

Spotfire Service for Python Installation and Administration

Software Release 1.22.0

Contents

| | |
|--|-----------|
| The Spotfire® Service for Python..... | 4 |
| System Requirements..... | 4 |
| Limiting exposure of your deployment..... | 4 |
| Containerized service..... | 6 |
| Configuring a custom Docker image on a node with internet access..... | 7 |
| Configuring a custom Docker image on a node with no internet access..... | 8 |
| Configuring a custom startup script to build a custom Docker image..... | 11 |
| Pulling a custom Docker image from an authenticated repository..... | 13 |
| Installing the service on a node manager for a Spotfire Server..... | 16 |
| Configuring the Service..... | 18 |
| Custom configuration properties..... | 19 |
| Allowed engines..... | 19 |
| Compressed job contents and results..... | 20 |
| Custom Python interpreter..... | 20 |
| Disable warnings..... | 21 |
| Docker container built for no internet access..... | 21 |
| Engine ports..... | 21 |
| Engine pruning..... | 22 |
| Engine timeout..... | 22 |
| File size upload limit..... | 22 |
| Logging level..... | 23 |
| Manage Java options..... | 23 |
| Package library location..... | 24 |
| Safeguarding your environment..... | 25 |
| Startup script..... | 25 |
| JMX monitoring..... | 26 |
| Containerized configuration..... | 26 |
| Package management for the Spotfire Service for Python..... | 27 |
| Find help..... | 28 |
| Using an alternative Python package repository..... | 28 |
| Included packages..... | 29 |
| The Spotfire Package (SPK)..... | 32 |
| SPK versioning..... | 32 |

| | |
|--|-----------|
| Distribute Python packages..... | 33 |
| Installing Python packages manually..... | 35 |
| Creating a Spotfire package for Python packages on the node manager..... | 36 |
| Creating a Spotfire package for Python packages from a Windows computer..... | 39 |
| Build a Spotfire package for Spotfire Service for Python (Linux)..... | 42 |
| Use an alternative Python interpreter..... | 44 |
| Creating a Spotfire package for an alternative Python interpreter (for Windows)..... | 45 |
| Build a suitable Python interpreter for Spotfire Service for Python (for Linux)..... | 47 |
| Service resource management scenarios..... | 49 |
| Service logs..... | 51 |
| Monitoring the service using JMX..... | 52 |
| Troubleshooting the service..... | 54 |
| Spotfire Documentation and Support Services..... | 56 |
| Legal and Third-Party Notices..... | 57 |
| Index..... | 61 |

The Spotfire® Service for Python

The Spotfire® Service for Python extends access to the Python language from the installed Spotfire client to Spotfire web client users.

The Spotfire® Service for Python provides predictive and computational detail to users who access Spotfire analyses through the Spotfire web client.



In Spotfire 14.5 and later, the Spotfire Service for Python is **required** if you want web client users to be able to use built-in data functions from the Spotfire Data Science application. Because some of the calculations in the built-in data functions can take some time to finish, you might also need to adjust the **engine timeout** settings for the service in this case.

The Spotfire Service for Python is provided to Spotfire® Server administrators as a set of Spotfire SPK packages in one SDN distribution file. Spotfire Server administrators can install and configure the Spotfire Service for Python on a node manager that is available to the Spotfire Server.

This document is meant for Spotfire Server administrators who need to install and configure the Spotfire Service for Python, to install Python packages for data functions and analyses that use them, and to review logs and troubleshoot the Spotfire Service for Python.

System Requirements

Your deployment must meet certain requirements to run the Spotfire Service for Python.

See the [system requirements](#) for this version of the Spotfire Service for Python.

Limiting exposure of your deployment

The Spotfire Service for Python is installed on a Spotfire Server node running under Linux or Windows. The Linux installation provides the option of running the Spotfire Service for Python in a containerization platform.

When you install the Spotfire Service for Python and run the Python engine, you can take steps to protect the server deployment, to minimize the risk of unauthorized access, and to minimize the possibility of malicious acts.

Statistical engines such as Python provide functions to access data and packages on the internet. Additionally, they have functions that access the host computer system, such as those for executing system commands, and those for reading and writing files. By their very design, these languages can expose computer systems to risk from bad actors, unless the deployer takes steps to secure the environments in which they run. We strongly recommend reviewing and implementing the practices described here.



The Spotfire Service for Python installed on a Spotfire Server node running under Windows does not have a containerized installation available.

Restricting user access

- Run the Spotfire Service for Python using an account that limits network access to required external data sources and services only. (Note that taking this step can limit availability to data and package updates.)
- Always run the node manager containing the Spotfire Service for Python as a non-root user. (That is, not as root or under an Administrative account.)
- If you are running a system where other servers have access to computers running the Spotfire Service for Python, disable passwordless access between the server and other servers.

Configuring for tighter engine control

- If your deployment is on a Linux server, then the default configuration for the Spotfire Service for Python is to use containers (the property `use.engine.containers: TRUE`). Running the Spotfire Service for Python with containers enabled prevents the engines from having access to the host system. See [Containerized service](#) on page 6 for more information.



Docker is available under separate software license terms and is not part of the Spotfire Server or the Spotfire Service for Python. As such, Docker is not within the scope of your license for Spotfire Server or the Spotfire Service for Python. Docker is not supported, maintained, or warranted in any way by Cloud Software Group, Inc. Download and use of Docker is solely at your own discretion and subject to license terms applicable to Docker.

Containerized service

When you install the Spotfire Service for Python on a Linux computer, by default, it is configured to use a Docker® container. A containerized Spotfire Service for Python is available only for Linux installations.

To use a containerized Spotfire Service for Python on a Linux system, download and install Docker.

- If you have not yet installed the Spotfire Service for Python, install Docker first, and then install the Spotfire Service for Python.
- If you have already installed the Spotfire Service for Python before installing Docker, then stop the Spotfire Service for Python, install Docker, set the configuration to use Docker, and then restart the Spotfire Service for Python.

The version of Docker you use depends on your Linux system. See the [system requirements](#) for the recommended version. See www.docker.com for more information about Docker.

The primary benefit of installing and using the Spotfire Service for Python in a container is that it operates the service in a "sandbox", so the Python engine does not have access to the host file system. (For more information about script and data function trust, see the [Spotfire client user guide](#)). Running the Spotfire Service for Python in a container results in negligible performance impact.



Containerization of the engines does not, by default, limit access to the network. If your system supports untrusted or public users creating data functions, consider additional firewall configuration on the host system to limit container exposure to the network or internet to only necessary sites and servers. Consult your OS or Docker documentation for further guidance.

The only container framework with which we developed and tested the Spotfire Service for Python is Docker. We do not provide Docker with the base installation; however, you must have Docker installed for the Spotfire Service for Python to work properly. The service downloads and builds a default Docker image based on `debian:12-slim` from Docker Hub. While you cannot modify the image we provide, you can build and use a different Docker image if you have different configuration requirements. This section contains a few examples of specifying different Docker images. Alternatively, check Docker Hub for an image that might work for you.



Docker is available under separate software license terms and is not part of the Spotfire Server or the Spotfire Service for Python. As such, Docker is not within the scope of your license for Spotfire Server or the Spotfire Service for Python. Docker is not supported, maintained, or warranted in any way by Cloud Software Group, Inc. Download and use of Docker is solely at your own discretion and subject to license terms applicable to Docker.

You can export and change the configuration options to build and install a customized image. See [Configuring the Service](#) on page 18 for more information. The properties to change in the configuration are the following:

- `docker.image.name`
- `startup.hook.script`



Important After you install Docker, you must create the **docker** group and then add the **spotfire** user to the docker group. For more information about setting up this group, see <https://docs.docker.com/engine/install/linux-postinstall/>.

After you install and configure both the container and the Spotfire Service for Python, you can start the service. When the service starts, containers are created as needed.



Optionally, you can set the configuration option `use.engine.containers` to `FALSE` to disable the container option.

Configuring a custom Docker image on a node with internet access

If you have access to the internet, then you can build a Docker image for your Spotfire Service for Python, referencing it by its name and tag.

Perform this task from the command line on the computer where your Spotfire Server is installed. Some steps are performed on the computer where the node manager is installed. (This can be a different computer.)

You can create the custom configuration before installing the Spotfire Service for Python.

Prerequisites

- You must have Docker installed on the computer running the node manager. If you install and start the service before you install Docker, then exceptions are written to the log.
- You must have a Linux computer where the node manager is installed. (Your node manager and the Spotfire Server are usually on different computers).
- If you are using the script to build the base Docker image, you must have a connection to the internet. (A connection to the internet is not required if you are using a locally-available Docker image.)

Procedure

1. If you have already installed the service from the Spotfire Server **Nodes & Services** administration page, and if it is running, then stop the service.
2. On the node manager computer, create a new directory called `docker`. Inside that directory, create your Dockerfile.



Important Remember that for any script you write, the line endings must be appropriate for the operating system where the service runs. Many text editors can perform end-of-line (EOL) conversion.

```
#####
# A sample Dockerfile for installing JDK. #
#####
FROM debian:12-slim
RUN apt update -y \
    && apt upgrade -y \
    && apt install -y \
    ca-certificates \
    && apt install -y --no-install-recommends \
    && apt clean all
# install jdk
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y wget apt-transport-https gpg \
    && wget -qO - https://packages.adoptium.net/artifactory/api/gpg/key/public | gpg
    --dearmor | tee /etc/apt/trusted.gpg.d/adoptium.gpg > /dev/null \
    && echo "deb https://packages.adoptium.net/artifactory/deb $(awk -F= '/
^VERSION_CODENAME/{print$2}' /etc/os-release) main" | tee /etc/apt/sources.list.d/
adoptium.list \
    && DEBIAN_FRONTEND=noninteractive apt-get update \
    && DEBIAN_FRONTEND=noninteractive apt-get install -y temurin-21-jdk \
    && apt-get clean all

# set JAVA_HOME variable
ENV JAVA_HOME=/usr/lib/jvm/temurin-21-jdk-amd64
```

For more information, see <https://docs.docker.com/engine/reference/builder/>.

3. Install Python and pip.

```
#install the Debian-included Python and the pip package.
RUN apt-get update && apt-get install -y python-is-python3 python3 python3-pip && apt-get
clean
```

4. Install the spotfire package.

```
RUN pip install spotfire --break-system-packages
```

5. Update the following in the Dockerfile.
 - a) Install a compatible Python interpreter. See [Custom Python interpreter](#) on page 20 for more information.
 - b) Set the environment variable `SPOTFIRE_PYTHON_HOME` to the path to the directory for the installed Python interpreter that you want to use.
 - c) Using `pip`, install the spotfire Python package (available at <https://pypi.org/project/spotfire/>).
6. On the computer running the node manager, build the image with the name and tag.
The name and tag are comprised as `<name:version>`, as follows:

```
docker build -t pysrv:258 .
```

For more information, see <https://docs.docker.com/engine/reference/commandline/build/>.

7. On the computer running the Spotfire Server, export the service configuration file `custom.properties`:

```
config export-service-config --capability=PYTHON --deployment-area=<your deployment area name>
```

See the [export-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

8. Edit the settings in the `custom.properties` file, specifying the name and tag of your custom image.

```
use.engine.containers: TRUE
docker.image.name: <name:version>
```

9. On the computer running the Spotfire Server, import the service configuration:

```
config import-service-config --config-name=<new-config-name>
```

See the [import-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

10. From the Spotfire Server **Nodes & Services** administration page, install the service, specifying the configuration to use, and then start the service.

If you have already installed the service, then, under the node manager, select the service and click **Edit**. From the **Configuration** drop-down list, select the new configuration.

If problems occur, troubleshoot by examining the Dockerfile that the service writes. After the service runs, this Dockerfile is available at the root service directory on the computer running the node manager. For example, `/opt/nodemanager/<version>/nm/services/<language>-service-linux-<version_#_ID>/dockerfile/Dockerfile`.

Configuring a custom Docker image on a node with no internet access

If your node manager does not have external access to the internet, then you can create a Docker image on an internet-enabled computer, and then transfer it to your node manager.

Perform the first three steps of this task from the command line on a computer with internet access. Perform the rest of the task from the command line on the computer where your node manager is installed.

Prerequisites

- If you are using the script to build the base Docker image, you must have a connection to the internet. (A connection to the internet is not required if you are using a locally-available Docker image.)
- You must have a Linux computer where the node manager is installed. (Your node manager and the Spotfire Server are usually on different computers).
- You must have Docker installed on the computer running the node manager. If you install and start the service before you install Docker, then exceptions are written to the log.

Custom docker images for the service must contain the following.

- The Eclipse Temurin 21 Java Runtime.
- The JAVA_HOME environmental variable, correctly defined.

```
ENV JAVA_HOME=</correct/path/to/java>
```

Procedure

1. On a computer with internet access, create the Dockerfile.



Important Remember that for any script you write, the line endings must be appropriate for the operating system where the service runs. Many text editors can perform end-of-line (EOL) conversion.

```
#####
# A sample Dockerfile for installing JDK. #
#####
FROM debian:12-slim
RUN apt update -y \
    && apt upgrade -y \
    && apt install -y \
        ca-certificates \
    && apt install -y --no-install-recommends \
    && apt clean all
# install jdk
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y wget apt-transport-https gpg \
    && wget -qO - https://packages.adoptium.net/artifactory/api/gpg/key/public | gpg
    --dearmor | tee /etc/apt/trusted.gpg.d/adoptium.gpg > /dev/null \
    && echo "deb https://packages.adoptium.net/artifactory/deb $(awk -F= '/
^VERSION_CODENAME/{print$2}' /etc/os-release) main" | tee /etc/apt/sources.list.d/
    adoptium.list \
    && DEBIAN_FRONTEND=noninteractive apt-get update \
    && DEBIAN_FRONTEND=noninteractive apt-get install -y temurin-21-jdk \
    && apt-get clean all

# set JAVA_HOME variable
ENV JAVA_HOME=/usr/lib/jvm/temurin-21-jdk-amd64
```

For more information, see <https://docs.docker.com/engine/reference/builder/>.

