to/private/key/localhost-key.pem

server:
  ssl:
    bundle: server
    client-auth: NONE
```

Using Local Authorization Server in your Tests

This topic describes how to use VMware Tanzu Local Authorization Server in your tests.

Using in tests with Testcontainers

Tanzu Local Authorization Server can be packaged in a container, and used in tests with Testcontainers and Spring Boot support for Testcontainers.

1. First, configure the test profile with the default configuration that is printed in the CLI, as explained in Getting started. Then, add Testcontainer support to the project:

   Gradle

   ```
   testImplementation("org.springframework.boot:spring-boot-testcontainers")
   testImplementation("org.testcontainers:junit-jupiter")
   ```

   Maven

```
<dependencies>
    <!-- ... -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-testcontainers</artifactId>
    </dependency>
    <dependency>
        <groupId>org.testcontainers</groupId>
        <artifactId>junit-jupiter</artifactId>
    </dependency>
    <!-- ... -->
</dependencies>
```

2. Configure `@SpringBootTest` to use Tanzu Local Authorization Server and Testcontainers:

```
@Testcontainers(disabledWithoutDocker = true)
@SpringBootTest
class TestcontainersTests {

    @Container
    static GenericContainer<?> tanzuAuthServer = new GenericContainer<>("bellso
ft/liberica-openjre-alpine:21")
            .withCopyFileToContainer(
                    // Point Testcontainers to the Tanzu-Local-Authorization-Se
rver release
                    MountableFile.forHostPath("path/to/tanzu-local-authorizatio
n-server-<VERSION>.jar"),
                    "/tanzu-local-authorization-server.jar")
            .withCommand("java", "-jar", "/tanzu-local-authorization-server.ja
r")
            .withExposedPorts(9000);

    @Test
    void contextLoads() {
    }

    @DynamicPropertySource
    static void clientRegistrationProperties(DynamicPropertyRegistry registry)
{
        // This will configure your Spring Boot app to point to the running con
tainer
        registry.add("spring.security.oauth2.client.provider.tanzu-local-author
ization-server.issuer-uri",
                () -> "http://localhost:" + tanzuAuthServer.getFirstMappedPort
());
    }

}
```

Using Local Authorization Server in tests with Spring Boot Testjars

Combined with Spring Boot Testjars, using Tanzu Local Authorization Server with Spring Boot Testjars allows for fast testing without relying on Docker.

1. First, configure the test profile with the default configuration that is printed in the CLI, as explained in the Getting started section above. Then, add Spring Boot Testjars to the project:

   Gradle

   ```
   testImplementation("org.springframework.experimental.boot:spring-boot-testjars:0.0.2")
   ```

   Maven

   ```
   <dependency>
       <groupId>org.springframework.experimental.boot</groupId>
       <artifactId>spring-boot-testjars</artifactId>
       <version>0.0.2</version>
   </dependency>
   ```

2. Configure `@SpringBootTest`s to use Tanzu Local Authorization Server:

   ```
   @SpringBootTest
   class MyApplicationTests {

       @Test
       void contextLoads() {
       }

       @TestConfiguration(proxyBeanMethods = false)
       @EnableDynamicProperty
       static class TestOAuth2Login {

           @Bean
           @OAuth2ClientProviderIssuerUri(providerName = "tanzu-local-authorization-server")
           static CommonsExecWebServerFactoryBean authorizationServer() {
               return CommonsExecWebServerFactoryBean.builder()
                       // Point Spring Boot Testjars to the Tanzu-Local-Authorization-Server release
                       .classpath(cp -> cp.files("path/to/tanzu-local-authorization-server-<VERSION>.jar"))
                       // Instruct Spring Boot Testjars to use the Spring Boot 3.3 JarLauncher (bundled in Tanzu-Local-Authorization-Server)
                       .mainClass("org.springframework.boot.loader.launch.JarLauncher");
           }

       }

   }
   ```

For more information about Spring Boot Testjars and OAuth2 clients, see the official Github repo.

Reference Configuration

VMware Tanzu Local Authorization Server can be further customized with the following configuration properties. When a `user` is defined through custom configuration, the default user will not be registered.

When a `client` is defined through custom configuration, the default client is not configured. Clients registered through configuration can be configured to enable `redirect_uri` and `scope` validation.

The following configuration can also be created by running the Tanzu Local Authorization Server with the `--print-sample-config` flag.

```
server: # OPTIONAL
  # The port on which Tanzu Local Authorization Server runs. Defaults to 9000.
  port: 9000
tanzu:
  local-authorization-server:
    # OPTIONAL: whether to use a hardcoded RSA key for JWT signing, or a randomly gene
rated one.
    # Hardcoded keys mean faster startup time.
    jwk:
      # Defaults to false
      random: false
    # OPTIONAL: custom users for logging in
    users:
      - username: my-user # REQUIRED
        password: clear-text-password # REQUIRED
        # Attributes are added to the id_token based on requested scopes.
        # All attributes are optional.
        attributes: # OPTIONAL
          # standard OpenID Connect attributes:

          # scope: profile
          name: "Jane T. Spring"
          given_name: "Jane"
          family_name: "Spring"
          middle_name: "Team"
          nickname: "Spring"
          preferred_username: "jtspring"
          profile: "https://spring.io/team"
          picture: "https://spring.io/img/spring-2.svg"
          website: "https://spring.io"
          gender: "unspecified"
          birthdate: "1970-01-01"
          zoneinfo: "Europe/Paris"
          locale: "fr-FR"

          # scope: email
          email: "jane.spring@example.com"
          email_verified: true

          # scope: phone_number
          phone_number: "+1 (555) 555-1234"
          phone_number_verified: true

          # scope: address
          address:
            formatted: "1, OpenID St., Openid.net City, 1234 Identity Realm, Internet"
            street_address: "1, OpenID St."
            locality: "Openid.net City"
            region: "Identity Realm"
            postal_code: "1234"
            country: "Internet"
```

```
          # all other attributes are custom ("user-defined"), and added to the id_toke
n claims when
          # the "profile" scope is requested
          some-claim: "some-value"
          custom-age: 42
      - username: other-user
        password: other-password

    # OPTIONAL: custom client registrations, which must match the client application's
    # spring.security.oauth2.client.registration.<id>.* properties
    clients:
      - client-id: "custom-client"
        client-secret: "custom-secret"
        # MUST be one or more of the following
        client-authentication-methods:
          - "client_secret_basic"
          - "client_secret_post"
          - "none"
        # MUST be one or more of the following
        authorization-grant-types:
          - "authorization_code"
          - "client_credentials"
          - "refresh_token"
        # OPTIONAL, can be anything
        scope:
          - "openid"
          - "email"
          - "profile"
          - "address"
          - "phone"
          - "message.read"
          - "message.write"
        # REQUIRED when authorization-grant-type contains authorization_code, otherwis
e OPTIONAL
        redirect-uris:
          # This is default Spring Boot redirect URI for the tanzu-local-authorization
-server provider
          - "http://127.0.0.1:8080/login/oauth2/code/tanzu-local-authorization-server"
          - "http://localhost:8080/login/oauth2/code/tanzu-local-authorization-server"
          # Here are other examples:
          - "http://127.0.0.1:8081/authorized"
          - "http://127.0.0.1:8082/callback"
        # OPTIONAL: show the "consent" screen on the /oauth2/authorize call. Defaults
to false.
        require-consent: false
        # OPTIONAL: enforce redirect_uri validation. When set to true, Clients may onl
y use one of
        # the redirect_uris defined for this client. Defaults to false.
        validate-redirect-uri: false
        # OPTIONAL: enforce scope validation. When set to true, clients may only reque
st
        # the scopes defined for this client. Defaults to false.
        validate-scope: false

      - client-id: "other-client"
        client-secret: "other-secret"
        client-authentication-methods:
          - "client_secret_basic"
```

