TIBCO Spotfire® Service for Python Installation and Administration

Software Release 1.0 - LTS
June 2020

TIBCO®

Important Information

Contents

Documentation and support services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit https://docs.tibco.com.

Python - Server Edition documentation

The following documents for the TIBCO Spotfire® Service for Python can be found on the TIBCO Documentation website.

- *TIBCO® Spotfire Service for Python Installation and Administration*
- *TIBCO® Spotfire Service for Python Release Notes*

TIBCO Spotfire Server Documentation

The following documents for TIBCO Spotfire® Server can be found on the TIBCO Product Documentation website.

- *TIBCO Spotfire® Server and Environment - Installation and Administration*
- *TIBCO Spotfire® Server and Environment - Quick Start Guide*
- *TIBCO Spotfire® Cobranding*
- *TIBCO Spotfire® Server Release Notes*
- *TIBCO Spotfire® Server Web Services API Reference*
- *TIBCO Spotfire® Server Server Platform API Reference*
- *TIBCO Spotfire® Server Information Services API Reference*
- *TIBCO Spotfire® Server REST API Reference*
- *TIBCO Spotfire® Server License Agreement*

Product System Requirements

For a list of system requirements for this product and other Spotfire products, see the TIBCO Spotfire® Products System Requirements.

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit http://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to https://support.tibco.com. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO 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 TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to https://community.tibco.com.

For quick access to TIBCO Spotfire content, see https://community.tibco.com/products/spotfire.

The TIBCO Spotfire® Service for Python

Spotfire® Service for Python extends access to the Python® language from the TIBCO Spotfire® Analyst (Windows client) to TIBCO Spotfire® Business Author users.

Spotfire Service for Python provides robust predictive and computational detail to users who access Spotfire visualizations through the Spotfire Business Author or Spotfire Consumer.

To develop and share analyses using Spotfire Service for Python, analysts must have an installation of Spotfire® Analyst, version 10.10 or later.

Spotfire Service for Python is provided to TIBCO 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 Spotfire Server.

This help for Spotfire Service for Python is for Spotfire Server administrators who need to install and configure the Spotfire Service for Python, to install Python packages for visualizations that use them, and to review logs and troubleshoot the Spotfire Service for Python.

System Requirements and Compatibility

Spotfire Service for Python version 1.0.0 system requirements and third-party compatibility information are listed here.

Software

Software	Version
Docker (LINUX only; optional) The service builds a default Docker image based on Centos 7, which is also available from Docker Hub. (The recommended Docker version, 17.12.1-ce, was tested with Centos 7.4 and higher.) You cannot modify the image we provide. You can provide another prebuilt image. Check Docker Hub for an image that might work for you.	17.03.1-ce, 17.06.2-ce, 17.09.1-ce, or 17.12.1-ce Recommended: 17.12.1-ce
TIBCO Spotfire®	10.10.0 or later.

Hardware

Hardware	Requirement
Processor	• Minimum: 4 cores (Intel Core i3 or equivalent), 2 GHz, 64-bit. • Recommended: 8 cores or more (Intel Xeon E5 or equivalent), 2 GHz or higher, 64-bit.  The system default is one engine fewer than the number of cores. You can add cores to accommodate additional engines.

Hardware	Requirement
Ram	• Minimum: 8 GB. • Recommended: 16 GB.  The minimum amount of RAM is suitable for only basic test systems, or systems with very few simultaneous users. Production servers with large analyses or many simultaneous users require significantly more RAM. Contact TIBCO for further assistance with this.
Hard disk space	• Minimum: 12 GB. • Recommended: 25 GB.

Operating Systems

Tested operating system	Version
Red Hat Enterprise Linux	• 8 (Intel/AMD x64) • 7 (Intel/AMD x64) • 6 (Intel/AMD x64)
SUSE Enterprise Linux	• 12 (Intel/AMD x64) • 11 (Intel/AMD x64)
Microsoft Windows Server	• 2016 • 2012 R2 • 2012

Limiting exposure of your deployment

Version 1.0.0 of the Spotfire Service for Python runs the Spotfire Service for Python under Linux or Windows. The Linux installation provides the option of running the Python engine in a container, on a Spotfire Server node running under Linux.

When you install the Spotfire Service for Python and run a Spotfire Service for 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.