2. Install Python and pip.

```
#install the Debian-included Python and the pip package.
RUN apt-get update && apt-get install -y python-is-python3 python3 python3-pip && apt-get
clean
```

3. Install the spotfire package.

```
RUN pip install spotfire --break-system-packages
```

4. Optional: If you need or want to use a Python interpreter that is not bundled with Spotfire Service for Python, then update the following in the `Dockerfile`.
 - a) Install a compatible Python interpreter. See [Custom Python interpreter](#) on page 20 for more information.
 - b) Set the environment variable `SPOTFIRE_PYTHON_HOME` to the path to the directory for the installed Python interpreter that you want to use.
 - c) Using `pip`, install the spotfire Python package (available at <https://pypi.org/project/spotfire/>).

5. Build the image specifying the name and tag.

Use the command `docker build -t <name:version>`, as follows:

```
docker build -t pysrv:258 .
```

For more information, see <https://docs.docker.com/engine/reference/commandline/build/>.

6. Save the image to a `.tar` file.

Use the command `docker save -o <name-version>.tar <name:version>`, as follows:

```
docker save -o pysrv-258.tar pysrv:258
```

For more information, see <https://docs.docker.com/engine/reference/commandline/save/>.

7. If you have already installed the service from the Spotfire Server **Nodes & Services** administration page, and if it is running, then stop the service.
8. Transfer the `.tar` file to the target computer (where the node manager is running).
9. Load the `.tar` file into the node manager.

Use the command `docker load -i <name-version>.tar`, as follows:

```
$ sudo docker load -i pysrv-258.tar
f2419d350464: Loading layer [=====]
329.5MB/329.5MB
Loaded image: pysrv:258
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pysrv                258                9941b68e7f65       17 hours ago       517MB
```

For more information, see <https://docs.docker.com/engine/reference/commandline/load/>.

10. On the computer running the Spotfire Server, export the service configuration file `custom.properties`:

```
config export-service-config --capability=PYTHON --deployment-area=<your deployment area name>
```

See the [export-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

11. Edit the settings in the `custom.properties` file, specifying the name and tag of your custom image, and indicating that the docker image identifier should be used directly, without building an image on top of it.

```
use.engine.containers: TRUE
docker.image.name: <name:version>
use.immutable.container=true
```

See [Docker container built for no internet access](#) on page 21 for more information.

12. On the computer running the Spotfire Server, import the service configuration:

```
config import-service-config --config-name=<new-config-name>
```

See the [import-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

13. From the Spotfire Server **Nodes & Services** administration page, install the service, specifying the configuration to use, and then start the service.

If you have already installed the service, then, under the node manager, select the service and click **Edit**. From the **Configuration** drop-down list, select the new configuration.

See [Installing the service on a node manager for a Spotfire Server](#) on page 16 and [Configuring the Service](#) on page 18 for more information.

Configuring a custom startup script to build a custom Docker image

If you have access to the internet, then you can build a custom Docker image for your Spotfire Service for Python.

Perform this task from the command line on the computer where the Spotfire Service for Python is installed, and on the computer where your node manager is installed. For more information about the startup script, see [Startup script](#) on page 25.

Prerequisites

- You must have Docker installed on the computer running the node manager. If you install and start the service before you install Docker, then exceptions are written to the log.
- You must have a Linux computer where the node manager is installed. (Your node manager and the Spotfire Server are usually on different computers).
- If you are using the script to build the base Docker image, you must have a connection to the internet. (A connection to the internet is not required if you are using a locally-available Docker image.)

Custom docker images for the service must contain the following.

- The Eclipse Temurin 21 Java Runtime.
- The JAVA_HOME environmental variable, correctly defined.

```
ENV JAVA_HOME=</correct/path/to/java>
```

Procedure

1. If you have already installed the service from the Spotfire Server **Nodes & Services** administration page, and if it is running, then stop the service.
2. On the computer running the Spotfire Server, export the service configuration file `custom.properties`:

```
config export-service-config --capability=PYTHON --deployment-area=<your deployment area name>
```

See the [export-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

3. On computer running Spotfire Server, create a file called Dockerfile, and then save it to your custom configuration directory.



Important Remember that for any script you write, the line endings must be appropriate for the operating system where the service runs. Many text editors can perform end-of-line (EOL) conversion.

```
#####
# A sample Dockerfile for installing JDK. #
#####
FROM debian:12-slim
RUN apt update -y \
  && apt upgrade -y \
  && apt install -y \
  ca-certificates \
  && apt install -y --no-install-recommends \
  && apt clean all
# install jdk
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y wget apt-transport-https gpg \
  && wget -qO - https://packages.adoptium.net/artifactory/api/gpg/key/public | gpg
  --dearmor | tee /etc/apt/trusted.gpg.d/adoptium.gpg > /dev/null \
  && echo "deb https://packages.adoptium.net/artifactory/deb $(awk -F= '/
^VERSION_CODENAME/{print$2}' /etc/os-release) main" | tee /etc/apt/sources.list.d/
  adoptium.list \
  && DEBIAN_FRONTEND=noninteractive apt-get update \
  && DEBIAN_FRONTEND=noninteractive apt-get install -y temurin-21-jdk \
  && apt-get clean all

# set JAVA_HOME variable
ENV JAVA_HOME=/usr/lib/jvm/temurin-21-jdk-amd64
```

For more information, see <https://docs.docker.com/engine/reference/builder/>.

4. Install Python and pip.

```
#install the Debian-included Python and the pip package.
RUN apt-get update && apt-get install -y python-is-python3 python3 python3-pip && apt-get
  clean
```

5. Install the spotfire package.

```
RUN pip install spotfire --break-system-packages
```

6. Optional: If you need or want to use a Python interpreter that is not bundled with Spotfire Service for Python, then update the following in the Dockerfile.
 - a) Install a compatible Python interpreter. See [Custom Python interpreter](#) on page 20 for more information.
 - b) Set the environment variable `SPOTFIRE_PYTHON_HOME` to the path to the directory for the installed Python interpreter that you want to use.
 - c) Using `pip`, install the spotfire Python package (available at <https://pypi.org/project/spotfire/>).
7. On the computer running the service, create a custom script to build the Dockerfile, and then save it to your custom configuration directory.
The following example file is named `customScript.sh`.

```
#!/bin/bash

# Define the image name and tag
IMAGE_NAME="pysrv:customScript"
# Custom configuration files are at relative path conf/FILE
DOCKERFILE_NAME="conf/Dockerfile"
# Command to check if image exists
COMMAND="docker inspect ${IMAGE_NAME}"

# Run the command then check the status code
$COMMAND
RESULT=$?
if [ $RESULT -ne 0 ]; then
  # Image did not exist
```

```

echo ${IMAGE_NAME} does not exist. Building now...
COMMAND="docker build -f ${DOCKERFILE_NAME} -t ${IMAGE_NAME} ."
echo ${COMMAND}
echo "Building the custom docker image ${IMAGE_NAME} for the python-
service"
$COMMAND
echo "Completed building ${IMAGE_NAME}"
else
# Image exists already
echo The requested image ${IMAGE_NAME} already exists.
fi

```

8. Edit the relevant properties in the `custom.properties` file, specifying using the custom script.

```

use.engine.containers: TRUE
docker.image.name: pysrv:customScript
startup.hook.script: conf/customScript.sh

```

9. On the computer running the Spotfire Server, import the service configuration:

```
config import-service-config --config-name=<new-config-name>
```

See the [import-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

10. From the Spotfire Server **Nodes & Services** administration page, install the service, specifying the configuration to use, and then start the service.

If you have already installed the service, then, under the node manager, select the service and click **Edit**. From the **Configuration** drop-down list, select the new configuration.

If problems occur, troubleshoot by examining the `Dockerfile` that the service writes. After the service runs, this `Dockerfile` is available at the root service directory on the computer running the node manager. For example, `/opt/nodemanager/<version>/nm/services/<language>-service-linux-<version_#_ID>/dockerfile/Dockerfile`.

Pulling a custom Docker image from an authenticated repository

You can create a custom start script to configure the Spotfire Service for Python to log in to a remote authenticated repository and pull a custom Docker image.

This option is available if you want to specify a base image for the docker container, but it is in a repository that requires authentication to access. To set the appropriate authentication credentials, you can execute a Docker login command when you start the service, but before starting the Docker container, as part of a startup hook script.

This task demonstrates accessing a Docker image stored in the AWS Elastic container Registry, which is an authenticated repository.

Prerequisites

- You must have a Linux computer where the node manager is installed. (Your node manager and Spotfire Server are usually on different computers).

Custom docker images for the service must contain the following.

- The Eclipse Temurin 21 Java Runtime.
- The `JAVA_HOME` environmental variable, correctly defined.

```
ENV JAVA_HOME=</correct/path/to/java>
```

Procedure

1. If you have already installed the service from the Spotfire Server **Nodes & Services** administration page, and if it is running, then stop the service.
2. Install the AWS command-line interface (CLI) tool on the computer running the node manager.
See <https://docs.aws.amazon.com/cli/latest/userguide/awscli-install-bundle.html> for more information.
 - a) Run the command `aws configure`, and then connect to your account using your AWS Access Key and AWS Secret Access Key.
 - b) Verify that the user running the Spotfire Service for Python can run the `aws` process.
3. Determine your `docker.image.name` property.

- a) In your AWS account, navigate to **Amazon ECR > Repositories**.

The `docker.image.name` is listed after `Repository URI`, and the tag is listed after `Image Tags`.

```
Repository URI 123456.dkr.ecr.us-west-2.amazonaws.com/python/pysrv-sample
Image Tags: latest
```

The `docker.image.name` property is a concatenation of those two values.

```
docker.image.name: 123456.dkr.ecr.us-west-2.amazonaws.com/python/pysrv-sample:latest
```

4. On the computer running the Spotfire Server, export the service configuration file `custom.properties`:

```
config export-service-config --capability=PYTHON --deployment-area=<your deployment area name>
```

See the [export-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

5. On the computer running the Spotfire Server, create a custom script and save it to your custom configuration directory `<server installation dir>/tomcat/spotfire-bin/config/root/conf/`.

The script uses the AWS `get-login` command to fetch the `docker login` command. See the following links for more information:

- <https://docs.aws.amazon.com/cli/latest/reference/ecr/get-login.html>
- <https://docs.docker.com/engine/reference/commandline/login/>

In the script, use the absolute path to the `aws` command (`usr/local/bin/aws`).

We named this sample script `awsScript.sh`.

If saved to a custom configuration, it resides at the relative path `conf/awsScript.sh`.



Important Remember that for any script you write, the line endings must be appropriate for the operating system where the service runs. Many text editors can perform end-of-line (EOL) conversion.

```
#!/bin/bash
# Request a login from AWS
# The command will return a 'docker login' string
DOCKER_LOGIN=`/usr/local/bin/aws ecr get-login --no-include-email --region us-west-2`
echo Retrieved the command ${DOCKER_LOGIN}
# Execute that 'docker login'
${DOCKER_LOGIN}
echo docker login authentication completed.
```

6. From the command line, manually test your script at this stage to ensure that everything works correctly.

7. Edit the relevant properties in the file `custom.properties` with the appropriate values.

```
docker.image.name: 123456.dkr.ecr.us-west-2.amazonaws.com/python/pysrv-sample:latest
use.engine.containers: TRUE
startup.hook.script: conf/awsScript.sh
```

8. From the command line, manually test the script again to make sure that it works correctly.
9. On the computer running the Spotfire Server, import the service configuration:

```
config import-service-config --config-name=<new-config-name>
```

See the [import-service-config](#) page in the *Spotfire® Server and Environment - Installation and Administration* guide for more information.

10. From the Spotfire Server **Nodes & Services** administration page, install the service, specifying the configuration to use, and then start the service.

If you have already installed the service, then, under the node manager, select the service and click **Edit**. From the **Configuration** drop-down list, select the new configuration.

What to do next

If problems occur, troubleshoot by examining the `Dockerfile` that the service writes. After the service runs, this `Dockerfile` is available at the root service directory on the computer running the node manager. For example, `/opt/nodemanager/<version>/nm/services/<language>-service-linux-<version_#_ID>/dockerfile/Dockerfile`.

Installing the service on a node manager for a Spotfire Server

After installing and authorizing a node manager, you can install the Spotfire Service for Python.

Prerequisites

- You have installed and authorized a node manager. See the topics [Node manager installation](#) and [Trusting a node](#) in the *Spotfire® Server Installation and Administration* user guide. Note that the folder paths on Linux should not contain any whitespace characters (as per Linux naming best practices).
- The Spotfire Server and the node manager are up and running.
- You have deployed the operating-system-specific SDN for the Spotfire Service for Python into a deployment area of your Spotfire Server. For information about deploying the SDN, see the topic [Adding software packages to a deployment area](#) in the *Spotfire® Server Installation and Administration* user guide.



Important

- You can install the Spotfire Service for Python on a node manager running on a computer with an operating system (OS) that is different from that of your Spotfire Server.
 - All services belonging to the same Spotfire deployment area must run on node managers with the same OS.
- Optional: You have created and imported a custom configuration for the Spotfire Service for Python.

Procedure

1. Log in to the Spotfire Server and click **Nodes & Services**.
2. Under **Node managers**, select the node to which you want to add the Spotfire Service for Python. A running service shows a green circle with a check mark next to the selected node manager. In the **Services** area, the names of the current services are shown in the lower-right pane of the window.
3. Click **Create new service**.

4. Make your selections in the Create new service dialog:
 - a) Under **Deployment area**, select the area where you deployed the Spotfire Service for Python.



Administrators often create a Test deployment area to use as a staging server.

- b) Under **Capability**, select **PYTHON**.
 - c) Under **Configuration**, select the service configuration to apply to the service.



In most cases, this is the default configuration, unless you have created a custom configuration. See the *Spotfire® Server Installation and Administration* user guide for more information on creating a custom configuration.

- d) Under **Service name**, provide a display name for the service.
 - e) Under **Add instances**, specify the number of instances.
 - f) Under **Instances name**, provide a name for the instances.
 - g) Under **Number of instances**, leave the option set to 1.

The Spotfire Service for Python can have only one instance per node. If you set it to a value other than 1, the service does not work as expected.

One Spotfire Service for Python instance can serve multiple users simultaneously. See the [Custom configuration properties](#) for more information.