```
        authorization-grant-types:
          - "client_credentials"
```

Tanzu Spring Config Server

VMware Tanzu Spring Config Server is an externalized configuration server based on the open-source Spring Cloud Config project. It provides a centralized server for delivering external configuration properties to an application, and acts as a central source for managing this configuration across deployment environments. Tanzu Spring Config Server supports a number of backends, including Git and Hashicorp Vault.

Tanzu Spring Config Server is released as a standalone JAR and a capability:

- Tanzu Spring Config Server - standalone JAR
- Tanzu Spring Config Server - capability

Tanzu Spring Config Server - standalone JAR

The Tanzu Spring Config Server Standalone JAR requires Java 17 or higher.

- Tanzu Spring Config Server Release Notes
- Installing Spring Config Server
- Enabling Mutual TLS (mTLS)
- Running the Config Server
- Configuring the Config Server
- Enabling Client Applications

Config Server Release Notes

These are the release notes for VMware Tanzu Spring Config Server Standalone JAR

v1.0.0

Release Date: May 22, 2024

This is the first release.

Installing Spring Config Server

This topic provides instructions for installing the Tanzu Spring Config Server standalone JAR.

1. Follow the instructions at Broadcom Support to obtain access to the Spring Enterprise Subscription, and save the secure token for access.

2. Download Config Server from the Spring Artifact Repository: `https://packages.broadcom.com/artifactory/spring-enterprise/com/vmware/tanzu/spring/tanzu-config-server/`.
3. Create a YAML file that contains the configuration for the Config Server. This file can be named anything you like and be located anywhere on the filesystem.

   At minimum, this configuration file should specify a Git repository from which to serve configuration. For example:

   ```
   spring:
     cloud:
       config:
         server:
           git:
             uri: https://github.com/spring-cloud-services-samples/cook-config
   ```

   See Configuring the Spring Config Server Standalone JAR for more configuration options.
4. Specify the location of the configuration file by setting an environment variable named `SPRING_CONFIG_ADDITIONAL_LOCATION` or `SPRING_CONFIG_IMPORT`. Assuming that the configuration file is named `config-server.yml` and is placed in a directory named `samples`, you could set `SPRING_CONFIG_IMPORT` by running:

   ```
   export SPRING_CONFIG_IMPORT=samples/config-server.yml
   ```
5. The Spring Cloud Config Server JAR file is an executable JAR file. Using Java 17+ or higher, run the Config Server:

   ```
   java -jar config-server-1.0.0.jar
   ```
6. After a few moments, the Config Server should be running and listening on port 8888 (unless you set a different value for `server.port` in the configuration) file. To verify, you can use `curl` to fetch the configuration for the default application and profile by running:

   ```
   curl localhost:8888/application/default
   ```

Enabling Mutual TLS (mTLS)

This topic describes how to configure the Tanzu Spring Config Server standalone JAR to use mTLS.

1. Create the configuration file to include SSL properties.

   ```
   server:
     ssl:
       bundle: server
       client-auth: NEED

   spring:
     cloud:
       config:
         server:
           git:
   ```

```
        uri: https://github.com/spring-cloud-services-samples/cook-config
  ssl:
    bundle:
      pem:
        server:
          keystore:
            certificate: samples/tls/server/tls.crt
            private-key: samples/tls/server/tls.key

          truststore:
            certificate: samples/tls/ca/tls.crt
```

2. Set `SPRING_CONFIG_ADDITIONAL_LOCATION` or `SPRING_CONFIG_IMPORT` to reference this configuration, as described in Installing Config Server.
3. Run the application as an executable JAR file as described in Installing Config Server:

```
java -jar config-server-1.0.0.jar
```

4. Test it by supplying certificates and keys in the request:

```
curl \
    --cacert samples/tls/ca/tls.crt \
    --cert samples/tls/client/tls.crt \
    --key samples/tls/client/tls.key \
    https://localhost:8888/cook/default/main
```

Running the Config Server

This topic shows you how to run the Spring Config Server standalone JAR.

1. Create an image from the Config Server JAR file. The easiest way to do this is to use the `pack` command:

```
pack build tanzu/config-server:1.0.0 \
  --path ./config-server-tsr-1.0.0.jar \
  --builder paketobuildpacks/builder:tiny
```

   - If you will be running the image on an ARM host (such as an Apple machine with an Apple chipset), you must use a different builder:

```
pack build tanzu/config-serverx:1.0.0 \
  --path ./config-server-tsr-1.0.0.jar \
  --builder dashaun/builder:tiny
```

   - Alternatively, you can create an image using `docker build`. Create a Dockerfile with the following contents:

```
FROM openjdk:17-jdk
COPY config-server-1.0.0.jar cs.jar
ENTRYPOINT [ "java", "-jar", "cs.jar" ]
```

     This assumes that the JAR file is in the directory where you will create the image. Using the Docker CLI, create the image with this command (substitute “tanzu” with your

organization's Docker repository name):

```
docker build -t "tanzu/config-server:1.0.0" .
```

2. Create a configuration file. The example shown here is the minimum required. Your configuration is expected to be more comprehensive.

```
spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/spring-cloud-services-samples/cook-config
```

3. Make the configuration available to the container. The most basic way of doing this is to use a bind mount to mount a directory containing the configuration YAML file. For example, if the `config-server.yml` file is in a directory name `csconfig`, start the container by running:

```
docker run -d \
  -p 8888:8888 \
  --mount type=bind,source="$(pwd)"/csconfig,target=/csconfig
  -e SPRING_CONFIG_IMPORT='/csconfig/config-server.yml'
  tanzu/config-server:1.0.0
```

   This starts the container, forwards the local port 8888 to the Config Server's port 8888 running in the container, and sets the `SPRING_CONFIG_IMPORT` environment variable to reference the mounted configuration file.

4. Test it by making a request to the Config Server using curl:

```
curl localhost:8888/application/default
```

Configuring the Config Server

This topic describes the VMware Tanzu Spring Config Server standalone JAR properties you can add to your configuration YAML file.

In addition to configuring a Git URI, Config Server offers other properties for configuring Config Server to suit your needs. Place these configuration settings in your configuration YAML file.

Configuring Git Backends

All of the properties for configuring Git repositories are prefixed with `spring.cloud.config.server.git`.

Property	Default	Description
clone-on-start	false (start on demand)	Flag to indicate that the repository should be cloned on startup
refresh-rate	0 (always refresh)	Time (in seconds) between refreshes of the git repository
basedir		Base directory for local working copy of repository

Property	Default	Description
clone-submodules	false	Flag to indicate that the submodules in the repository should be cloned
default-label		The default label to be used with the remote repository
delete-untracked-branches	false	Flag to indicate that the branch should be deleted locally if its origin tracked branch was removed
force-pull	false	Flag to indicate that the repository should force pull
host-key		Valid SSH host key
host-key-algorithm		One of ssh-dss, ssh-rsa, ssh-ed25519, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, or ecdsa-sha2-nistp521. Must be set if `hostKey` is set.
ignore-local-ssh-settings	false	If `true`, use property-based instead of file-based SSH config.
known-hosts-file		Location of `.known_hosts` file
order		The order of the environment repository
passphrase		Passphrase for unlocking your SSH private key
password		Password for authentication with remote repository
preferred-authentications		Override server authentication method order
private-key		Valid SSH private key
proxy		HTTP proxy configuration
repos		Map of repository identifier to location and other properties
search-paths		Search paths to use in local working copy
skip-ssl-validation	false	Flag to indicate that SSL certificate validation should be bypassed when communicating with a repository served over an HTTPS connection
strict-host-key-checking	true	If false, ignore errors with the host key
timeout	5	Timeout (in seconds) for obtaining HTTP or SSH connection (if applicable)
try-master-branch	true	To maintain compatibility, try the `master` branch in addition to `main` when we try to fetch the default branch
username		Username for authentication with remote repository

Configuring Vault Backends

When using Vault backends, you must enable the `vault` profile:

```
spring:
  profiles:
    active: vault
```

If you are planning to use Vault backends alongside Git backends, you must explicitly enable both the `vault` profile and the `git` profile:

```
spring:
  profiles:
    active: vault,git
```

All properties for configuring Hashicorp Vault as a backend are prefixed with `spring.cloud.config.server.vault`. For example, here is a simple Vault configuration that references a Vault server running on the same machine as the Config Server:

```
spring:
  profiles:
    active: vault
  cloud:
    config:
      server:
        vault:
          host: 127.0.0.1
          port: 8200
```

What follows are the properties you can use to configure a Vault backend for Config Server.

Property	Default	Description
backend	secret	Vault backend.
default-key	application	The key in vault shared by all applications
host	127.0.0.1	Vault host
kv-version	1	Value to indicate which version of Vault kv backend is used
namespace		The value of the Vault X-Vault-Namespace header
path-to-key		KV2 API required `data` after `mount-path`
port	8200	Vault port
profile-separator	, (comma)	Vault profile separator
proxy		HTTP proxy configuration
scheme	http	Vault scheme
skip-ssl-validation	false	Flag to indicate that SSL certificate validation should be bypassed when communicating with a repository
ssl.cert-auth-path	cert	Mount path of the TLS cert authentication backend
ssl.key-store		Trust store that holds certificates and private keys
ssl.key-store-password		Password used to access the key store
sss.trust-store		Trust store that holds SSL certificates

Property	Default	Description
ssl.trust-store.password		Password used to access the trust store
timeout	5	Timeout (in seconds) for obtaining HTTP connection
token		Static Vault token

Configuring Composite Backends

If you want to configure two or more of the same kind of backend (for example, three Git backends, two Vault backends, or some combination of Git and Vault where there are two or more of one of them), you must use the composite configuration style.

For example, suppose that you have configured one each of a Git and Vault backend like this:

```
spring:
  profiles:
    active: git, vault
  cloud:
    config:
      server:
        git:
          uri: https://github.com/spring-cloud-services-samples/cook-config
          order: 2
        vault:
          host: 127.0.0.1
          port: 8200
          order: 1
```

And then you decide to add another Git backend to the configuration. The `spring.cloud.config.server.git` properties are already set, so you cannot set another backend using the same properties. Instead, you need to set the `spring.cloud.config.server.composite` property with an array of backends as shown in the following example:

```
spring:
  profiles:
    active: git, vault
  cloud:
    config:
      server:
        composite:
        - type: git
          uri: https://github.com/someorg/other-config
          order: 3
        - type: git
          uri: https://github.com/spring-cloud-services-samples/cook-config
          order: 2
        - type: vault
          host: 127.0.0.1
          port: 8200
          order: 1
```

Enabling Client Applications

This topic contains instructions for enabling client applications.

Spring Boot applications can use the Tanzu Spring Config Server Standalone JAR by including the client dependency in their builds and configuring connection details to access the running Config Server. This results in externalized configuration, served by the Tanzu Spring Config Server standalone JAR, being injected into the Spring `Environment`, and being available to inject into application properties annotated with `@Value`, or in beans that are annotated with `@ConfigurationProperties`.

Adding the Client Dependency to your Build

To use a Tanzu Spring Config Server, a client app must include the necessary client dependency. Specifically, you must add the Spring Cloud OSS Config Server Client dependency to your project's build. In addition, you must add the Spring Cloud Bill of Materials (BOM) into the build's dependency management.

For Gradle builds

For a Gradle build, the Config Server Client dependency looks like this:

```
implementation 'org.springframework.cloud:spring-cloud-starter-config'
```

The dependency management entry should look similar to this:

```
dependencyManagement {
  imports {
    mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersio
n}"
  }
}
```

Set the `springCloudVersion` property to reference the latest Spring Cloud OSS version:

```
ext {
  set('springCloudVersion', "2023.0.1")
}
```

For Maven builds

If your project is built with Maven, add the following dependency to the `<dependencies>` section of the build:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

The dependency management section should include the Spring Cloud BOM:

```
<dependencyManagement>
  <dependencies>
```

```
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

And you can set the `spring-cloud.version` property in the `<properties>` section like this:

```
<properties>
  <spring-cloud.version>2023.0.1</spring-cloud.version>
</properties>
```

Specifying Connection Details

If Config Server is running locally and listening on port 8888, you can add the following entry to the `application.properties` file for your application:

```
spring.config.import=optional:configserver:
```

The `optional:` prefix is optional, but if it is omitted, this will cause the Config Client to fail if it is unable to connect to the Config Server.

If Config Server is running elsewhere, you must set `spring.config.import` to reference the location of the Tanzu Spring Config Server. For example, if Tanzu Spring Config Server is running on a host named `myconfigserver` and listening on port 8888, here is how to set `spring.config.import`:

```
spring.config.import=optional:configserver:http://myconfigserver:8888
```

Enabling TLS (mTLS) Authentication

If Tanzu Spring Config Server requires TLS authentication, you can configure the client side certificates and trust store in `application.properties` with the following entries:

```
spring.config.import=optional:configserver:http://myconfigserver:8888
spring.cloud.config.tls.enabled: true
spring.cloud.config.tls.key-store: <path-to-key-store>
spring.cloud.config.tls.key-store-type: PKCS12
spring.cloud.config.tls.key-store-password: <key-store-password>
spring.cloud.config.tls.password: <key-password>
spring.cloud.config.tls.trust-store: <path-of-trust-store>
spring.cloud.config.tls.trust-store-type: PKCS12
spring.cloud.config.tls.trust-store-password: <trust-store-password>
```

Modify the values of these properties with the specific details for your client certification and trust store.

For more information about configuring the Spring Cloud Config Client, see the OSS Spring Cloud Config Client documentation.

Tanzu Spring Config Server - capability

The Tanzu Spring Config Server capability is available on Tanzu Platform.

- Overview
- Release Notes
- Installing Spring Config Server
- Create Config Server Resources
- Configure Workloads to use Config Server Resources
- Create App Config Resources
- Configure Workloads to use App Config Resources

Overview

`ConfigServer` is an externalized configuration server based on the open-source Spring Cloud Config project. `ConfigServer` provides a centralized server for delivering external configuration properties to an application, and a central source for managing this configuration across deployment environments. `ConfigServer` is designed to be consumed by Spring Boot applications.

`AppConfig` is an adapter that is sed to allow non-Spring applications (e.g. Golang, Python, NodeJS, …) to easily consume externalized configuration from `ConfigServer` as well.

Both `ConfigServer` and `AppConfig` are best consumed with service bindings. For more information, see the Tanzu Platform for Kubernetes documentation.

Capacity requirements

Each node of the capability's controller requires:

- 64 MiB of memory
- 10 m of vCPU

with limits of:

- 128 MiB of memory
- 500 m of vCPU

You can scale the controller horizontally for higher availability.