The version 1.0.0 installed on a Spotfire Server node running under Windows does not have a containerized installation available.

Restricted user access

- Run the Spotfire Service for Python using an account that limits network access to only required external data sources and services. (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 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.

Tighter engine control

- The configuration for the Spotfire Service for Python uses containers. Running the Spotfire Service for Python with containers prevents the engines from having access to the host system. See Containerized Python service 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 TIBCO Software Inc. Download and use of Docker is solely at your own discretion and subject to license terms applicable to Docker.

Containerized Python service

Spotfire Service for Python on Linux is configured to use a Docker® container by default.

To use a Spotfire Service for Python running a containerized Python engine 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.

We tested using version 17.03.1-ce, 17.06.2-ce, 17.09.1-ce, or 17.12.1-ce (17.12.1-ce is recommended). See www.docker.com for more information.

Containerized Spotfire Service for Python operates the Python engine in a "sandbox", so the Python engine does not have access to the host file system. Because containerized Spotfire Service for Python prevents the Python engine from accessing the host system, it can execute Spotfire data functions if the data function is from a trusted source. (For more information about script and data function trust, see the *TIBCO Spotfire Analyst User's Guide*). Running the Spotfire Service for Python in a container results in negligible performance impact. If the data function is running in a container and attempts to access the file system or other host resource, the data function will fail.

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 out of the box. The service downloads and builds a default Docker image based on CentOS 7 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 Spotfire Service for Python. Docker is not supported, maintained, or warranted in any way by TIBCO Software 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 Spotfire Service for Python for more information. The property to change in the configuration is startup.hook.script.

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.

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 Spotfire Service for Python.

Prerequisites

- You must have Docker installed on the computer running the node manager. If you install and start Spotfire Service for Python 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 Spotfire Server can be on different computers).
- You must have access to the internet.

Custom docker images for the Python service must contain the following.

- The Java 11 Runtime.
- The JAVA_HOME environmental variable, correctly defined.

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

Procedure

1. If you have already installed Spotfire Service for Python from the Spotfire Server Nodes & Services 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`.

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

   ```
   ##########################################
   # A sample Dockerfile for a startup script. #
   ##########################################
   FROM centos:7.7.1908
   #  install openjdk
   RUN yum update -y && yum install -y java-11-openjdk && yum clean all
   #  set JAVA_HOME variable
   ENV JAVA_HOME=/usr/lib/jvm/jre
   ```

   For more information, see https://docs.docker.com/engine/reference/builder/.
3. 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/.
4. On the computer running Spotfire Server, export the `custom.properties` as described in steps 1-3 of Configuring Spotfire Service for Python.

5. Edit the settings in the file `custom.properties`, specifying the name and tag of your custom image.
```
use.engine.containers: TRUE
docker.image.name: <name:version>
```
6. On the computer running Spotfire Server, import the `custom.properties` as described in Configuring Spotfire Service for Python.
7. From the Spotfire Server Nodes & Services page, install the service, specifying the configuration to use, and then start the service.

   See Installing a Spotfire Service for Python instance on a node manager for a Spotfire Server for more information.

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

What to do next

If problems occur, troubleshoot by examining the `Dockerfile` that Spotfire Service for Python writes. After the service runs, this `Dockerfile` is available at the root service directory on the computer running the node manager. For example, `/opt/tibco/tsnm/<server version_#>/nm/services/Python 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

- You must have a computer with access to internet, and that has Docker installed on it.
- You must have a Linux computer where the node manager is installed. (Your node manager and Spotfire Server can be on different computers).
- You must have Docker installed on the computer running the node manager. If you install and start Spotfire Service for Python before you install Docker, then exceptions are written to the log.

Custom docker images for the Python service must contain the following.

- The Java 11 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`.

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

```
##########################################
# A sample Dockerfile for a startup script. #
##########################################
FROM centos:7.7.1908
#  install openjdk
RUN yum update -y && yum install -y java-11-openjdk && yum clean all
#  set JAVA_HOME variable
ENV JAVA_HOME=/usr/lib/jvm/jre
```

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

2. 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/.

3. 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/.

4. If you have already installed Spotfire Service for Python from the Spotfire Server Nodes & Services page, and if it is running, then stop the service.
5. Transfer the `.tar` file to the target computer (where the node manager is running).
6. 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: terrsrv: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/.