- h) Under **Port**, you can change the default as needed.

5. Click **Create service**.

To view the progress of the installation, click the **Activity** tab.

Result

The service is installed and starts.



If you experience errors, click **View logs** for more information.

What to do next

For information on the remaining setup tasks, see the topic [Post-installation steps](#) in the *Spotfire® Server Installation and Administration* [user guide](#).

Configuring the Service

You can customize certain behaviors for the Spotfire Service for Python by exporting the service properties, editing the file, importing service properties, and then applying the new configuration.

Perform this task on the computer where you have installed Spotfire Server.



For general information about configuring services for Spotfire Server, see the [Spotfire® Server Installation and Administration](#) user guide.

Prerequisites

- You must have completed the steps outlined in [Installing the service on a node manager for a Spotfire Server](#) on page 16.
- You must have administrative read-write privileges to save the changed configuration file.

Procedure

1. Open a command line as administrator and change the directory to the location of the command-line config tool (on Windows, `config.bat`; on Linux, `config.sh`).

The default location is `<server installation dir>/tomcat/spotfire-bin`.

2. On the command line, issue the following command:

```
config export-service-config --capability=Python --deployment-area=<your deployment area name>
```

If you already have a configuration name from previously editing the configuration, and you want to change that configuration, provide the configuration name using the `--config-name=<configuration name>` option.

The file named `custom.properties` is exported and written to the directory `<server installation dir>/tomcat/spotfire-bin/config/root/conf`.

3. When prompted, provide the password for the config tool.
4. Using a text editor, open and edit the file `<server installation dir>/tomcat/spotfire-bin/config/root/conf/custom.properties`.
The text file contains comments to provide you with information about each property. Alternatively see the individual reference topics for the properties for more information.
5. Save the changes, and then close the text editor.
6. Optional: Copy any additional files to add to the configuration into the directory `<server installation dir>/tomcat/spotfire-bin/config/root/conf`.
For example, you can add a configuration script. (Configuration scripts must be specified in the custom property `startup.hook.script`. See [Startup script](#) on page 25 for more information.).
For a Linux OS deployment, you can add a Dockerfile.
7. From a command line, return to the directory for the command-line config tool.
The default location is `<server installation dir>/tomcat/spotfire-bin`.

8. On the command line, issue the following command:

```
config import-service-config --config-name=<new-config-name>
```

The `config-name` you specify identifies this configuration, so provide a name that is meaningful for the change. For example, if you create a configuration with a specific debugging level, you might name the configuration `Debugging`.



You cannot overwrite the default configuration. You must provide a configuration name when you import the custom configuration.

See the reference topic for `import-service-config` in the *Spotfire Server and Environment Installation and Administration* [user guide](#) for information about additional options.

9. Open a web browser and log in to the administration console for Spotfire Server.
10. Click **Nodes & Services**.
11. Under Network, select **Node managers**, and then select the Spotfire Service for Python.
12. Click **Edit**.
The Edit service dialog is displayed.
13. In the **Configuration** drop-down list box, select the configuration name to apply, and then click **Save**.

Result

The service is stopped, and then Spotfire Server restarts the service and applies the new configuration. The Spotfire Service for Python begins recording information to the Service Logs. For more information, see [Service Logs](#).



To change the new configuration, export it again, specifying its name. If you do not specify the name, the default configuration is exported.

Custom configuration properties

You can fine tune the behavior of the Spotfire Service for Python by setting custom configuration properties.

Allowed engines

You can specify the number of Python engines that can run concurrently, and the number of Python engines that are allocated in the Spotfire Service for Python queue.

| Configuration property | Default setting | Description |
|---------------------------------|---|---|
| <code>engine.session.max</code> | <i><one less than the number of logical processors available on the node></i> | <p>The maximum number of Python engine sessions that are allowed to run concurrently in the Spotfire Service for Python. Each user running data functions in a Spotfire analysis uses its own session.</p> <p>The default is one less than the number of logical processors on the host.</p> |
| <code>engine.queue.size</code> | <i><one quarter of the number of logical processors on the host, constrained to a minimum of 1 and a maximum of 10. ></i> | <p>The number of Python engines preallocated and available for new sessions in the Spotfire Service for Python queue. The service always starts enough engines to keep the queue at the requested level.</p> <p>The total number of engines that can run at the same time is the sum of <code>engine.session.max</code> + <code>engine.queue.size</code>.</p> <p>This number can be set manually to a value higher than 10.</p> |

For more information on how engine resources can be managed, see [Service resource management scenarios](#) on page 49.

Compressed job contents and results

You can compress large data sets sent to Python. You can also compress returned results for Python data functions that are then sent to the Spotfire Service for Python.

| Configuration property | Default setting | Description |
|---|-----------------|--|
| <code>jetty.gzip.compression-level</code> | 4 | Set to a value from 1 to 9, inclusive. 1 offers the fastest compression speed but at a lower ratio. 9 offers the highest compression ratio but at a lower speed. |
| <code>jetty.gzip.min-gzip-size</code> | 32 | The minimum size in bytes before the response is compressed. |
| <code>jetty.gzip.inflate-buffer-size</code> | 2048 | The size, in bytes, of the buffer to inflate a compressed request. Set to -1 to disable compression uploads. |

Custom Python interpreter

You can specify a Python interpreter that is different from the interpreter that is bundled with the distribution.

For example, due to constraints or requirements of some Linux operating systems, you might need to use the version of Python that is installed on your system. For more information about system requirements and their installed Python versions, see the Spotfire Service for Python [system requirements](#).

If you want to use the system-installed Python interpreter in a containerized system, see [Configuring a custom Docker image on a node with internet access](#) on page 7.

To use the system-installed Python interpreter, or another compatible Python interpreter that is different from the interpreter bundled with the Spotfire Service for Python, you must meet the following prerequisites.

Prerequisites for using a custom Python interpreter

- Ensure that Python version 3.9.x or higher is installed on the system.
- Set the environment variable `SPOTFIRE_PYTHON_HOME` to the path to the directory for the installed Python interpreter that you want to use.
 - On Linux, the Spotfire Service for Python uses and searches for `<SPOTFIRE_PYTHON_HOME>/bin/python3`.
 - On Windows, the Spotfire Service for Python searches for `<SPOTFIRE_PYTHON_HOME>/python.exe`
- Install the spotfire Python package (available at <https://pypi.org/project/spotfire/>).



The environment variable `SPOTFIRE_PYTHON_HOME` must be accessible to the running node manager.

- For Linux, you can establish this connection with the file `start-python-service.sh` (located at `/opt/spotfire/nodemanager/<version>/nm/services/python-service-linux-<guid>/start-python-service.sh`).
- For Windows, you can set this connection with the file `startPython.bat` (located at `C:/spotfire/nodemanager/<version>/nm/services/Python Service Windows-<guid>/startPython.bat`).


Disable warnings

By default, warnings from the data function (or included packages) are sent to the client when executing a data function through the service. It is possible to suppress warnings by disabling the warning alert.

| Configuration property | Default setting | Description |
|------------------------------------|--------------------|--|
| <code>disable.warning.alert</code> | <code>false</code> | Set to true to disable sending warnings from the data function to the client when executing a data function. |

Docker container built for no internet access

If you are deploying a Spotfire Service for Python to use in a system with no internet access, you must configure the container image so that it does not try to build an image on top of the one that you have initially built in a system with internet access.

| Configuration property | Default setting | Description |
|--------------------------------------|--------------------|--|
| <code>use.immutable.container</code> | <code>false</code> | <p>Set to true to indicate to the service that the image identifier (name:tag) that is given in the <code>docker.image.name</code> is immutable, and should be used directly without building an image on top of it. The default is false.</p> <p>This property is required only for containers that are used in Spotfire Server deployments that do not have access to the internet.</p> <div>  <p>To use an immutable container, you must have installed the supported version of the Java Development Kit (for the Spotfire Server) in the container.</p> </div> |

For more information, see [Configuring a custom Docker image on a node with no internet access](#) on page 8.

Engine ports

Python engines running under the Spotfire Service for Python require open ports to communicate. The first available port, and the range to the last available port, are determined by these two settings.

The defaults specify a range between 62001 and 63000.

| Configuration property | Default setting | Description |
|--------------------------------|-----------------|---|
| <code>engine.port.min</code> | 62001 | The first specified available port set for a Python engine. |
| <code>engine.port.range</code> | 1000 | This value, added to the value specified in <code>engine.port.min</code> , indicates the range of the ports available for the Python engines. |

Engine pruning

When the Spotfire Service for Python reaches a certain percentage of capacity of usage, then the Spotfire Service for Python begins pruning Python engines to free service resources.

| Configuration property | Default setting | Description |
|--------------------------------------|-----------------|--|
| <code>engine.prune</code> | 10 | The time, in seconds, that a Python engine can be idle before the Spotfire Service for Python prunes it. |
| <code>dynamic.prune.threshold</code> | 60 | <p>The Spotfire Service for Python capacity at which idle pruning is engaged, as a percentage value.</p> <p>By default, when the Spotfire Service for Python reaches 60% capacity of usage, then it begins the idle-pruning process as specified by <code>engine.prune</code>.</p> <ul style="list-style-type: none"> Set to 0 to always prune when a Python engine is idle. Set to 100 to never prune when a Python engine is idle. |

For more information, see [Service resource management scenarios](#) on page 49.

Engine timeout

You can specify the length of time a Python engine runs to complete a task before failing with a timeout error. You can also specify the length of time for a Python session to exist.

| Configuration property | Default setting | Description |
|---------------------------------------|-----------------|---|
| <code>engine.execution.timeout</code> | 600 | The length of time, in seconds, that the Spotfire Service for Python allows the Python engine to execute a request before stopping the execution with a timeout error. |
| <code>engine.session.maxtime</code> | 1800 | <p>The length of time, in seconds, that the Spotfire Service for Python allows the Python engine session to exist before killing it.</p> <p>To disable session pruning, set this value to -1.</p> |

File size upload limit

When planning for uploading files for the Spotfire Service for Python, you can set the file size limit for uploading using the properties setting for the Spring Boot framework. If you change this setting, consider how the file size might affect the speed at which files can be uploaded.

| Configuration property | Default setting | Description |
|--|-----------------|--|
| <code>spring.servlet.multipart.max-file-size</code> | 100 MB | The total file size for upload cannot exceed the value for this setting. |
| <code>spring.servlet.multipart.max-request-size</code> | 100 MB | The total request size for a multipart file upload cannot exceed the value for this setting. |



Logging level



By default, the logging level is set for the Spotfire Service for Python to provide informational progress. In the `custom.properties` file, you can set the logging level through the property `loggingLevel`. The Spotfire Service for Python uses Log4J2-defined logging levels.

| Level | Description |
|-------|--|
| ALL | All levels are reported. |
| TRACE | Reports a finer-grained level of events than the <code>DEBUG</code> level. |
| DEBUG | Reports a fine-grained level of events. This setting is most useful when you are debugging problems with the service. |
| INFO | The default. Reports informational messages that highlight the progress of the Spotfire Service for Python, but at coarse-grained level. |
| WARN | Reports potentially harmful situations. |
| ERROR | Reports errors. These errors might still allow the Spotfire Service for Python to continue running. |
| FATAL | Reports only very severe errors that cause the Spotfire Service for Python to stop. |
| OFF | Turns off logging. |

Manage Java options

You can set certain Java command-line options for the Spotfire Service for Python for managing such settings as the Java heap size.

| Configuration property | Default setting | Description |
|-------------------------------------|-----------------|--|
| <code>disable.java.core.dump</code> | TRUE | <p>By default, when the JVM stops responding, it does not write full core dumps to the <code>temp</code> directory. Set this value to <code>FALSE</code> to enable full core dumps.</p> <div>  <p>Setting this value to <code>FALSE</code> can potentially cause the core dump to fill all available disk space. Use with caution.</p> </div> <div>  <p>To get the core dump file in a non-containerized environment you must make sure that:</p> <ul style="list-style-type: none"> • <code>ulimit</code> is set to unlimited (<code>\$ ulimit -c unlimited</code>). • The location of core dumps for your operating system is specified by configuring the <code>/proc/sys/kernel/core_pattern</code> file. </div> |

| Configuration property | Default setting | Description |
|------------------------|-----------------|--|
| javaOptions | none | <p>In systems with a lot of memory, administrators might want to limit the initial or maximum heap size that the Spotfire Service for Python can use.</p> <p>In the following example, the custom property sets the Java initial heap size to 1GB.</p> <pre>javaOptions:-Xms1g</pre> <p>In the following example, the custom property sets the Java initial heap size to 2GB and the maximum heap size to 4GB.</p> <pre>javaOptions:-Xms2g,-Xmx4g</pre> <div>  <p>The javaOptions setting controls Java memory only, and does not control memory allocated in the Python runtime. (Memory allocated by Java is typically very small, which makes it possible to set a very low value for the initial heap size, such as Xms64M.)</p> </div> <p>To include multiple java options, delimit the options with a comma and no space. The following examples demonstrate setting an empty property, setting one property, setting two properties, and setting three properties.</p> <pre>javaOptions: javaOptions:-Xms2g javaOptions:-Xms2g,-Xmx4g javaOptions:-Dfoo="foobar",-Xms2g,-Xmx4g</pre> <div>  <p>The javaOptions property cannot contain spaces. For example, -Dfoo="foo bar" is not a valid property setting.</p> </div> <p>For other Java command line options, see the Java documentation.</p> |

Package library location

You can set the location of packages that Python can use in the Spotfire Service for Python configuration settings.


| Configuration property | Default setting | Description |
|------------------------|-----------------|--|
| packagePath | none | <p>The absolute path to the shared package library location. When specifying the path, you must use a forward slash regardless of operating system.</p> <ul style="list-style-type: none"> If you install packages into your Spotfire® deployment using the SPK process, then this setting is not required. If you install packages onto the server using <code>pip install <packagename></code>, then set this path to the package installation location. |

Example

```
packagePath: /opt/python/library
packagePath: C:/Python/Lib
```




Safeguarding your environment

This custom property setting helps minimize the risk of malicious acts in your environment.

| Configuration property | Default setting | Description |
|--|-----------------|--|
| <code>disable.spotfire.trust.checks</code> | FALSE | <p>By default, the Spotfire Service for Python checks whether a data function has come from a trusted source.</p> <p>Set to <code>TRUE</code> to not check for the data function trust status of any data function run on the Spotfire Service for Python.</p> <div>  <p>Setting this value to <code>TRUE</code> results in all Spotfire data functions executing unrestricted. We strongly recommend that you ensure that your service is fully secured, that engine containers are enabled, and that network access from the containers is limited (using a firewall) to only necessary servers and ports.</p> <p>For more information about script and data function trust, see the <i>Spotfire® User Guide</i> and the <i>Spotfire® Administration Manager User Guide</i>.</p> </div> |

Startup script

You can specify a script to run before a container or Python engine is started.

| Configuration property | Default setting | Description |
|----------------------------------|-----------------|---|
| <code>startup.hook.script</code> | none | <p>The path and name of a startup hook script that runs before any container or Python engine is started. This value can be empty (for no script) or a relative path from the Spotfire Service for Python working directory.</p> <p>Place the startup script in the directory with the <code>custom.properties</code> file.</p> <div>  <p>To specify the path, you must use forward slash (/) regardless of the host operating system.</p> </div> <p>The Spotfire Service for Python can run either a <code>.bat</code> or a <code>.sh</code> file format, depending on the host operating system.</p> <p>On a Linux system, the script must have appropriate permissions before the Spotfire Service for Python executes it.</p> <div>  <p>Important Remember that for any script you write, the line endings must be appropriate for the operating system where the service runs. Many text editors can perform end-of-line (EOL) conversion.</p> </div> <p>You can use the startup script to set environment variables, create directories, download files, or prepare the file system settings in other ways before the service starts. (For example, you can perform Docker commands in the script for a Linux deployment area, or you can run another script. See Containerized service on page 6 for more information.)</p> |

Example

- Relative path for a Linux deployment area: `conf/mystartupscript.sh`
- Relative path for a Windows deployment area: `conf/mystartupscript.bat`

JMX monitoring

You can use an installation of Java Management Extensions (JMX) and the Remote Method Invocation (RMI) connector to monitor the Spotfire Service for Python.