Release Notes

These are the release notes for the VMware Tanzu Spring Config Server Capability `config-server.spring.tanzu.vmware.com`

v1.2.0

Release Date: Nov 25, 2024

- `AppConfig` projects fields *type* and *provider*

v1.1.0

Release Date: Nov 1, 2024

- Introduces `AppConfig`

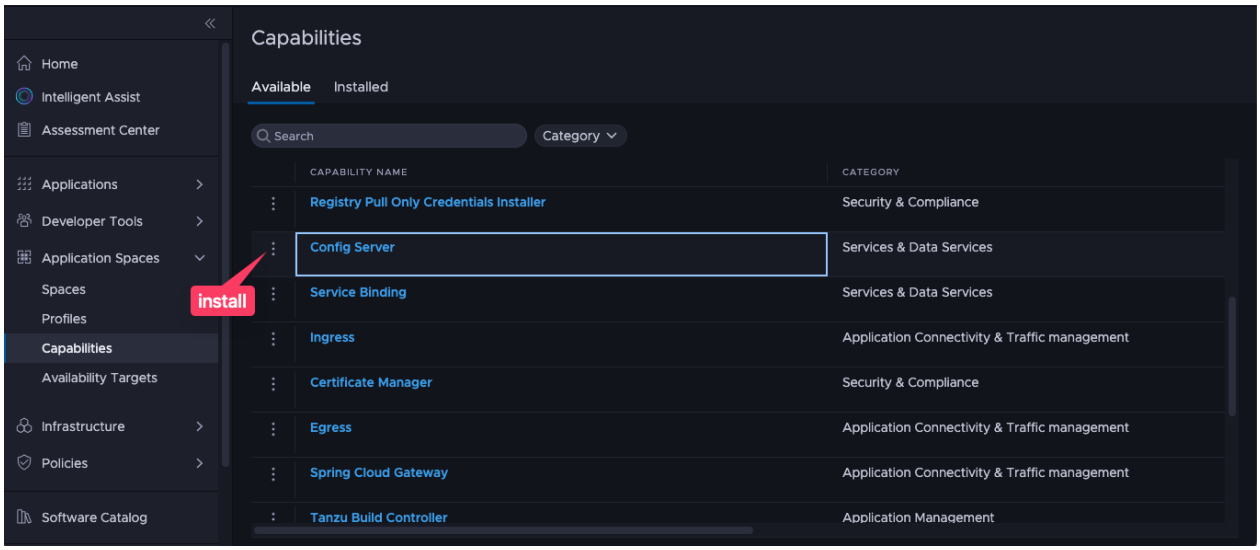
v1.0.0

Release Date: April 29, 2024

This is the first release.

Installing Spring Config Server

The capability `config-server.spring.tanzu.vmware.com` is not installed on cluster groups by default. The capability can be installed from the Tanzu Platform UI.



Create Config Server Resources

This topic tells you about the options available when creating a `ConfigServer` resource.

Detect available parameters

Examine the available parameters when creating a `ConfigServer` resource by running:

```
tanzu space use <your-space>
export KUBECONFIG=~/.config/tanzu/kube/config
kubectl explain configserver.spec
```

For example:

```
$ kubectl explain configserver.spec
KIND:     ConfigServer
VERSION:  config-server.spring.tanzu.vmware.com/v1alpha1

RESOURCE: spec <Object>
```

```
DESCRIPTION:
     ConfigServerSpec defines the desired state of ConfigServer

FIELDS:
   backends  <[]Object> -required-
     List of backends used by the config server. There must be at least one
     backend for config-server.

   replicas  <integer> -required-
     Number of desired config server replicas. Defaults to 1.

   resources  <Object>
     The config server compute resource requirements

   tls  <Object>
     TLS configuration for the config server
```

Here is a second example describing the Git configuration fields:

```
kubectl explain configserver.spec.backends.git
```

For example:

```
$ kubectl explain configserver.spec.backends.git
KIND:     ConfigServer
VERSION:  config-server.spring.tanzu.vmware.com/v1alpha1

RESOURCE: git <Object>

DESCRIPTION:
     Git backend configuration

FIELDS:
   basicAuth  <Object>
     For HTTP/S addresses, the credentials used to access the Git repository if
     protected by HTTP Basic authentication. Optional.

   defaultLabel  <string>
     The default label used if a request is received without a label. Optional:
     Defaults to main.

   paths  <[]string>
     A list of patterns used to search for configuration-containing
     subdirectories in the Git repository. Optional.

   proxy  <Object>
     The Proxy configuration for the Git repository. Optional.

   skipTLSVerify  <boolean>
     For HTTPS addresses, whether to skip validation of the SSL certificate on
     the Git repository's server. Optional: Defaults to false.

   ssh  <Object>
     For SSH addresses, the credentials used to access the Git repository if
     protected by SSH. Optional.

   timeout  <integer>
```

```
    Number of seconds that the config server will wait to acquire a connection
    to the Git repository. Optional: Defaults to 5 seconds.

  ttl  <integer>
    Number of seconds to wait before updating the repository clone from Git
    repository, when a client requests configuration. Optional: Defaults to 0
    seconds. Default value (0) means the repository clone is updated every time
    a client requests configuration. Negative value means the repository clone
    will not be updated, after it is cloned.

  uri  <string> -required-
    The HTTP/S or SSH address of the Git repository.
```

Create a ConfigServer using the Tanzu CLI

To create a `ConfigServer` resource using the `tanzu` CLI for using the Spring Cloud Services demo application cook in a Space called `cook`:

1. Use the Space:

   ```
   tanzu space use cook
   ```

2. Create a `ConfigServer` resource by using the following YAML definition:

   ```
   tanzu service create ConfigServer/cook-server --parameter backends='[{git: {ur
   i: https://github.com/spring-cloud-services-samples/cook-config}}]' --parameter
   replicas=1 --skip-bind-prompt
   ```

3. Open egress to the `ConfigServer`'s github.com backend by running:

   ```
   tanzu egress create github.com --host github.com --port 443 --protocol HTTPS
   ```

   This step is only required if the `egress.tanzu.vmware.com` capability is present in your Space. This opens egress to `https://github.com` for all workloads in the Space. For more information about egress, see the egress documentation.

4. Get the `ConfigServer` resource by running:

   ```
   tanzu service get ConfigServer/cook-server
   ```

Create a ConfigServer using a YAML file

To create a `ConfigServer` resource using a YAML file for using the Spring Cloud Services demo application cook in a Space called `cook`:

1. Create a `ConfigServer` resource by using the following YAML definition:

   ```
   ---
   apiVersion: config-server.spring.tanzu.vmware.com/v1alpha1
   kind: ConfigServer
   metadata:
     name: cook-server
   spec:
   ```

```
  replicas: 1
  backends:
    - git:
        uri: https://github.com/spring-cloud-services-samples/cook-config
```

2. Save the YAML definition as `configserver.yaml`.
3. Create an `EgressPoint` to the `ConfigServer`'s github.com backend by running:

```
---
apiVersion: networking.tanzu.vmware.com/v1alpha1
kind: EgressPoint
metadata:
 name: github.com
spec:
 targets:
   - hosts: ["github.com"]
     port:
       number: 443
       protocol: HTTPS
```

   This step is only required if the `egress.tanzu.vmware.com` capability is present in your Space. This opens egress to `https://github.com` for all workloads in the Space. For more information about egress, see the egress documentation.
4. Save the YAML definition as `egress.yaml`.
5. Use the Space:

```
tanzu space use cook
```

6. Export the Space's kubeconfig:

```
export KUBECONFIG=~/.config/tanzu/kube/config
```

7. Apply the YAML definitions by running:

```
kubectl apply -f configserver.yaml -f egress.yaml
```

8. Get the `ConfigServer` resource by running:

```
kubectl get configserver cook-server
```

Configure Workloads to use Config Server Resources

This topic tells you how to configure Tanzu Platform workloads running Spring Boot applications to connect to `ConfigServer` resources.

Prepare

1. Deploy your workload. For instructions, see Deploy your first application using Spaces.
2. Create a `ConfigServer` resource. For instructions, see Create ConfigServer resources.

Bind the workload to the ConfigServer

Bind your `ContainerApp` to the `ConfigServer` with the `tanzu` CLI:

```
tanzu services bind ConfigServer/<config-server-name> ContainerApp/<workload-name> --a
s config
```

Create App Config Resources

This topic tells you about options when creating an `AppConfig` resource. An `AppConfig` depends on a `ConfigServer`. The `AppConfig`'s service binding `Secret` will contain the configuration key-by-key.

Note that the configuration an AppConfig represents is limited to `1MB` in size. That's due to the size limitation of `Secret`.

Detect available parameters

Examine the available parameters when creating an `AppConfig` resource by running:

```
tanzu space use <your-space>
export KUBECONFIG=~/.config/tanzu/kube/config
kubectl explain appconfig.spec
```

For example:

```
GROUP:      config-server.spring.tanzu.vmware.com
KIND:       AppConfig
VERSION:    v1alpha1

FIELD: spec <Object>


DESCRIPTION:
    AppConfigSpec defines the desired state of AppConfig

FIELDS:
  applications  <[]string>
    <no description>

  configServerRef       <Object> -required-
    LocalObjectReference contains enough information to let you locate the
    referenced object inside the same namespace.

  labels        <[]string>
    <no description>

  profiles      <[]string>
    <no description>
```

The only required field is `configServerRef`, which identifies the `ConfigServer` it should pull configuration from.

Create an AppConfig using the Tanzu CLI

Prerequisite: Create a `ConfigServer`. See Create ConfigServer Resources.

Create an `AppConfig` resource using the `tanzu` CLI in a Space called `cook` for a `ConfigServer` called `cook-server`:

1. Use the Space:

```
tanzu space use cook
```

2. Create an `AppConfig` resource that identifies the `ConfigServer`:

```
tanzu service create AppConfig/cook-config --parameter configServerRef='{name:
cook-server}' --skip-bind-prompt
```

3. Get the `AppConfig` resource by running:

```
tanzu service get AppConfig/cook-config
```

Create an AppConfig using a YAML file

Prerequisite: Create a `ConfigServer`. See Create ConfigServer Resources.

Create an `AppConfig` resource using the `tanzu` CLI in a Space called `cook` for a `ConfigServer` called `cook-server`:

1. Create a `ConfigServer` resource by using the following YAML definition:

```
---
apiVersion: config-server.spring.tanzu.vmware.com/v1alpha1
kind: AppConfig
metadata:
  name: cook-config
spec:
  configServerRef:
    name: cook-server
```

2. Save the YAML definition as `appconfig.yaml`.
3. Use the Space:

```
tanzu space use cook
```

4. Export the Space's kubeconfig:

```
export KUBECONFIG=~/.config/tanzu/kube/config
```

5. Apply the YAML definitions by running:

```
kubectl apply -f appconfig.yaml
```

6. Get the `ConfigServer` resource by running:

```
kubectl get appconfig cook-config
```

Configure Workloads to use App Config Resources

This topic tells you how to configure Tanzu Platform workloads running non-Spring applications to use `AppConfig` for externalized configuration.

Prepare

1. Deploy your workload. For instructions, see Deploy your first application using Spaces.
2. Create a `ConfigServer` resource. For instructions, see Create ConfigServer resources.
3. Create an `AppConfig` resource targeting the `ConfigServer`. For instructions, see Create AppConfig resources.

Bind the workload to the AppConfig

You can bind your `ContainerApp` to the `ConfigServer` using the `tanzu` CLI:

```
tanzu services bind AppConfig/<app-config-name> ContainerApp/<workload-name> --as appc
onfig
```

Read configuration

The AppConfig will be projected onto the workload's filesystem using service bindings.

The service bindings community maintains libraries for several platforms to programmatically access service bindings. See language-specific libraries. These libraries can be used by non-Spring applications to read the configuration provided by an `AppConfig`.

For example, if the workload is implemented in Go it could use github.com/nebhale/client-go:

```
package main

import (
        "encoding/json"
        "fmt"
        "os"

        "github.com/nebhale/client-go/bindings"
)

var config map[string]any

func main() {
        b := bindings.FromServiceBindingRoot()
        b = bindings.Filter(b, "appconfig")
        if len(b) != 1 {
                return nil, fmt.Errorf("expected one appconfig, but got %d", len(b))
        }

        c, ok := bindings.Get(b[0], "__appconfig.json")
        if !ok {
                return nil, fmt.Errorf("expected entry __appconfig.json")
        }
```

```
        if err := json.Unmarshal([]byte(c), &config); err != nil {
                return nil, fmt.Errorf("unable to unmarshal appconfig: %w\n", err)
        }

        // use config ...
}
```

Troubleshooting

ConfigServer is not becoming ready

It is possible that a `ConfigServer` in a Space does not become ready because the egress to its backend is locked.

Open egress to the `ConfigServer` backends by running:

```
tanzu egress create github.com --host github.com --port 443 --protocol HTTPS
```

This step is only required if the `egress.tanzu.vmware.com` capability is present in your Space *and* there are no egress points for the backends yet.

Note that this opens egress to `https://github.com` for all workloads in the Space. For more information about egress, see the egress documentation.

Tanzu Spring Service Registry

VMware Tanzu Service Registry is a service registry based on the open-source Spring Cloud Netflix Eureka Server project. It provides a registry through which applications can register themselves and be discovered by other applications in a microservice architecture.

Tanzu Service Registry requires Java 17 or higher.

Tanzu Service Registry Release Notes

These are the release notes for VMware Tanzu Service Registry.

v1.0.0

Release Date: July 24, 2024

This is the first release.

Installing Tanzu Service Registry

This topic provides instructions for installing Tanzu Service Registry.

1. Follow the instructions at Broadcom Support to obtain access to the Spring Enterprise Subscription, and save the secure token for access.
2. Download Service Registry from the Spring Artifact Repository: `https://packages.broadcom.com/artifactory/spring-enterprise/com/vmware/tanzu/spring/tanzu-service-registry/`.
3. The Spring Cloud Service Registry JAR file is an executable JAR file. Using Java 17+ or higher, run the Service Registry:

   ```
   java -jar tanzu-service-registry-1.0.0.jar
   ```

   After a few moments, the Service Registry should be running and listening on port 8761 (unless you set a different value for `server.port` in the configuration file). To verify, you can open the Service

Registry dashboard.

Alternatively, you can test the installation using `curl` to make a request to the apps endpoint:

```
curl localhost:8761/eureka/apps
```

Configuring Tanzu Service Registry

This topic describes the VMware Tanzu Service Registry properties you can add to your configuration YAML file to enable peer awareness.

By default, Tanzu Service Registry is configured to work in standalone mode, in which it is the only instance. But it can be made even more resilient and available by running multiple instances and asking them to register with each other.

To enable peer awareness, add the following to your configuration YAML:

```
---
spring:
  profiles: peer1
eureka:
  instance:
    hostname: <peer-1-hostname>
  client:
    serviceUrl:
      defaultZone: https://<peer-2-hostname>/eureka/

---
spring:
  profiles: peer2
eureka:
  instance:
    hostname: <peer-2-hostname>
  client:
    serviceUrl:
      defaultZone: https://<peer-1-hostname>/eureka/
```

In this configuration there are two peers, configured under profiles named “peer1” and “peer2”. When running the two instances, you must activate the “peer1” profile for one of the instances and the “peer2” profile for the other instance. One way to do this is to specify the active profiles in an environment variable.

For example, to activate the “peer1” profile, set the following environment variable on the peer-1-host:

```
export SPRING_PROFILES_ACTIVE=peer1
```

Likewise, set the following environment variable on the peer-2-host:

```
export SPRING_PROFILES_ACTIVE=peer2
```

When running using Docker, this can be accomplished by specifying the environment variable with the `-e` flag. For example, when starting the first peer:

```
docker run -d \
    -p 8761:8761 \
    --mount type=bind,source="$(pwd)"/srconfig,target=/srconfig
```

```
    -e SPRING_CONFIG_IMPORT='/srconfig/service-registry.yml'
    -e SPRING_PROFILES_ACTIVE=peer1
    tanzu/service-registry:1.0.0
```

When there are 3 or more peers, the configuration is similar, but can be streamlined to set the default zone in the default profile:

```
eureka:
  client:
    serviceUrl:
      defaultZone: https://<peer-1-host>/eureka/,http://<peer-2-host>/eureka/,http://<
peer-3-host>/eureka/

---
spring:
  profiles: peer1
eureka:
  instance:
    hostname: peer1

---
spring:
  profiles: peer2
eureka:
  instance:
    hostname: peer2

---
spring:
  profiles: peer3
eureka:
  instance:
    hostname: peer3
```

You can add as many peers as desired to a system. As long as they are connected to each other by at least one edge, they will synchronize their registrations with each other.