Remove the comment marker and set the properties to connect to JMX using RMI in the custom properties file. To use JMX monitoring, you must provide valid settings for all five of these properties.



Important Because JMX monitoring requires connecting to the specific IP address of the node, you must create a custom configuration for each node to monitor.

| Configuration property | Default setting | Description |
|-------------------------------|-----------------|---|
| <code>jmx.rmi.username</code> | None | Set this value to the JMX user name. |
| <code>jmx.rmi.password</code> | None | Set this value to the JMX password. |
| <code>jmx.rmi.host</code> | None | Set this value to the IP address of the host computer (that is, the computer where the node manager and the Spotfire Service for Python are installed.) |
| <code>jmx.rmi.port</code> | 1099 | Set this value to an available port for the RMI connection with JMX. |
| <code>jmx.active</code> | None | Set this value to <code>TRUE</code> to activate JMX. |

For more information, see [Monitoring the service using JMX](#) on page 52.

Containerized configuration

The Spotfire Service for Python provides custom properties that are specific to the Linux operating system.

| Configuration property | Default setting | Description |
|------------------------------------|-----------------------------|--|
| <code>use.engine.containers</code> | <code>TRUE</code> | Runs the Python engine inside a container when this value is set to <code>TRUE</code> (the default). |
| <code>ram.limit</code> | 2500 | The amount of RAM and SWAP memory to which the Python engine containers are constrained, in megabytes. |
| <code>docker.image.name</code> | <code>debian:12-slim</code> | <p>When you use containers, the Spotfire Service for Python builds a custom image based on a starting image.</p> <p>This property is used by the Dockerfile as the <code>FROM</code> line.</p> <p>See https://docs.docker.com/engine/reference/builder/#from for more information.</p> <p>The Spotfire Service for Python default is <code>debian:12-slim</code>.</p> |

For more information, see [Containerized service](#) on page 6.

Package management for the Spotfire Service for Python

Spotfire Service for Python programmers in your organization can develop their own packages or take advantage of some of the thousands of compatible packages developed by other Python programmers, and then share the analyses that use those functions with Spotfire users in your organization.

The largest and most commonly-used curated repository for Python packages is the Python Package Index (PyPI). Administrators should check with authors to make sure they know the location for needed packages.

Spotfire analysts can create data functions that use these packages, and then share them with either other analysts using the installed Spotfire client, or with Spotfire web client users.



Important Packages must be installed in the same type of environment as where they are executed. For example, if you built the `debian:12-slim` Docker image for your containerized Spotfire Service for Python, make sure the packages are installed on a node manager running the same operating system as the containerized engines. If this is not possible, then see the workaround options in [Troubleshooting the service](#) on page 54.

- To share a collection of Python packages and data functions that use them with Spotfire installations connected to your Spotfire Server, create the Spotfire package (SPK) containing the Python packages, and use the `--analyst` flag. For more information, see the instructions in the guide *Python Data Functions in Spotfire®*, provided with the [Spotfire documentation](#). Packages deployed in this manner are distributed to other Spotfire installations in your organization when they connect to the Spotfire Server where the packages are deployed.



Packages that require post-install steps, such as [pywin32](#), cannot be installed using a Spotfire SPK but must be installed manually in Spotfire Analyst or Python Service.

- To enable data functions that use the code in the Python packages for web client users, see [Distribute Python packages](#) on page 33. Packages deployed in this manner are available on the Spotfire Server node manager where the Spotfire Service for Python is installed.

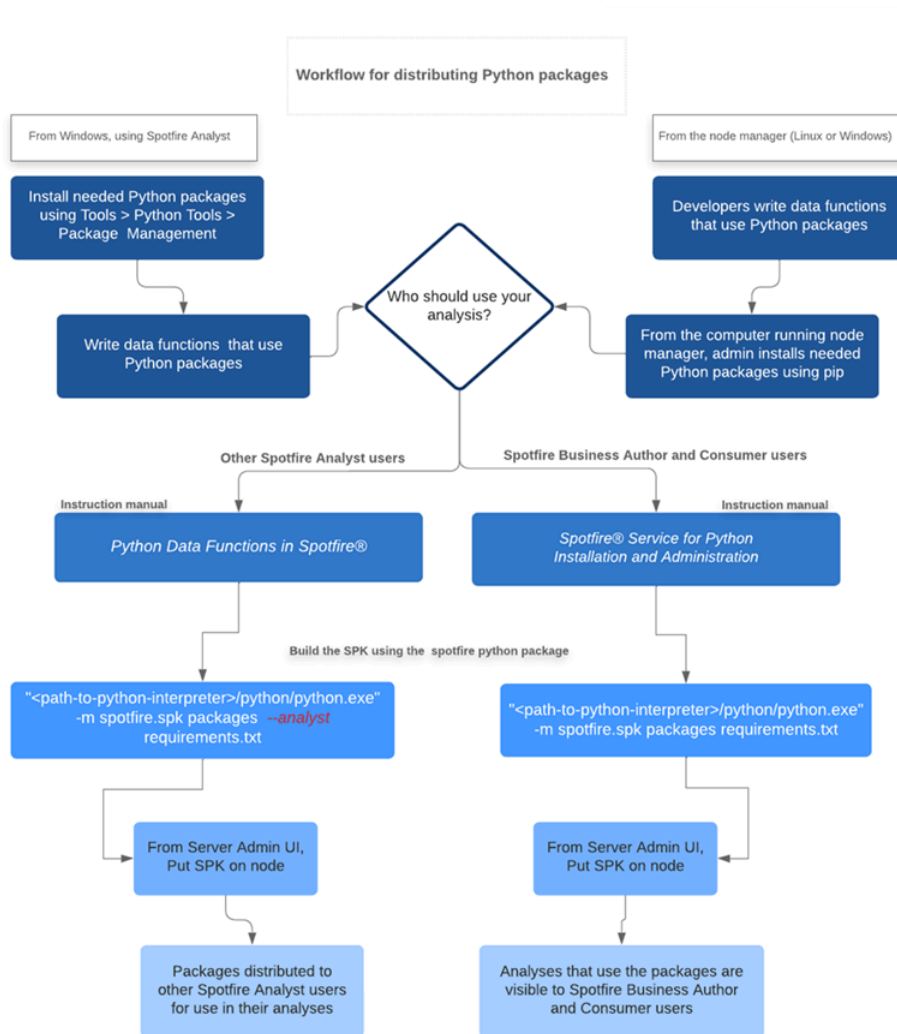
A Spotfire package must be created using the same operating system as that of the node manager running the Spotfire Service for Python.



The exception to this rule is if you use the provided Docker containers to bundle Python packages for a Linux node manager: you can create this SPK for a Linux deployment on either a Linux computer or on a Windows computer. See [Build a Spotfire package for Spotfire Service for Python \(Linux\)](#) on page 42.

- If your node manager is running on a Windows server, then you can create the SPK either from a Windows computer running the installed Spotfire client, or you can create the SPK from the computer running the node manager where the Spotfire Service for Python is installed.
- If your node manager is running on a Linux server, then you can create the SPK using the provided Docker containers from your desktop computer, or you can create the SPK from the computer running the node manager where the Spotfire Service for Python is installed.

The following image shows the workflow for creating an SPK to share Python packages with other analysts, or to enable data functions for Spotfire web client users.



Find help

Spotfire includes many avenues to help with packages, whether they are Python language packages to use with the Spotfire Service for Python or Spotfire packages (SPKs).

| Task | Help resource |
|---|---|
| Deploying a package using the Spotfire package mechanism. | Spotfire® Server Installation and Administration at https://docs.tibco.com/products/spotfire-server |

Using an alternative Python package repository

If your company keeps an internal repository for approved Python packages, you can set a preference using the Spotfire Administration Manager to point to the URL for the internal repository.

Changing this preference applies only to installing packages using the installed Spotfire client feature **Tools > Python Tools > Package Management**. If you are installing packages using the `pip` command, and you want to use your internal repository, then pass the URL for your internal repository in the `pip` command parameter `--index-url`.



Packages that are provided with the Spotfire Service for Python are not affected by this setting.

Prerequisites

You must have a Spotfire Administration license to access the Administration Manager tool.

Procedure

1. In the installed Spotfire client, click **Tools > Administration manager**.
2. In the Administration Manager, click the **Preferences** tab.
3. From the **Selected group** list, click the name of the group for which to set the property.
For example, from the list select **Script Author** to set the preference for any user in the Script Author group. To change the value for all users, select **Everyone**.
4. In the Preferences list for the selected group, expand **DataFunctions**, and then click **DataFunctionsPreferences**.
5. Click **Edit**.
The properties for **DataFunctionsPreferences** are now editable.
6. From the property list, select **PythonPackageRepositoryIndexURL** and then, in the right column for this property, provide the URL for your internal repository.

Result

The custom URL overrides the default Python package index (PyPI) and accesses only the packages available in the specified custom internal repository.

What to do next

Repeat this process for any other groups that author data functions using the Spotfire Service for Python. For more information, see the help for Administration Manager.

Included packages

Your installation of Spotfire includes version 3.12.9 of the Python interpreter and several packages it needs to run under Spotfire.





Important The packages listed in this table are required for the Spotfire Service for Python and the Python interpreter to work together. Removing any of these packages can cause your Spotfire Service for Python data functions to fail.

If you create and distribute an SPK containing a different Python interpreter, and the interpreter specifies different packages or package versions than those listed here, your data functions could fail to work as expected. Python package authors strive for backward compatibility, but if you encounter such an issue, Spotfire Support can help you.

To help protect against this sort of issue, your distribution of the Spotfire Service for Python includes the PIP constraints file `interpreter-constraints.txt`. If you create an SPK for distribution that contains a different version of the Python interpreter, specify the `--constraints` option. Any dependency on an included package is "constrained" and the included version is used instead.

See [Creating a Spotfire package for an alternative Python interpreter \(for Windows\)](#) on page 45 for an example of using the constraints file.

| Package name | Version | Description | More information |
|-----------------|-------------|---|---|
| numpy * | 2.2.4 | <p>Provides the following.</p> <ul style="list-style-type: none"> • An N-dimensional array object. • Broadcasting functions. • Tools for integrating C/C++ and Fortran code. • Linear algebra, Fourier transform, and random number capabilities. <div>  <p>NumPy version 1.19.4 has known incompatibilities when it is run on certain more recent versions of Windows. These compatibilities can cause Python to fail when it is used with Spotfire. If you build custom packages SPK on Windows, ensure the SPK does not include NumPy version 1.19.4.</p> </div> <div>  <p>Starting in NumPy version 2.0.0, operations involving NumPy arrays and Python scalars now follow stricter and more predictable promotion rules, as described in NEP 50 – Promotion rules for Python scalars.</p> <p>This means that Python data function authors must check and adjust any Python data functions using scalars to make sure they follow the Python scalar promotion rules.</p> </div> | https://pypi.org/project/numpy/2.2.4/ |
| packaging | 24.2 | Provides support for parsing and handling Python package versioning, dependency specifications, and other metadata related to Python packaging. | https://pypi.org/project/packaging/24.2/ |
| pandas * | 2.2.3 | Provides data structures and data analysis tools for dealing with tabular data, ordered and unordered time series data, matrix data, and other types of data sets. | https://pypi.org/project/pandas/2.2.3/ |
| pip | 25.0.1 | Provides support for installing packages. | https://pypi.org/project/pip/25.0.1/ |
| python-dateutil | 2.9.0.post0 | Provides extensions to the datetime module in Python. | https://pypi.org/project/python-dateutil/2.9.0.post0/ |
| pytz | 2025.2 | Provides a platform for cross-platform time zone calculations. | https://pypi.org/project/pytz/2025.2/ |
| setuptools | 78.1.0 | Provides tools for building, installing, upgrading, and uninstalling Python packages. | https://pypi.org/project/setuptools/78.1.0/ |
| six | 1.17.0 | Provides utility functions for smoothing over the differences between the Python versions 2 and 3. | https://pypi.org/project/six/1.17.0/ |

| Package name | Version | Description | More information |
|--------------|---------|--|---|
| spotfire | 2.3.0 | <p>Provides functions for integrating Python with Spotfire.</p> <p>In spotfire version 2.2.0 (and later), the following changes were made:</p> <ul style="list-style-type: none"> Export of geographic data (for example, GeoDataFrame objects from the geopandas package) from Python back into Spotfire now performs less processing. The export process will no longer create (or recreate) the XMin, XMax, XCenter, YMin, YMax, and YCenter columns and most of the additional table and column properties. If these columns are required, or you are creating a geocoding table, you can use the new public function <code>set_geocoding_table</code>: <pre>import spotfire import geopandas as gpd geodata = gpd.GeoDataFrame(...) spotfire.set_geocoding_table(geodata)</pre> <ul style="list-style-type: none"> Information about a data function's configured inputs and outputs have now been made available to a data function at execution time. The global variables "spotfire_inputs" and "spotfire_outputs" now contain read-only sequences of objects which describe each input or output (modifying the contents of these variables will not change the behavior of Spotfire; the objects themselves only have read-only properties). For example: <pre>out = {'Kind': [], 'Name': [], 'Type': []} for i in spotfire_inputs: out['Kind'].append("input") out['Name'].append(i.name) out['Type'].append(i.type) for o in spotfire_outputs: out['Kind'].append("output") out['Name'].append(o.name) out['Type'].append("unknown")</pre> <ul style="list-style-type: none"> A new module (spotfire.support) has been added to the spotfire package, which gathers information about your Python execution environment and puts it into a troubleshooting bundle for sharing with Spotfire Support. You might be asked to generate such a bundle if you open a case involving Python data functions with Spotfire Support; they will provide you with the proper Python code for your environment | https://pypi.org/project/spotfire/2.3.0 |
| tzdata | 2025.2 | Provides zic-compiled binaries for the IANA time zone database, intended to be a fallback for systems that do not have system time zone data installed (or do not have it installed in a standard location). | https://pypi.org/project/tzdata/2025.2/ |
| wheel | 0.45.1 | The reference implementation of the Python wheel packaging standard, as defined in PEP 427 . | https://pypi.org/project/wheel/0.45.1/ |