Enabling Mutual TLS (mTLS)

This topic describes how to configure Service Registry to use mTLS.

1. Create the configuration file to include SSL properties.

   ```
   server:
     ssl:
       bundle: server
       client-auth: NEED

   spring:
     ssl:
       bundle:
         pem:
           server:
             keystore:
               certificate: samples/tls/server/tls.crt
               private-key: samples/tls/server/tls.key
   ```

```
          truststore:
            certificate: samples/tls/ca/tls.crt
```

2. Set `SPRING_CONFIG_ADDITIONAL_LOCATION` or `SPRING_CONFIG_IMPORT` to reference this configuration, as described in Installing Service Registry.
3. Run the application as an executable JAR file as described in Installing Service Registry:

```
java -jar tanzu-service-registry-1.0.0.jar
```

4. Test it by supplying certificates and keys in a request to the app's endpoint:

```
curl \
    --cacert samples/tls/ca/tls.crt \
    --cert samples/tls/client/tls.crt \
    --key samples/tls/client/tls.key \
    https://localhost:8761/eureka/apps
```

Running the Service Registry

This topic shows you how to run the Tanzu Service Registry.

1. Create an image from the Service Registry JAR file. The easiest way to do this is to use the `pack` command:

```
pack build tanzu/service-registry:1.0.0 \
  --path ./tanzu-service-registry-1.0.0.jar \
  --builder paketobuildpacks/builder:tiny
```

   - If you will be running the image on an ARM host (such as an Apple machine with an Apple chipset), you must use a different builder:

```
pack build tanzu/service-registry:1.0.0 \
  --path ./tanzu-service-registry-1.0.0.jar \
  --builder dashaun/builder:tiny
```

   - Alternatively, you can create an image using `docker build`. Create a Dockerfile with the following contents:

```
FROM openjdk:17-jdk
COPY tanzu-service-registry-1.0.0.jar sr.jar
ENTRYPOINT [ "java", "-jar", "sr.jar" ]
```

     This assumes that the JAR file is in the directory where you will create the image. Using the Docker CLI, create the image with this command (substitute "tanzu" with your organization's Docker repository name):

```
docker build -t "tanzu/service-registry:1.0.0" .
```

2. Optional. Create a configuration file. The Service Registry is packaged with minimal configuration. If you need to provide additional configuration, see Configuring the Service Registry.
3. Start the container by running:

```
docker run -d \
  -p 8761:8761 \
  tanzu/service-registry:1.0.0
```

This starts the container and forwards the local port 8761 to the Service Registry's port 8761 running in the container.

If you created a configuration file, you must make the configuration available to the container. The most basic way of doing this is to use a bind mount to mount a directory containing the configuration YAML file. For example, if the `service-registry.yml` file is in a directory name `srconfig`, start the container by running:

```
docker run -d \
  -p 8761:8761 \
  --mount type=bind,source="$(pwd)"/srconfig,target=/srconfig
  -e SPRING_CONFIG_IMPORT='/srconfig/service-registry.yml'
  tanzu/service-registry:1.0.0
```

In addition to starting the container and forwarding ports, this mounts the configuration from the local filesystem to the container's filesystem and sets the `SPRING_CONFIG_IMPORT` environment variable to reference the mounted configuration file.

4. Test it by opening the Service Registry dashboard in a browser. Alternatively, use `curl` to make a request to the apps endpoint:

```
curl localhost:8761/eureka/apps
```

Enabling Client Applications

This topic contains instructions for enabling client applications.

Spring Boot applications can use the Tanzu Service Registry by including the client dependency in their builds and configuring connection details to access the running Service Registry. This results in the client application registering itself with Service Registry at startup and periodically refreshing that registration to indicate to the Service Registry that it is still available. It also enables the application to discover other services by name from the Service Registry.

Adding the Client Dependency to your Build

To use a Tanzu Service Registry, a client app must include the necessary client dependency. Specifically, you must add the Spring Cloud OSS Eureka Discovery Client dependency to your project's build. In addition, you must add the Spring Cloud Bill of Materials (BOM) into the build's dependency management.

For Gradle builds

For a Gradle build, the Service Registry Client dependency looks like this:

```
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
```

The dependency management entry should look similar to this:

```
dependencyManagement {
  imports {
    mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersio
n}"
  }
}
```

Set the `springCloudVersion` property to reference the latest Spring Cloud OSS version:

```
ext {
  set('springCloudVersion', "2023.0.3")
}
```

For Maven builds

If your project is built with Maven, add the following dependency to the `<dependencies>` section of the build:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

The dependency management section should include the Spring Cloud BOM:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Set the spring-cloud.version property in the `<properties>` section like this:

```
<properties>
  <spring-cloud.version>2023.0.3</spring-cloud.version>
</properties>
```

Specifying Connection Details

The Eureka Discovery Client defaults to connecting to the Service Registry at localhost port 8761. If Service Registry is running locally and listening on port 8761, you do not need to provide connection information. But you should specify, at minimum, the application name. In this example, the name of your application is `my-application`.

```
spring.application.name=my-application
```

This is the name that the application will be registered under in the Service Registry.

If Service Registry is running on a different host and/or port, you must specify the location of the Service Registry in the configuration for your application:

```
eureka.client.service-url.defaultZone=http://my-eureka:9761/eureka
```

Enabling TLS (mTLS) Authentication

If Tanzu Service Registry requires TLS authentication, you can configure the client side certificates and trust store in `application.properties` with the following entries:

```
eureka.client.tls.enabled=true
eureka.client.tls.key-store=<path-to-key-store>
eureka.client.tls.key-store-type=PKCS12
eureka.client.tls.key-store-password=<key-store-password>
eureka.client.tls.key-password=<key-password>
eureka.client.tls.trust-store-type=PKCS12
eureka.client.tls.trust-store=<path-of-trust-store>
eureka.client.tls.trust-store-password=<trust-store-password>
```

Modify the values of these properties with the specific details for your client certification and trust store.

For more information about configuring the Spring Cloud Eureka Discovery Client, see the OSS Spring Cloud Service Discovery documentation.

VMware Tanzu Distribution of OpenJDK

VMware Tanzu OpenJDK

Broadcom distributes the BellSoft Liberica distribution of OpenJDK® under the name VMware Tanzu Distribution of OpenJDK.

Installation

BellSoft Liberica is available on Broadcom Support portal.

To install it:

1. Download the latest binary from Broadcom Support portal.
2. Extract the compressed file into that directory using the following command:

   ```
   $ tar xf bellsoft-jdk11.0.7+10-linux-amd64.tar.gz
   ```

3. Add this version of Java to your PATH:

   ```
   $ export PATH=$PWD/jdk-11.0.7/bin:$PATH
   ```

4. Verify that the Java version is correct:

   ```
   $ java -version
   ```

Support Lifecycle

The following table shows the end of support dates for the currently supported JDKs:

Version	End of Support
OpenJDK 21	March 2032
OpenJDK 17	March 2030
OpenJDK 11	March 2027
OpenJDK 8	March 2031

VMware Tanzu OpenJDK end of support dates follow those set by BellSoft Liberica. For more information about the BellSoft support lifecycle, see the Liberica JDK Support Roadmap.

Java and OpenJDK are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Spring Enterprise Subscription

Spring Enterprise Subscription provides access to supported releases of Spring Boot that are no longer under OSS support and are within the “Enterprise” support window.