* Exporting an SBDF that contains empty String columns causes an error with pandas and numpy. See [Troubleshooting the service](#) on page 54 for more information.

The Spotfire Package (SPK)

A Spotfire SPK is usually created and tested by developers to package and deploy third-party extensions to the Spotfire Server, which can then be distributed to the Spotfire Server node for use by another service (and in some cases, distributed to Spotfire clients).

Even though they are both called “packages”, the Python package and the Spotfire package (SPK) are different.



- The Python library (PyPI) contains Python modules.
- The Spotfire package is a means to deploy extensions to the Spotfire Server, which either distributes its contents to installed Spotfire client users, or installs a service, such as the Spotfire Service for Python or an alternative Python interpreter, to use from the Spotfire Server node.

This Spotfire installation provides a specialized Python package, called 'spotfire', that creates an SPK to hold packages or an alternative Python interpreter. The 'spotfire' package is also available for download from PyPI.

SPK versioning

To share packages among data function authors in your organization, you can create the file `<your-filename>.spk` containing the packages to distribute to others. You might need to change or update the packages or package versions that you distribute, which requires changing the version of the SPK containing the packages.

You can create or change a Spotfire SPK using the steps described in [Creating a Spotfire package for Python packages from a Windows computer](#) on page 39. The package `spotfire.spk` creates a new SPK using the versioning rule details for the following tasks.



- Python package versions shared among team members must be kept synchronized.
- You can install multiple SPKs containing Python packages on the Spotfire Server, as long as each SPK has a unique name and ID.
- Uploading a new SPK overwrites any older version of that same SPK that was previously deployed.

Table

| Task | Procedure | Version result | Version example | Comment |
|---|--|--|-----------------|---|
| Generating a new requirements.txt. | Pass the new requirements.txt to the Python script. | The version is always set to 1.0.0.0 by default. | 1.0.0.0 | The script overwrites the old SPK, and the list contains only the packages you provide in requirements.txt. |
| Recreating a new requirements.txt using the same version. (That is, you do not need to increment or keep the older requirements.txt.) | Regenerate the SPK, passing the new requirements.txt to the Python script. | The version is always set to 1.0.0.0 by default. | 1.0.0.0 | Spotfire Server does not register the package as a new one, so it does not distribute the package to the users. |

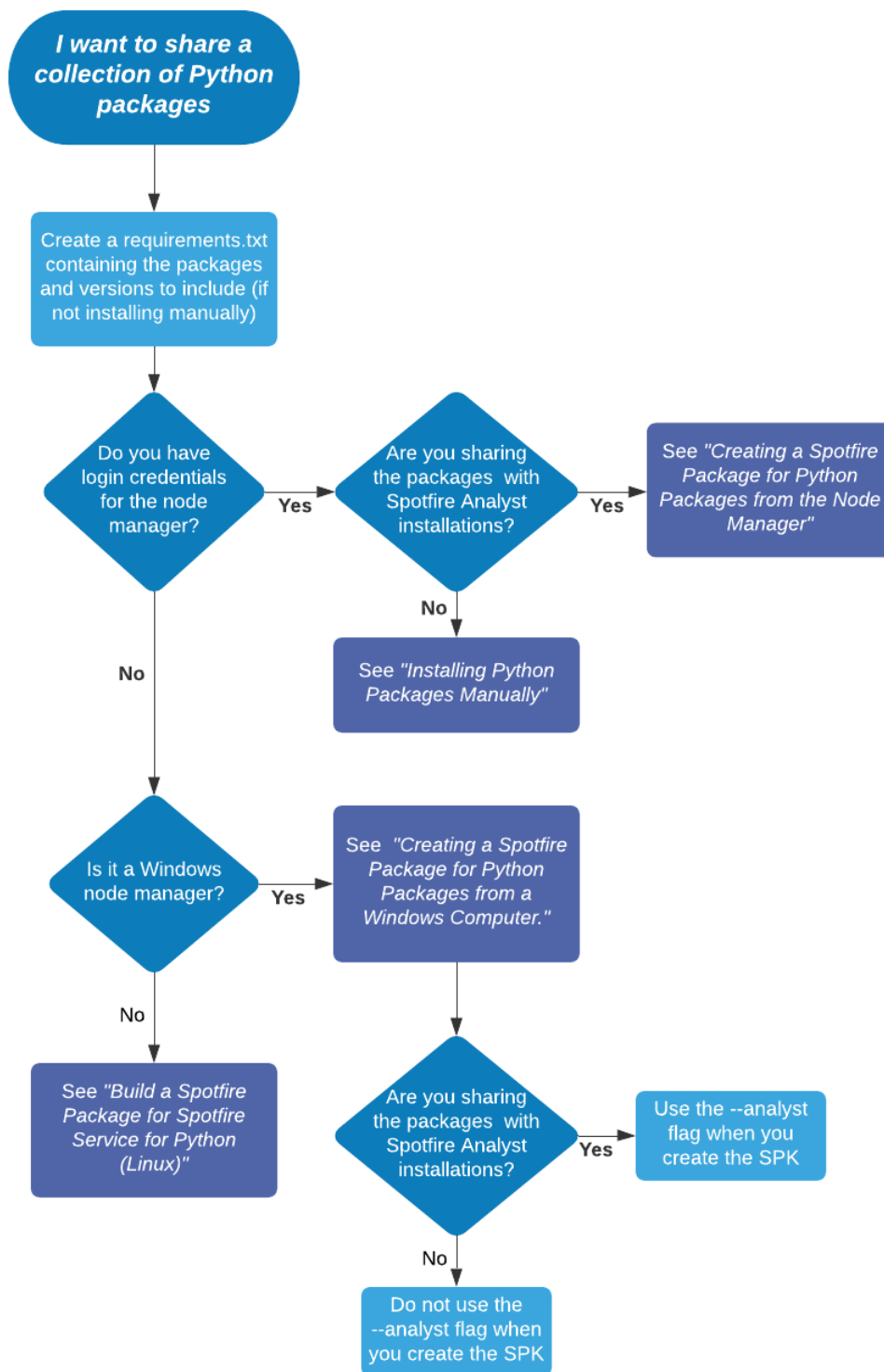
| Task | Procedure | Version result | Version example | Comment |
|--|---|---|-----------------|--|
| Adding package names to an existing <code>requirements.txt</code> . | Edit the <code>requirements.txt</code> , and then pass it to the Python script. | The version is incremented to a minor version number. | 1.1.0.0 | The script overwrites the old SPK, and the list contains only the packages you provide in <code>requirements.txt</code> . Spotfire Server registers the SPK as changed and distributes it to the users. |
| Removing package names from an existing <code>requirements.txt</code> . | Edit the <code>requirements.txt</code> , and then pass it to the Python script. | The version is incremented to a major version number. | 2.0.0.0 | The script overwrites the old SPK, and the list contains only the packages you provide in <code>requirements.txt</code> . Spotfire Server registers the SPK as changed and distributes it to the users. |
| Assigning a specific version number to a <code>requirements.txt</code> . | Run the Python script and pass in the <code>requirements.txt</code> in the command, along with the <code>version</code> , setting it to the version you want. | The version number is set to the value provided in the argument | 1.2.3.4 | The version argument must be passed as a string containing four components (for example, " <code>--version = 1.2.3.4</code> " or " <code>-v 1.2.3.4</code> "). |

Distribute Python packages

You can distribute Python packages by using the Spotfire package (SPK) mechanism for either a Spotfire Server installation with a node manager running on Windows OS, or for a Spotfire Server installation running a node manager on a supported Linux OS.

- If your node manager is running on a Windows server, then you can create the SPK containing Python packages from a Windows computer. You can specify whether the packages are installed only on the node manager or are also distributed to installed Spotfire clients connected to the Spotfire Server.
- If your node manager is running on a Linux server, then you can do one of the following.
 - Use the provided Docker container and the SPK mechanism.
 - Create the SPK containing Python packages from the Linux computer that is running the node manager where the Spotfire Service for Python is installed. (This option requires login credentials to that computer.)

The following image shows the options for deciding the best way to distribute Python packages.



Installing Python packages manually

If you have a small Spotfire Server deployment, and you do not need to manage packages across several nodes or servers, then you can install packages directly on the computer running the node manager, rather than creating an SPK.

Perform this task on the computer hosting the Spotfire Service for Python (in the directory where Python is installed), and then on the computer where Spotfire Server is installed.

Any time you install additional packages or update existing packages, be sure to install them in the directory you specified for your `packagePath`. You can have only one package path for the Spotfire Service for Python installation. See [packagePath](#) for more information.

Avoid installing packages that are included in the Spotfire Service for Python. Installing a different version of an included package can cause unexpected results. For a list of these packages, see [Included packages](#) on page 29.



When you update your Python installation, be sure to update your package installations, too.

Prerequisites

- You must have administrative privileges to edit files on the computer running the node manager.
- You must have administrative privileges and the tools password to update the `custom.properties` file.

Procedure

1. Create the directory to store the Python packages.
This directory is specified as the path to use to install Python packages, and to set the Spotfire Service for Python custom property, [packagePath](#).
2. From the command prompt, browse to the directory where the Python interpreter for your Spotfire Service for Python is installed.
By default, this directory is `/nodemanager/services/<spotfire-service-for-python-name>/python`.
3. Run the following command to install the needed package.

```
python -m pip install --target=<packagePath> <packagename>
```



Define the target location to install packages to the value you provided in the [packagePath](#) custom configuration setting.

The package and its dependent packages are installed.

4. Update the Spotfire Service for Python configuration to specify the package path.

You need to export, edit, and reimport the `custom.properties` file only the first time to set the package path.



Remember that when you change the `custom.properties`, you must restart the Spotfire Service for Python to have it take effect.

- a) Follow steps 1-3 in [Configuring the Service](#) on page 18 to export the service configuration file `custom.properties`.
- b) In the exported `custom.properties` file, locate the entry for `packagePath`.
- c) Provide the path that you specified for the installed packages.



The configuration setting `packagePath` requires forward slashes (/) regardless of operating system.

- d) Complete the steps to save and import the changed service configuration file, as described in [Configuring the Service](#) on page 18.

Creating a Spotfire package for Python packages on the node manager

An installation of Spotfire Service for Python on the node manager includes a Python interpreter and a set of packages to enable using Python in Spotfire.

The package named 'spotfire' provides tools for building SPKs to share Python packages with other data function authors in an organization, or to enable web client users to see Python-enabled analyses in a web browser.

Important You must build a package for the operating system on which your Spotfire Server node manager is installed. You can build the SPK from either a Linux or a Windows computer running the node manager. This topic describes building the SPK using the spotfire package that is included in the download bundle.

- If you are building an SPK for Linux only, then the SPK is deployed to the node manager. The packages it contains are used by data functions in analyses accessed through a web browser by web client users.
- If you are building an SPK for Windows only, then the SPK can be deployed to the node manager, to other installed Spotfire clients, or to both. Packages deployed to the node manager are used by data functions in analyses accessed through a web browser by web client users. (Alternatively, you can build the SPK from an installation of Spotfire. For those details, see [Creating a Spotfire package for Python packages from a Windows computer](#) on page 39.)

The package builder relies on pip, the Python command-line application to install Python packages. The package builder uses a `requirements.txt` file to specify the packages to include in your SPK.

By default, the file `requirements.txt` searches the PyPI package site for the specified package and version.

- To include a package from a different repository or in a local file path, in the `requirements.txt` file, use the option `-i` or `--index-url`, followed by the location URL.

```
#example
#
mylib -i http://my.domain.org/lib/1.0.0/mylib/
```

- To include a `.whl` package, in the `requirements.txt` file, provide the relative path to the package from the current working directory.


```
#simple-example
#
```

```
./my_path/my_package.whl
packaging==1.0.0
```

If you are building an SPK intended for the Spotfire Service for Python, then you must avoid installing packages that are already included in the installation. Installing a different version of a package that is included in the service installation can cause unexpected errors. If a package you are installing depends on or requires one of the included packages, then it is filtered out when the SPK is built. For a list of included packages, see [Included packages](#) on page 29.

For more information about creating a `requirements.txt` file for your package list, see its documentation at the following location.