The following guides can help you get access to the artifact repository where Enterprise supported releases of Spring Boot (and other Spring projects that may be made available in the future) are stored:

- Guide for Artifact Repository Administrators
- Guide for Application Developers

For more information about the specific OSS and Enterprise release support dates for Spring Boot, go to the Spring Boot project support page. Release notes, reference documentation, and Javadoc for Spring Enterprise Subscription releases are indexed at enterprise.spring.io. Release-specific documentation for many Spring releases is also published in a docs zip file within the repository tree.

Spring Enterprise Subscription for Artifact Repository Administrators

This guide walks you through how artifact repository administrators can synchronize Spring Enterprise supported releases from https://packages.broadcom.com/artifactory/spring-enterprise to their internal artifact repository. After the Spring Enterprise artifacts are synced, application development teams with access to your internal artifact repository can include them as dependencies in their projects.

Prerequisites

Access to the Spring Enterprise Subscription artifact repository is available to entitled customers through the Broadcom Customer Support Portal with at least one entitlement in:

- VMware Tanzu Spring
- VMware Tanzu Platform for Cloud Foundry (formerly Tanzu Application Service)
- VMware Tanzu Platform for Kubernetes

Additional Prerequisites for Air-Gapped Environments

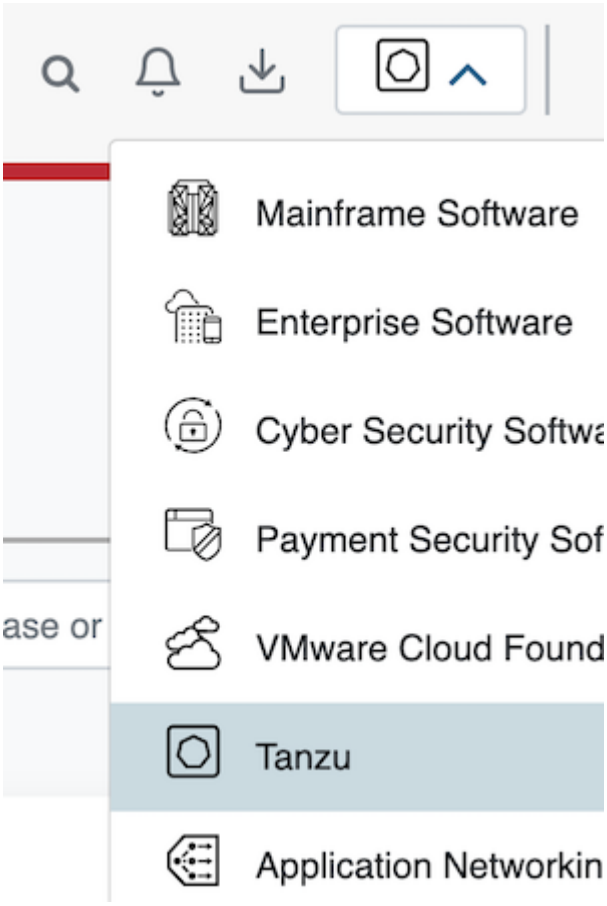
For air-gapped environments in which external access is tightly controlled, you must add the following domains to your allow list:

- packages.broadcom.com
- JFrog Artifactory domains

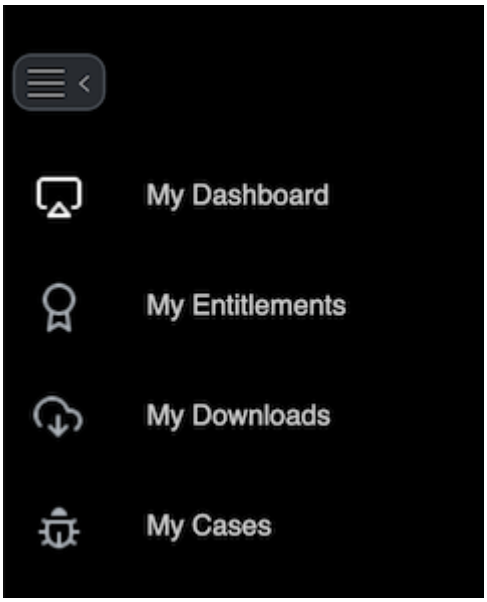
Accessing Spring Enterprise Subscription Artifact Repositories

To access the Spring Enterprise Subscription artifact repository:

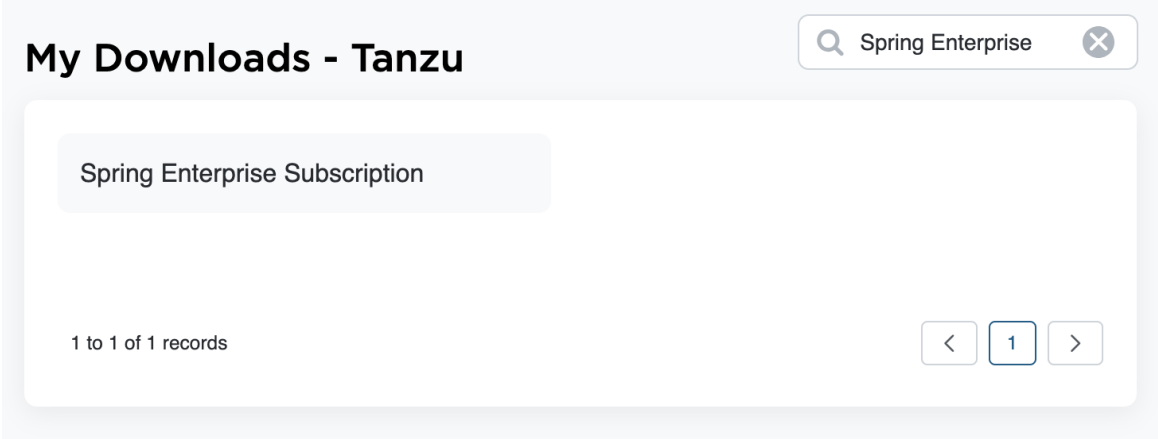
1. Log in to the Broadcom Customer Support Portal.
2. Ensure that **Tanzu** is selected in the top navigation bar of the site.



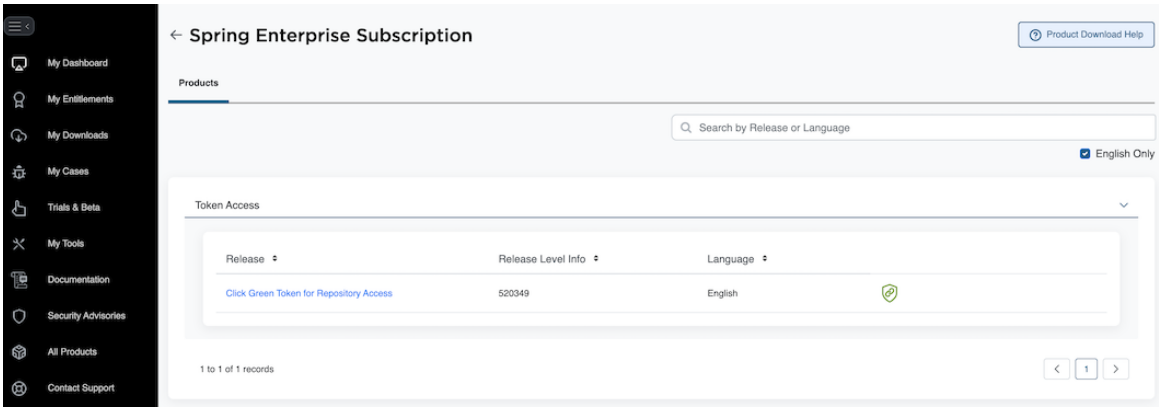
3. On the left side navigation, select **My Downloads**.



4. On the **My Downloads** page, search for “Spring Enterprise.”



5. Retrieve the access token by expanding the **Token Access** row and clicking on the green icon, after which you are presented with an instructions pop-up.



   The download instructions walk you through how to save your access token, along with steps for setting up your own artifact repository to sync the artifacts from Spring Enterprise Subscription. The access token is valid for 6 months by default.

6. Save your access token value in a local file. Click the **Save File** button on the instructions pop-up and select the save location. This access token serves as a password for your email address or as a bearer token for API requests. When authenticating to Spring Enterprise Subscription artifact repository URL directly, use the value `access_token` in the saved JSON file.

Spring Enterprise Subscription repository details

- URL: https://packages.broadcom.com/artifactory/spring-enterprise/
- Type: Maven (or Gradle)
- Description: Repository GA Spring Enterprise Releases

Adding a Remote Repository in Artifactory

JFrog Artifactory and Sonatype Nexus are two commonly used artifact repository managers. The following sections describe how to add `Release` artifacts to your local Artifactory server.

- Reference: https://jfrog.com/help/r/jfrog-artifactory-documentation/remote-repositories
- Permissions: Artifactory Platform Administrator access required

To configure a remote repository:

1. Go to the **Administration** module.
2. Go to **Repositories** and select **Add Repositories**, then select **Remote Repository**.
3. On that page, select type **Maven**, and provide the configuration values listed in the following sections.

Configuring Release Artifacts

The following configuration values can be used to cache the Spring Enterprise Subscription release artifacts in Artifactory:

1. Configure:
    - **Repository Key**: spring-enterprise-mvn-remote (or preferred on-premise naming convention)
    - **URL**: https://packages.broadcom.com/artifactory/spring-enterprise
    - **User Name**: email address for this account
    - **Password** / **Access Token**: the value with the save Access Token file for attribute `access_token`
2. Click **Test** to confirm that the credentials are working.
    - **Repository Layout**: maven-2-default
    - **Remote Layout Mapping**: maven-2-default
3. Ensure that **Handle Releases** is selected.

Maven versus Gradle

The Spring Enterprise artifact repository is of type `Maven` but can be used by both Maven and Gradle clients, as the Spring team does today.

Artifactory Smart Repository

Artifactory platform administrators might see an alert saying, “Artifactory Smart Repository Detected” when testing connectivity of a new remote repository to packages.broadcom.com.

If you want to assist the Spring team by delivering a limited set of repository statistics back upstream to packages.broadcom.com, you can enable all settings for this feature, but this is not required.

Advanced Settings

Advanced settings for the remote repository depend largely on individual platform requirements for storage, network, local compliance and regulations, and so on. These settings are outside the scope of Spring Enterprise Subscription support.

When storage is a concern, the Spring Team generally sets a non-zero value for **Unused Artifacts Cleanup Period (Hr)** for remote snapshot and other repositories.

Downstream Repository Replications

Pull replication is a convenient way to proactively populate a remote cache. This avoids waiting for artifacts to arrive when first requested, reduces traffic on the Spring Enterprise server, and is permitted from packages.broadcom.com. The ability to configure downstream repository replication is available in the JFrog Artifactory commercial offering and Sonatype Nexus Proxy Repository.

Reference Documentation

Release notes, reference documentation, and Javadoc for Spring Enterprise Subscription releases are available at enterprise.spring.io. Release-specific documentation for many Spring releases is also published in a docs zip file in the repository tree.

Spring Enterprise Subscription for Application Developers

This guide describes how application developers can access Spring Enterprise releases directly from Spring Enterprise Subscription artifact repositories when developing their applications.

Prerequisites

You must retrieve an access token to use Spring Enterprise Subscription repository artifacts during build execution. Follow the steps in Accessing Spring Enterprise Subscription Artifact Repositories to retrieve your access token.

Using Spring Enterprise Artifacts

Follow the steps below to configure your Maven and Gradle build environments to access Spring Enterprise Subscription artifacts during build execution. There are several ways to do this.

Option 1: Create a shared Maven profile for all Maven and Gradle projects

Using this approach prevents modification of the configuration of any of your existing Maven and Gradle repositories.

This configuration will be especially useful when you want to upgrade your Spring applications using the commercial OpenRewrite recipes with Spring Application Advisor.

Add the following to your `$HOME/.m2/settings.xml` for the environment in which you are executing these builds, and replace `[email]` and `[access_token]` with your user name for Spring Enterprise Subscription and your valid access token, respectively:

```
<servers>
  <server>
    <id>spring-enterprise-subscription</id>
    <username>[email]</username>
    <password>[access_token]</password>
  </server>
</servers>
<profiles>
  <profile>
    <id>spring-enterprise</id>
    <activation>
```

```
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>spring-enterprise-subscription</id>
          <url>https://packages.broadcom.com/artifactory/spring-enterprise</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>spring-enterprise-subscription</id>
          <url>https://packages.broadcom.com/artifactory/spring-enterprise</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
   <activeProfile>spring-enterprise</activeProfile>
 </activeProfiles>
```

If you do not know if your Gradle repositories are using `mavenLocal()` to load the local Maven settings, VMware recommends storing the following configuration for Gradle in `$HOME/.gradle/init.d/init.gradle`:

```
apply plugin: SpringEnterpriseRepositoryPlugin

class SpringEnterpriseRepositoryPlugin implements Plugin<Gradle> {

    void apply(Gradle gradle) {
        gradle.allprojects { project ->
            project.repositories {
                // add the Spring enterprise repository
                maven {
                    name "SPRING_ENTERPRISE_REPO"
                    url "https://packages.broadcom.com/artifactory/spring-enterprise"

                    credentials {
                      username "[email]"
                      password "[password]"
                    }
                    authentication {
                      basic(BasicAuthentication)
                    }
                }
            }
        }
    }
}
```

Option 2: Configure a single Maven repository

To track changes in the Maven repositories, consider doing the following:

1. Add the credentials for the Spring Enterprise Subscription artifact repository in your `$HOME/.m2/settings.xml`. This is an example:

```
  <servers>
    <server>
      <id>spring-enterprise-subscription</id>
      <username>[email]</username>
      <password>[access_token]</password>
    </server>
  </servers>
```

Replace `[email]` and `[access_token]` with your user name for Spring Enterprise Subscription and your valid access token, respectively.

2. Define new Maven repositories in the `pom.xml` of the repository consumers.

```
...
      <repositories>
        <repository>
          <id>spring-enterprise-subscription</id>
          <url>https://packages.broadcom.com/artifactory/spring-enterprise</url
>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>spring-enterprise-subscription</id>
          <url>https://packages.broadcom.com/artifactory/spring-enterprise</url
>
        </pluginRepository>
      </pluginRepositories>
...
```

Option 3: Configure a single Gradle repository

To configure the Spring Enterprise Subscription artifact repository for Gradle, the build environment must be configured with access as both a dependency repository and a plug-in repository.

Add the following to your `settings.gradle` for the environment in which you are executing these builds, and replace `[email]` and `[access_token]` with your user name for Spring Enterprise Subscription and your valid access token, respectively.

```
pluginManagement {
  repositories {
    maven {
      url "https://packages.broadcom.com/artifactory/spring-enterprise"
      credentials {
        username "[email]"
        password "[access_token]"
      }
      authentication {
        basic(BasicAuthentication)
      }
    }
    mavenCentral()
  }
}

dependencyResolutionManagement {
```

```
  repositories {
    maven {
      url "https://packages.broadcom.com/artifactory/spring-enterprise"
      credentials {
        username “[email]“
        password “[access_token]”
      }
      authentication {
        basic(BasicAuthentication)
      }
    }
    mavenCentral()
  }
}
```

Option 4: Create a remote Maven repository for the Spring Enterprise Subscription artifact repository

If you have an enterprise Maven repository, you can configure it to work as a proxy. For configuration instructions, see Adding a Remote Repository in Artifactory. In this case, there will be no changes to any of your configurations, and the Spring Enterprise supported artifacts will be available for the enterprise.