The PIP packaging system might deliver a specious warning in the following form:



```
WARNING: Target directory <<temporary location>>/<<package>> already exists. Specify
--upgrade to force replacement.
```

The text of the warning can vary depending on the contents of the third-party wheel packages and operating system. This warning is from the PIP system; for our purposes, it is of no importance and can be disregarded.

- https://pip.readthedocs.io/en/stable/user_guide/#requirements-files
- https://pip.readthedocs.io/en/stable/reference/pip_install/#requirements-file-format

This task creates an SPK for the Spotfire Service for Python, running on a node available to Spotfire Server. If you need to build a package to distribute to installed Spotfire clients, see [Python Data Functions in Spotfire®](#).

Perform this task from a command prompt on the Linux or Windows computer where the installed node manager includes the installation of the Spotfire Service for Python.

Prerequisites

- You must have write access to the computer running the node manager where the Spotfire Service for Python is installed.
- You must have created the file `requirements.txt` containing the list of packages to include in your SPK.

The following example specifies these packages and versions from PyPI.

```
##### example-requirements.txt #####
#
scipy == 1.11.2
matplotlib == 3.9.2
statsmodels == 0.14.0
```

Procedure

1. From the command line, type the command to create the SPK.




```
"%python_service_home%/python/python" -m spotfire.spk packages
[--name "<package-name>"]
<name.spk>
<path-to>requirements.txt
```

`"%python_service_home%"` is the Python Service installation location. On Linux, this is usually `/opt/nodemanager/<server-version#>/nm/services/python-service-linux-<installed-service-guid>`. Specify the values for the installed Python Interpreter `<version#>`, the `package-`

`name`, `name.spk`, and the path to the `requirements.txt` file. See the Options table for more information.



Remember that the path to the Python interpreter has spaces in it, so you must quote the path string.

| Option | Description |
|-------------------------------|--|
| <code>spotfire.spk</code> | <p>The package containing the Python code to download and bundle the needed packages specified in <code>requirements.txt</code>.</p> <div>  <p>From the command prompt, you can view help for the <code>spotfire</code> package by typing the following command:</p> <pre>"<path-to-python-interpreter>\python\python.exe" -m spotfire.spk packages --help</pre> </div> |
| <code>packages</code> | The subcommand that creates the package bundle. |
| <code>--name</code> | <p>A string that sets the internal name of the generated package (for deploying multiple SPK files).</p> <div>  <p>If you do not specify the <code>--name</code> argument, then the package builder reverts to the package name that is embedded in the <code>spotfire.spk</code> stamp in the <code>requirements</code> file (located beneath "BuiltName"), to match the previous version of the package. If the package name is missing entirely, it reverts to "Python Packages Linux" for server packages.</p> </div> |
| <code>--analyst</code> | <p>This option is available, but it is not used in this example.</p> <p>This option specifies that the Spotfire Server should distribute the packages in the SPK to other installed Spotfire clients connected to the Spotfire Server. For more information about using this option, see Python Data Functions in Spotfire®.</p> <div>  <p>Everyone should use the same packages and package versions as those deployed on Spotfire Server.</p> </div> |
| <code>name.spk</code> | The name of the SPK file that is created by this task. |
| <code>requirements.txt</code> | The full path to the file <code>requirements.txt</code> . |

The following example creates an SPK named `my_pkgs.spk`, containing the packages specified in `requirements.txt`.

```
%Python_Service_Home%\python\python"
-m spotfire.spk packages
--name "example-packages"
my_pkgs.spk opt/files/requirements.txt
```

The packages and all of their dependencies are written to the SPK named `my_pkgs.spk` in the current working directory where the command was run, and the version information is recorded in the file `opt/files/requirements.txt`. For example:

```
##### example-requirements.txt #####
#
scipy == 1.11.2
matplotlib == 3.9.2
statsmodels == 0.14.0
## spotfire.spk: {"BrandVersion":2,"Analyst":{"BuiltBy":"3.11.9 (tags/v3
## spotfire.spk: .11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64 bi
## spotfire.spk: t (AMD64)]","BuiltAt":"Mon Aug 21 11:52:29 2023","Built
## spotfire.spk: File":"approved-packages.spk","BuiltName":"Approved Pac
## spotfire.spk: kages","BuiltId":"d9310771-b10a-4395-ac46-1c9f344c2c89"
## spotfire.spk: ,"BuiltVersion":"1.0.0.0","BuiltPackages":{"contourpy":
## spotfire.spk: "1.1.0","cyclor":"0.11.0","fonttools":"4.42.1","kiwisol
```

```
## spotfire.spk: ver:"1.4.4", "matplotlib":"3.9.2", "packaging":"23.1", "p
## spotfire.spk: atsy:"0.5.3", "Pillow":"10.0.0", "pyparsing":"3.0.9", "sc
## spotfire.spk: ipy:"1.11.2", "statsmodels":"0.14.0", "tzdata":"2023.3"}
## spotfire.spk: }, "Server":{}}
```

2. Locate the SPK you created in the working directory where you ran the command.
3. Add the SPK to the Spotfire Server Deployment area, and then validate and save the area.

Result

The packages are added to the Spotfire Server node manager, where Spotfire users can access analyses that use the functions in the packages from their web browsers.

For an overview of the entire process, see Spotfire Server.

Creating a Spotfire package for Python packages from a Windows computer

The installed Spotfire client includes a Python interpreter and a set of packages to enable using Python in Spotfire.

One of these packages, 'spotfire', provides tools for building SPKs to share Python packages with other data function authors in an organization, or to enable web client users to see Python-enabled analyses in a web browser. For more information about package workflow, see [Package management for the Spotfire Service for Python](#) on page 27. If you are using a Python interpreter other than the one provided with your Spotfire installation, then you must first run the following command:

```
-m pip install spotfire
```



Important You must build a package for the operating system on which your Spotfire Server node manager is installed. This topic describes building the SPK using the spotfire package supplied with your Windows installation of Spotfire. If you are installing the SPK on a Linux node, see [Creating a Spotfire package for Python packages on the node manager](#) on page 36.

The installed Spotfire client relies on pip, the Python command-line application for Python package installation. The installed Spotfire client uses a `requirements.txt` file to specify the packages to include in your SPK. By default, the file `requirements.txt` searches the PyPI package site for the specified package and version.

- To include a package from a different repository or in a local file path, in the `requirements.txt` file, use the option `-i` or `--index-url`, followed by the location URL.

```
#example
#
mylib -i http://my.domain.org/lib/1.0.0/mylib/
```

- To include a `.whl` package, in the `requirements.txt` file, provide the relative path to the package from the current working directory.

```
#simple-example
#
./my_path/my_package.whl
packaging==1.0.0
```

The installed Spotfire client relies on the Python packages included in the installation. Removing any of these packages causes your Spotfire installation to not work with the included Python interpreter. For a list of included packages, see [Included packages](#) on page 29.

If you are building an SPK intended for the Spotfire Service for Python, then avoid specifying packages that are included with the service installation. Installing a different version of one of these packages can cause unexpected errors. If a package you are installing depends on or requires one of the included packages, then it is filtered out when the SPK is built.

The PIP packaging system might deliver a specious warning in the following form:

WARNING: Target directory <<temporary location>>\<<package>> already exists. Specify --upgrade to force replacement.

The text of the warning can vary depending on the contents of the third-party wheel packages and operating system. This warning is from the PIP system; for our purposes, it is of no importance and can be disregarded.

For more information about creating a `requirements.txt` file for your package list, see its documentation at the following location.

- https://pip.readthedocs.io/en/stable/user_guide/#requirements-files
- https://pip.readthedocs.io/en/stable/reference/pip_install/#requirements-file-format

This task creates an SPK for the Spotfire Service for Python, running on a node available to Spotfire Server. If you need to build a package to distribute to installed Spotfire clients, see [Python Data Functions in Spotfire®](#).

Perform this task from a command prompt on the Windows computer where Spotfire is installed.

Prerequisites

- You must have the appropriate Spotfire license for authoring data functions.
- You must have created the file `requirements.txt` containing the list of packages to include in your SPK.

The following example specifies these packages and specified versions from PyPI.

```
##### example-requirements.txt #####
#
scipy == 1.11.2
matplotlib == 3.9.2
statsmodels == 0.14.0
```

Procedure

1. From the command line, type the command to create the SPK.


For our example, use the following command, specifying values for the installed Python Interpreter `<version#>`, the `package-name`, `name.spk`, and the path to the `requirements.txt` file. See the Options table for more information.



```
"%SPOTFIRE_HOME/Modules/Python Interpreter_<version#>/python/python.exe"
-m spotfire.spk packages
[--name "<package-name>"]
<name.spk>
<path-to>/requirements.txt
```



Remember that the path to the Python interpreter has spaces in it, so you must quote the path string.

Table

| Option | Description |
|--------------|---|
| spotfire.spk | <p>The package containing the Python code to download and bundle the needed packages specified in <code>requirements.txt</code>.</p> <div>  <p>From the command prompt, you can view help for the spotfire package by typing the following command:</p> <pre>"<path-to-python-interpreter>\python\python.exe" -m spotfire.spk packages --help</pre> </div> |

| Option | Description |
|----------------------------|--|
| packages | The subcommand that creates the package bundle. |
| --name | <p>A string that sets the internal name of the generated package (for deploying multiple SPK files).</p> <p> If you do not specify the --name argument, then the package builder reverts to the package name that is embedded in the spotfire.spk stamp in the requirements file (located beneath "BuiltName"), to match the previous version of the package. If the package name is missing entirely, it reverts to "Python Packages Windows" for server packages.</p> |
| --analyst | <p>This option is available, but it is not used in this example.</p> <p>This option specifies that the Spotfire Server should distribute the packages in the SPK to other installed Spotfire clients connected to the Spotfire Server. For more information about using this option, see Python Data Functions in Spotfire®.</p> <p> Everyone should use the same packages and package versions as those deployed on the Spotfire Server.</p> |
| name.spk | The name of the SPK file that is created by this task. |
| <path-to>/requirements.txt | The full path to the file requirements.txt. |

The following example creates an SPK named `my_pkgs.spk`, containing the packages specified in the example `requirements.txt`.

```
%SPOTFIRE_HOME%\Modules\Python Interpreter_3.12.9.xx\python\python.exe" -m spotfire.spk
packages --name "example-packages" my_pkgs.spk c:\files\requirements.txt
```

The packages and all of their dependencies are written to the SPK named `my_pkgs.spk` in the current working directory where the command was run, and the version information is recorded in the file `requirements.txt`. For example:

```
##### example-requirements.txt #####
#
scipy == 1.11.2
matplotlib == 3.9.2
statsmodels == 0.14.0
## spotfire.spk: {"BrandVersion":2,"Analyst":{"BuiltBy":"3.11.9 (tags/v3
## spotfire.spk: .11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64 bi
## spotfire.spk: t (AMD64)]","BuiltAt":"Mon Aug 21 11:52:29 2023","Built
## spotfire.spk: File":"approved-packages.spk","BuiltName":"Approved Pac
## spotfire.spk: kages","BuiltId":"d9310771-b10a-4395-ac46-1c9f344c2c89"
## spotfire.spk: ,"BuiltVersion":"1.0.0.0","BuiltPackages":{"contourpy":
## spotfire.spk: "1.1.0","cyclor":"0.11.0","fonttools":"4.42.1","kiwisol
## spotfire.spk: ver":"1.4.4","matplotlib":"3.9.2","packaging":"23.1","p
## spotfire.spk: atsy":"0.5.3","Pillow":"10.0.0","pyparsing":"3.0.9","sc
## spotfire.spk: ipy":"1.11.2","statsmodels":"0.14.0","tzdata":"2023.3"}
## spotfire.spk: },"Server":{}}
```

2. Locate the SPK you created in the working directory where you ran the command.
3. Add the SPK to the Spotfire Server Deployment area, and then validate and save the area.

Result

The packages are added to the Spotfire Server node manager, where Spotfire users can access analyses that use the functions in the packages from their web browsers.

For an overview of the entire process, see [Package management for the Spotfire Service for Python](#) on page 27.

Build a Spotfire package for Spotfire Service for Python (Linux)

The installation of the Spotfire Service for Python on your Spotfire Server includes a dockerfile that you can use to deploy a collection of Python packages to a Spotfire Server and node manager running on the Linux operating system.

If you do not have login credentials for the Spotfire Server running on Linux, or if you have a large deployment of node managers running the Spotfire Service for Python on Linux, then it might not be possible to install packages directly on the server. In this case, you can build your Spotfire package (SPK) containing the packages to distribute using a Docker container.



debian:12-slim is our preferred Linux distribution for use with the Spotfire Service for Python.

This version of the Spotfire Service for Python includes dockerfiles that you can use to build an SPK for packages to use with the Spotfire Service for Python. You can build the SPK from either Windows or Linux.

Table

| Tool | Description |
|-------------------|--|
| python-build-base | Contains a full Python build environment including all necessary dependencies. |
| spotfire-spk | Used to call the python-build-base container to build the interpreter or the SPKs containing the packages for the Spotfire Service for Python. |

Building a Spotfire package (SPK) for Python packages from a Docker image on Windows

You can build a Spotfire package (SPK) using a Docker image on Windows, and then place the SPK on a Spotfire Server running on Linux.

Perform this task from the command line on a Windows computer.

Prerequisites

Before you begin, complete the following tasks.

1. Locate and download Spotfire® Statistics Services for your operating system on the [Spotfire Download site](#) (license and account required). The product Spotfire® Statistics Services includes the Spotfire Service for Python.
2. In the Spotfire Statistics Services download bundle, find the component Spotfire Service for Python.
3. Extract the contents of the Spotfire Service for Python installation archive.

Procedure

1. Create the file `requirements.txt` and place it in the same directory from where you are running the `spotfire-spk` script.

The file `requirements.txt` contains a list of packages, with their version numbers, that are included in the SPK. For example:

```
##### example-requirements.txt #####
#
scipy == 1.11.2
matplotlib == 3.9.2
statsmodels == 0.14.0
```

2. Call the following commands:

```
set PYTHON_VERSION=3.12.9
```

```
docker build --build-arg PYTHON_VERSION -t python-build-base:%PYTHON_VERSION% python-build-base
```

3. Build the SPK.

```
spotfire-spk <python-build-base-tag> [arguments to spotfire.spk module]
spotfire-spk 3.12.9 packages python-packages.spk requirements.txt
```

The file `python-packages.spk` containing the packages specified in `requirements.txt` is created.

What to do next

Add the SPK to the Spotfire Server Deployment area, and then validate and save the area. See [Adding Software Packages to a Deployment Area](#) and [Updating Services](#) in the *Spotfire® Server Installation and Administration* guide for more information. If you have specified the `--analyst` flag in the arguments to the `spotfire.spk` module, then the next time the installed Spotfire client users connect to the Spotfire Server, they are prompted to update their installations with the new packages. Business authors and consumers connecting to Spotfire from a web browser can use analyses with data functions that use the functions in the packages.

Building a Spotfire package (SPK) for Python packages with a Docker image on Linux

You can build a Spotfire package (SPK) using a Docker image on Linux, and then place the SPK on a Spotfire Server running on Linux.

Perform this task from the command line on a Linux computer.

Prerequisites

Before you begin, complete the following tasks.

1. Locate and download Spotfire® Statistics Services for your operating system on the [Spotfire Download site](#) (license and account required). The product Spotfire® Statistics Services includes the Spotfire Service for Python.
2. In the Spotfire Statistics Services download bundle, find the component Spotfire Service for Python.
3. Extract the contents of the Spotfire Service for Python installation archive.

Procedure

1. Create the file `requirements.txt` and place it in the same directory from where you are running the `spotfire-spk` script.

The file `requirements.txt` contains a list of packages, with their version numbers, that are included in the SPK. For example:

```
##### example-requirements.txt #####
#
scipy == 1.11.2
matplotlib == 3.9.2
statsmodels == 0.14.0
```

2. Call the following commands:

```
export PYTHON_VERSION=3.12.9
docker build --build-arg PYTHON_VERSION -t python-build-base:${PYTHON_VERSION} python-build-base
```

3. Build the SPK using the following commands.

```
./spotfire-spk <python-build-base-tag> [arguments to spotfire.spk module]
```

EXAMPLE:

```
./spotfire-spk 3.12.9 packages python-packages.spk requirements.txt
```

The file `python-packages.spk` containing the packages specified in `requirements.txt` is created.

What to do next

Add the SPK to the Spotfire Server Deployment area, and then validate and save the area. See [Adding Software Packages to a Deployment Area](#) and [Updating Services](#) in the *Spotfire® Server Installation and Administration* guide for more information. If you have specified the `--analyst` flag in the arguments to the `spotfire.spk` module, then the next time the Spotfire Analyst users connect to the Spotfire Server, they are prompted to update their installations with the new packages. Business authors and consumers connecting to Spotfire from a web browser can use analyses with data functions that use the functions in the packages.

Use an alternative Python interpreter

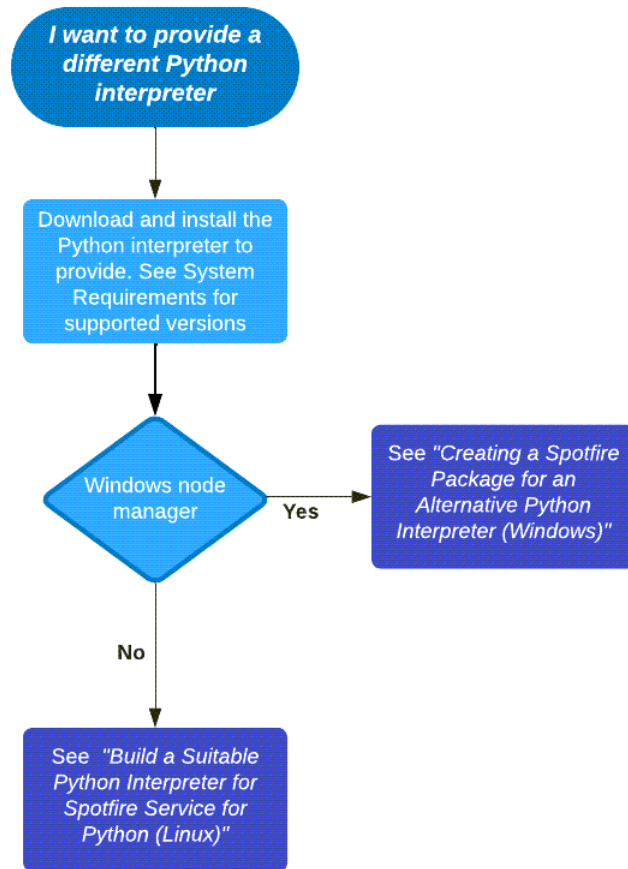
You can use a Python interpreter that is different from the one provided with your Spotfire installation by uploading it in a Spotfire package.

You can create a Spotfire package (SPK) containing an alternative Python interpreter from either a Windows computer or a Linux computer. You can then upload the SPK to the Spotfire Server deployment area for distribution to the computers running the node managers, or for distribution to users connected to the Spotfire Server running Spotfire Analyst.

- If your node manager is running on a Windows server, then you can create an SPK containing the Python interpreter from a Windows computer where the alternative Python interpreter is installed.
- If your node manager is running on a Windows server, but you want to keep your installation of Python "pristine", then you can create a virtual environment where you can create the SPK for the alternative interpreter.
- If your node manager is running on a Linux server, then you can create the SPK containing the alternative Python interpreter from the Linux computer that is running the node manager, and where both the default Python interpreter and the alternative Python interpreter are installed. You can also create the SPK from a Windows computer and then deploy it to your Spotfire Server for distribution to your node manager running on Linux.

This section provides instruction for packaging and uploading the alternative Python interpreter for all of those cases.

Each Help reference is a link to the topic. Click the square to see more information.



Creating a Spotfire package for an alternative Python interpreter (for Windows)

The installation of the Spotfire Service for Python on your Spotfire Server includes a Python interpreter and a set of packages to enable using Python. The installed Spotfire client that connects to the Spotfire Server also includes the same version of a Python interpreter. You can provide a different version of the Python interpreter, if needed, on both the Spotfire Server and installed Spotfire clients.

To install an alternative version of the Python interpreter on both the Spotfire Server and in the installed Spotfire client, create a Spotfire package (SPK) containing the Python interpreter for each of the installations, and then upload the SPKs to the Spotfire Server.



Remember that the same version of the Python interpreter must run on both the Spotfire Server and the computers running Spotfire clients that connect to it.

Perform this task on a Windows computer where the Python interpreter that you want to distribute is installed.

Building an SPK on a Windows computer creates an SPK that works only on a Spotfire Server node manager installed on a Windows server. If you are running a Spotfire Server node manager on a Linux server, see [Build a suitable Python interpreter for Spotfire Service for Python \(for Linux\)](#) on page 47

This procedure describes installing the 'spotfire' package into the Python interpreter that you want to distribute. Alternatively, you can create a virtual environment to keep your installation of Python pristine, in the case where you either do not want to (or cannot) install the 'spotfire' package into the Python interpreter. See [Creating a Virtual Environment for an Alternative Python Interpreter](#) for those instructions.

Prerequisites

Before you begin, complete the following tasks.

1. Download and install a suitable 64-bit interpreter. Python 3.9 or higher is required. Make sure it is configured to work correctly with your system. (The steps in this task demonstrate building the Python interpreter using Python version 3.12.9.)
2. Locate and download Spotfire® Statistics Services for your operating system on the product download site [Spotfire Download site](#) (license and account required). The product Spotfire® Statistics Services includes the Spotfire Service for Python.
3. In the Spotfire Statistics Services downloaded bundle, find the component Spotfire Service for Python.
4. Extract the contents of the Spotfire Service for Python installation archive.

Procedure

1. Install the 'spotfire' package into your Python interpreter.

```
path/to/python/to/package/ python -m pip install spotfire
```

Where *path/to/python/to/package* is the path to the Python interpreter to include in the package.



This also downloads and installs the other packages required by the Spotfire Service for Python.

2. Build the Python interpreter SPK for the Spotfire Server.

```
path/to/python/to/package/python -m spotfire.spk python name1.spk
```

This command creates an SPK to install the Spotfire Service for Python on the node manager. See the table in Step 3 for more information about the options.




Provide a meaningful name for the .spk file so you can find it easily and distinguish it from the one you create in Step 3.


3. Build the Python interpreter SPK for the installed Spotfire clients.

```
path/to/python/to/package/python -m spotfire.spk python --analyst name2.spk
```



Provide a meaningful name for the .spk file so you can find it easily and distinguish it from the one you create in Step 2.

| Option | Description |
|--------------|---|
| spotfire.spk | <p>The package containing the Python code to download and bundle the Python interpreter.</p> <div>  <p>From the command prompt, you can view help for the spotfire package by typing the following command:</p> <pre>"<path-to-python-interpreter>\python\python.exe" -m spotfire.spk packages --help</pre> </div> |
| python | Specifies that the SPK contains the Python interpreter from which you issued the command. |
| --analyst | Specifies that the Spotfire Server should distribute the Python interpreter in the SPK to other installed Spotfire clients connected to the Spotfire Server. |

| Option | Description |
|---------------|--|
| nameX.spk | The name of the SPK file that is created by this task. |
| --constraints | <p>Optional. Constrains the required package versions to those listed in the file <code>interpreter-constraints.txt</code>, which is included with your distribution of Spotfire Service for Python. Provide the complete path to the file. For example:</p> <pre>--constraints C:/files/interpreter-constraint.txt</pre> <div>  <p>Important If you omit this option, the required packages included might be updated to versions newer than the ones included with the Spotfire Service for Python. The included package versions are tested with Spotfire. While typically Python package authors take care for backwards compatibility, it is possible that a change to one of these packages could result in your data functions failing to work correctly. Spotfire Support can help you if you encounter such a breaking issue. See Included packages on page 29 for more information.</p> </div> |

Result

The packages containing the Python interpreter to upload to the Spotfire Server can be found in the `.spk` files that you created.

What to do next

Add the SPKs to the Spotfire Server Deployment area, and then validate and save the area. See [Adding Software Packages to a Deployment Area](#) and [Updating Services](#) in the *Spotfire® Server Installation and Administration* guide for more information. The next time the installed Spotfire client user connects to the Spotfire Server, they are prompted to update their installations with the new Python Interpreter and packages.

Build a suitable Python interpreter for Spotfire Service for Python (for Linux)

The installation of the Spotfire Service for Python on your Spotfire Server includes a Python interpreter and a set of packages to enable using Python. The installed Spotfire client also includes the same version of a Python interpreter. You can provide a different version of the Python interpreter, if needed, on both the Spotfire Server and the installed Spotfire client.

Most OS-provided Python interpreters on Linux are not well-suited to use with the Spotfire Service for Python. If you want to use another Python interpreter on your Spotfire Server node manager on Linux, then for best compatibility and results, build your Python interpreter from source using a Docker image that closely reflects the runtime environment of the Spotfire Service for Python.



debian:12-slim is our preferred Linux distribution for use with the Spotfire Service for Python.

This version of the Spotfire Service for Python includes dockerfiles that you can use to build a fully-compatible Python interpreter, or you can build an SPK for packages to use with the Spotfire Service for Python. You can build the Python interpreter from either Windows or Linux.

Important When you build the SPK containing a different Python interpreter, include the `--constraints` option to use the packages specified in the file `interpreter-constraints.txt`, which is included with your distribution of the Spotfire Service for Python. Provide the complete path to the file. For example:



```
--constraints /opt/files/interpreter-constraint.txt
```

If you omit this option, the required packages might be updated to versions newer than the ones included with the Spotfire Service for Python. The included package versions are tested with Spotfire. While typically Python package authors take care for backwards compatibility, it is possible that a change to one of these packages could result in your data functions failing to work correctly. Spotfire Support can help you if you encounter such a breaking issue.

Table

| Tool | Description |
|--------------------------------|--|
| <code>python-build-base</code> | Contains a full Python build environment including all necessary dependencies. |
| <code>python-base</code> | Contains all runtime libraries required to support Python packages. |
| <code>spotfire-spk</code> | Used to call the <code>python-build-base</code> container to build the SPK containing the Python interpreter, and to build the SPK containing the required packages for the Spotfire Service for Python. |

Service resource management scenarios

You can use a combination of the Spotfire Service for Python custom properties, including the pruning properties `engine.prune` and `dynamic.prune.threshold`, to ensure the best usage of the Python engines that the Spotfire Service for Python allocates.

The custom properties `engine.session.max` and `engine.queue.size` determine the number of engines that are available, and the number of engines allowed in the queue, respectively. These values are determined by the number of logical processors available on the node where the Spotfire Service for Python is running. Additionally, you might want to set properties that control how long a Python engine in a session can remain idle, how long to run an execution before timing out, or the percentage of engines that can run in a session before pruning is triggered.

The following two configuration examples, and their associated scenarios, demonstrate the resource management for different combinations of custom properties. These non-exhaustive usage scenarios are provided only to give two of many configurations for engine pruning and engine idle timeout. Your needs can vary, depending on your job sizes, the number of users, and the number of available logical processors.

Configuration A

Assume the following configuration values, where three engines are created and waiting in the queue for jobs.

```
#Configuration A
engine.execution.timeout: 60
engine.session.maxtime: 120
# by default, these are set to number of logical processors, minus 1, on the system
engine.session.max: 3
engine.queue.size: 3
# the idle timeout
engine.prune: 10
# The service capacity at which idle pruning is engaged, as a percentage value.
# 0 = always idle prune.
# 100 = never idle prune.
dynamic.prune.threshold: 100
```

The following three scenarios show how this configuration affects the jobs that users submit.

| Scenario | Result |
|--|---|
| 1A: A single user submits a job that runs for more than 60 seconds | Because this job runs for longer than the value set for <code>engine.execution.timeout</code> , the execution is halted and the engine is destroyed. Results for this long job are not returned. When the user submits another job, a new engine is provided from the queue. |
| 2A: A single user submits a job that runs for 5 seconds | The job completes and returns results, and the engine persists. The capacity of the service is equal to the number of engines in use divided by the maximum sessions. In this case, the capacity is 1/3, or 33%, which is below the <code>dynamic.prune.threshold</code> value of 100. The user can access this same engine for up to <code>engine.session.maxtime</code> (in this case 120 seconds). |
| 3A: Four users submit jobs that execute for 5 seconds each | The first three user jobs get engines, and the fourth user job must wait for an engine to become available, because the <code>engine.session.max</code> has been reached. Because <code>dynamic.prune.threshold</code> is set to 100, which specifies never idle prune, the first three users claim their engines for the duration of the value of the <code>engine.session.maxtime</code> of 120 seconds. At this point, the engines are destroyed and new engines are made available on a first-come-first-served basis. |

Configuration B

Change the configuration as follows, where only the `dynamic.prune.threshold` has been changed from 100 to 0.

```
#Configuration B
engine.execution.timeout: 60
engine.session.maxtime: 120
# by default, these are set to number of logical processors on the system
engine.session.max: 3
engine.queue.size: 3
# the idle timeout
engine.prune: 10
# The service capacity at which idle pruning is engaged, as a percentage value.
# 0 = always idle prune.
# 100 = never idle prune.
dynamic.prune.threshold: 0
```

The same user scenarios show how this changed configuration affects the jobs that users submit.

| Scenario | Result |
|--|--|
| 1B: A single user submits a job that runs for more than 60 seconds | No change in behavior from Configuration A. |
| 2B: A single user submits a job that runs for 5 seconds | The job completes and return results. The capacity is at 33 which is now higher than the <code>dynamic.prune.threshold</code> of 0, so the engine is destroyed after the <code>engine.prune</code> (idle timeout) of 10 seconds. If the user submits a new job, a new engine is created from the queue. |
| 3B: Four users submit jobs that execute for 5 seconds each | The first three user jobs get engines, and the fourth user job must wait for an engine to become available, because the <code>engine.session.max</code> has been reached. However, <code>dynamic.prune.threshold</code> is set to always idle prune (0), so after submitting a job and getting results, if a user sits idle for longer than <code>engine.prune</code> of 10 seconds, the engine is destroyed. More engines are created and made available to the fourth user. |

Conclusion


In scenario 3A (where four users submit jobs, and the `dynamic.prune.threshold` is set to 100), the fourth user might have to wait for up to 2 minutes for an available engine (the `engine.session.maxtime`), whereas in scenario 3B (where four users submit jobs, and the `dynamic.prune.threshold` is set to 0), the fourth user could wait for just 15 seconds (job run of 5 seconds and `engine.prune` idle timeout of 10 seconds).

By default, the `dynamic.prune.threshold` is set to 60, because this setting balances both access for a high volume of users and faster response times for a lower volume of users. The default values for `engine.execution.timeout` and `engine.session.maxtime` are set to balance security and availability. For your on-premises usage, you might find it useful to increase execution timeouts or disable idle timeouts altogether.

See [Engine pruning](#) on page 22 for more information about these custom properties. See [Configuring the Service](#) on page 18 for information about setting all custom properties.

Service logs

After the Spotfire Service for Python is installed and started, it begins writing to the logs. These logs are stored in the directory `<node manager installation>/logs`.

| Log name | Description |
|--|---|
| <code>service-<instanceid>-stdout.log</code> | <p>Prints INFO-level information about the service startup, shut down, and any exceptions. It also prints the Jetty component to the standard log files.</p> <div>  <p>To set Jetty logging to DEBUG level, in the startup script (<code>start-py-service.sh</code> for Linux nodes, <code>startPy.bat</code> for Windows nodes), uncomment the command <code>SET JETTY_DEBUG=-Dorg.eclipse.jetty.LEVEL=DEBUG</code>.</p> </div> |
| <code>python-service-<instanceid>.log</code> | <p>Prints configuration options that the Spotfire Service for Python starts with. Provides granular level of the individual engines, job execution details.</p> |

Monitoring the service using JMX

The Spotfire Service for Python supports JMX monitoring integration. JMX monitoring is turned off by default.

Prerequisites

You can install and use JConsole for monitoring the Spotfire Service for Python using JMX. JConsole is provided as part of the Java SE Development Kit. Alternatively, you can install and use VisualVM to monitor the Spotfire Service for Python using JMX.



Important Because JMX monitoring requires connecting to the specific IP address of the node, you must create a custom configuration for each node to monitor.

Procedure

1. Stop the Spotfire Service for Python.
2. Export and edit the `custom.properties`, setting the following properties:

```
jmx.rmi.username: username
jmx.rmi.password: password
jmx.rmi.host: <IP address of the Node Manager running Spotfire Service for R>
jmx.rmi.port: 1099
jmx.active: TRUE
```

See [Configuring the Service](#) on page 18 for detailed instructions.

3. Start the Spotfire Service for Python.
4. Check the `INFO` logs for the connection string.

If the setup and connection are successful, a JMX connection string is printed to logs at the `INFO` level.

```
2022-06-09T21:03:11,520 | INFO | [main] c.s.s.t.ServiceConfig: Service configured JMX Connection
string: service:jmx:rmi://10.10.100.60:1099/jndi/rmi://10.10.100.60:1099/jmxrmi
```

- If `jmx.rmi.username`, `jmx.rmi.password`, or `jmx.rmi.host` are blank, then a log message is printed indicating that the property is blank, and that the JMX connection is not created.
- If `jmx.rmi.port` is blank or undefined, then the port value defaults to 1099.
- If the `jmx.rmi.host` is configured incorrectly, the connection times out and the service fails to start. An error message is printed to the admin UI and the logs.

A successful client connection is printed to logs at the `DEBUG` level. An unsuccessful client connection attempt due to bad or missing username or password is printed at the `ERROR` level.



If you are connecting to a remote host, the port must be opened in the firewall to allow the connection.

5. Open JConsole, and in the **remote process** field, provide the JMX connection string provided by the logs as shown:

```
service:jmx:rmi://10.10.100.60:1099/jndi/rmi://10.10.100.60:1099/jmxrmi
```

6. Provide the user name and password that you set in `custom.properties`.



If a message is shown indicating the connection could not be made using SSL. Would you like to try without SSL?, then click **Insecure connection**.

JConsole should now show information from the service.

7. To view metrics specific to the Spotfire Service for Python, click the tab **MBeans**.

8. In the left panel, expand the group labeled **metrics**.

Metrics are listed, including many JVM metrics. Some of the metrics specific for Python are as follows:

```
serviceQueueCurrentSize - total number of engines currently waiting in the queue
serviceQueueEnginesDestroyed - total number of engines destroyed after successful use
serviceQueueEnginesFailed - total number of engines that failed on startup due to
    configuration, environmental, or other exceptions
serviceQueueEnginesInUse - total number of engines currently executing
serviceQueueEnginesStarted - total number of successful engines started
serviceQueueEnginesStarting - total number of engines currently initializing
serviceQueueIdealSize - the ideal queue size as defined by engine.queue.size in
    custom.properties
serviceQueueLastPortSelected - the last port chosen for engine creation
serviceUsageBytesDownloaded - total bytes downloaded through the service
serviceUsageBytesUploaded - total bytes uploaded through the service
serviceUsageCapacity - the current capacity of the service as a percentage: current
    session over maximum allowed concurrent sessions
serviceUsageJobs - total number of jobs the service has created and run
serviceUsageSessions - total number of sessions the service has created
serviceUsageMillisInUse - total time spent executing successful jobs, in milliseconds
```

Troubleshooting the service

If you have problems with the service, review these tips.

Problems with the startup script

Check your script line endings.



Important Remember that for any script you write, the line endings must be appropriate for the operating system where the service runs. Many text editors can perform end-of-line (EOL) conversion.

Packages installed on a node manager running a different operating system than the engine

This unlikely case can happen if, for example, you are running the `debian:12-slim` Docker image for your Spotfire Service for Python, but your node manager computer is running Red Hat Enterprise Linux (RHEL) 9. Because of symbol version issues, packages must be installed in an equivalent OS to the execution environment. If this is not possible, try one of the following workarounds:

- Install the packages on a computer running the same operating system as your execution environment, and then copy those packages to the node manager running the other operating system. Provide the path in `custom.properties` file for the configuration `packagepath`.
- Create a customized image with `debian:12-slim`, and then install the required packages in the image. Provide the image name in the `custom.properties` file in the configuration setting `docker.image.name`.

Empty String columns in exported SBDF causes an error

When you export an SBDF from Python (for example, as an output for a data function), and your output contains an empty column, you can encounter the following error:

```
spotfire.sddf.SBDFError: cannot determine type for column 'EmptyString'; all values are missing
```

This error occurs because the Spotfire data function environment cannot determine the proper Spotfire type to export the data as if all values in the column are missing (in other words, Python's `None`, NumPy's `nan`, or Panda's `NA` or `NaT` values).

To resolve this issue, edit your data function to use the helper function `set_spotfire_types`.

Example

In the following example, the variable name `EmptyString` refers to a string column that contains no values. (For your export, use the actual empty column name.) In this example, the function `spotfire.set_spotfire_types` sets the column specifically to the Spotfire data type `String`.

```
import pandas as pd
import spotfire

outp = pd.DataFrame(inp)
spotfire.set_spotfire_types(outp, {'EmptyString': 'String'})
```

Memory usage

The maximum memory needed to run a Python script can exceed the `javaOptions` setting, depending on the size of the data, the Python packages used, and how the Python script is written.



The `javaOptions` setting controls Java memory only, and does not control memory allocated in the Python runtime. (Memory allocated by Java is typically very small, which makes it possible to set a very low value for the initial heap size, such as `Xms64M`.)

If you encounter memory issues:

- Try running your script using a stand-alone Python engine to gauge its memory usage. You should see no difference between the memory needed for running your script in a stand-alone Python engine and running under the Spotfire Service for Python.
- If you are running a containerized service, you can set a limit on the memory usage for the engine process. See the example under `ram.limit` in [Containerized configuration](#) on page 26.

- Use utility functions such as `memAvailable` to find, report, and then reallocate available memory.

Spotfire Documentation and Support Services

For information about the Spotfire® products, you can read the documentation, contact Spotfire Support, and join the Spotfire Community.

How to Access Spotfire Documentation

Documentation for Spotfire and TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The website is updated frequently and is more current than any other documentation included with the product.

Spotfire Documentation

The documentation for all Spotfire products is available on the [Spotfire Documentation](#) page. This page takes you directly to the latest version of each document.

To see documents for a specific Spotfire product or version, click the link of the product under 'Other versions', and on the product page, choose your version from the top right selector.

Release Version Support

Some release versions of Spotfire products are designated as long-term support (LTS) versions. LTS versions are typically supported for up to 36 months from release. Defect corrections will typically be delivered in a new release version and as hotfixes or service packs to one or more LTS versions. See also <https://spotfi.re/lts>.

How to Contact Support for Spotfire Products

You can contact the Support team in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the support portal at <https://spotfi.re/support>.
- For creating a Support case, you must have a valid maintenance or support contract with Cloud Software Group, Inc. You also need a user name and password to log in to <https://spotfi.re/support>. If you do not have a user name, you can request one by clicking **Register** on the website.

System Requirements for Spotfire Products

For information about the system requirements for Spotfire products, visit <https://spotfi.re/sr>.

How to join the Spotfire Community

The Spotfire Community is the official channel for Spotfire customers, partners, and employee subject matter experts to share and access their collective experience. The Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from Spotfire products. In addition, users can submit and vote on feature requests from within the [Ideas Portal](#). For a free registration, go to <https://spotfi.re/community>.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. ("CLOUD SG") SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, "INCLUDED SOFTWARE"). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

Spotfire, the Spotfire logo, TERR, and TIBCO are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries. A list of Cloud SG's trademarks and trademark guidelines is available at <https://www.cloud.com/legal>.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG's Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the Apache License, Version 2.0, which is available at <https://www.apache.org/licenses/LICENSE-2.0> and reprinted in the Addendum below.

Copyright (c) Christian Robertson / Google, Roboto font.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the "readme" file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.tibco.com/patents>.

Copyright © 2017-2025 Cloud Software Group, Inc. All Rights Reserved.

Addendum to Legal and Third-Party Notices

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions

for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

Index

C

compression [20](#)
 configuration [18](#)
 container [6](#)
 custom package list [28](#)

D

data function [6](#), [20](#)
 disable.spotfire.trust.checks [25](#)
 disable.warning.alert [21](#)
 Docker [6](#)
 docker.image.name [21](#), [26](#)
 documentation [28](#)
 dynamic.prune.threshold [22](#)

E

engine.execution.timeout [22](#)
 engine.port.min [21](#)
 engine.port.range [21](#)
 engine.prune [22](#)
 engine.queue.size [19](#)
 engine.session.max [19](#)
 engine.session.maxtime [22](#)
 environment variable for Python home [20](#)
 error [54](#)

F

file size [22](#)

H

hard disk [4](#)
 help [28](#)

J

javaOptions [23](#)
 Jetty logging [51](#)
 jetty.gzip.comression-level [20](#)
 jetty.gzip.inflate-buffer-size [20](#)
 jetty.gzip.min-gzip-size [20](#)
 JMX [26](#), [49](#), [52](#)
 jmx.active [26](#), [49](#)
 jmx.rmi.host [26](#), [49](#)
 jmx.rmi.password [26](#), [49](#)
 jmx.rmi.port [26](#), [49](#)
 jmx.rmi.username [26](#), [49](#)

L

Linux system Python, using [20](#)
 Linux versions [4](#)
 logging [51](#)
 loggingLevel [23](#)

M

memory [54](#)
 monitoring [52](#)

O

options [18](#)

P

packagePath [24](#)
 packages [27](#)
 PyPI [27](#)

R

ram.limit [26](#)
 RMI [26](#), [49](#)

S

service-<guid>-stdout.log [51](#)
 service-<service-ID>-stdout.log [51](#)
 SPK [32](#)
 SPOTFIRE_PYTHON_HOME environment variable [20](#)
 Spring [22](#)
 spring.servlet.multipart.max-file-size [22](#)
 spring.servlet.multipart.max-request-size [22](#)
 startup.hook.script [25](#)
 system Python [20](#)

T

timeout [22](#)
 troubleshooting [51](#)

U

upload file [22](#)
 use.engine.containers [6](#), [26](#)
 use.immutable.container [21](#)

W

Windows version [4](#)