7. On the computer running Spotfire Server, export the `custom.properties` as described in steps 1-3 of Configuring Spotfire Service for Python.
8. Edit the settings in the file `custom.properties`, specifying the name and tag of your custom image.

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

9. On the computer running Spotfire Server, import the `custom.properties` as described in Configuring Spotfire Service for Python.
10. From the Spotfire Server Nodes & Services page, install the service, specifying the configuration to use, and then start the service.

    See Installing a Spotfire Service for Python instance on a node manager for a Spotfire Server for more information.

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

Configuring a custom startup script to build a custom Docker image

You can provide a startup script that is installed and configured on your Spotfire Server to build a custom Docker image for your Spotfire Service for Python.
Perform this task from the command line on the computer where your Spotfire Server is installed, and on the computer where your node manager is installed. For more information about the startup script, see Startup script.

Prerequisites

- You must have a Linux computer where the node manager is installed. (Your node manager and Spotfire Server can be 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 Python service must contain the following.

- The Java 11 Runtime.
- The JAVA_HOME environmental variable, correctly defined.

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

Procedure

1. If you have already installed Spotfire Service for Python from the Spotfire Server Nodes & Services page, and if it is running, then stop the service.
2. On the computer running Spotfire Server, export the `custom.properties` as described in steps 1-3 of Configuring Spotfire Service for Python.
3. On the computer running Spotfire Server, create a file called `Dockerfile`, and then save it to your custom configuration directory.

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

```
##########################################
# A sample Dockerfile for a startup script. #
##########################################
FROM centos:7.7.1908
#  install openjdk
RUN yum update -y && yum install -y java-11-openjdk && yum clean all
#  set JAVA_HOME variable
ENV JAVA_HOME=/usr/lib/jvm/jre
```

4. On the computer running Spotfire Server, create a custom script to build the `Dockerfile`, and then save it to your custom configuration directory.

   (By default, *<serverinstallation dir>*`/tomcat/spotfire-bin/config/root/conf`.)

   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
```

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

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

6. On the computer running Spotfire Server, import the `custom.properties` as described in Configuring Spotfire Service for Python.
7. From the Spotfire Server Nodes & Services page, install the service, specifying the configuration to use, and then start the service.

   See Installing a Spotfire Service for Python instance on a node manager for a Spotfire Server for more information.

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

What to do next

If problems occur, troubleshoot by examining the `Dockerfile` that Spotfire Service for Python writes. After the service runs, this `Dockerfile` is available at the root service directory on the computer running the node manager. For example, `/opt/tibco/tsnm/<server version_#>/nm/services/Python service Linux-<version_#_ID>/dockerfile/Dockerfile`

Pulling a custom docker image from an authenticated repository

You can create a custom start script to configure 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 can be on different computers).

Custom docker images for the Python service must contain the following.

- The Java 11 Runtime.
- The JAVA_HOME environmental variable, correctly defined.

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

Procedure

1. If you have already installed Spotfire Service for Python from the Spotfire Server Nodes & Services 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 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 > Respositories**.
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/pysrv/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/pysrv/pysrv-
sample:latest
```

4. On the computer running Spotfire Server, export the `custom.properties` as described in steps 1-3 of Configuring Spotfire Service for Python.
5. On the computer running 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`

Remember that for any script you write, the line endings must be appropriate for the operating system where Spotfire Service for Python 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.
```

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

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

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

7. From the command line, manually test the script to make sure that it works correctly.
8. On the computer running Spotfire Server, import the `custom.properties` as described in Configuring Spotfire Service for Python.
9. From the Spotfire Server Nodes & Services page, install the service, specifying the configuration to use, and then start the service.
See Installing a Spotfire Service for Python instance on a node manager for a Spotfire Server for more information.

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

What to do next

If problems occur, troubleshoot by examining the `Dockerfile` that Spotfire Service for Python writes. After the service runs, this `Dockerfile` is available at the root service directory on the computer

running the node manager. For example, `/opt/tibco/tsnm/<server version_#>/nm/services/Python service Linux-<version_#_ID>/dockerfile/Dockerfile`

Installing a Spotfire Service for Python instance on a node manager for a Spotfire Server

After installing and authorizing a node manager, install Spotfire Service for Python. Any computer on the network can access Spotfire Service for Python.

Prerequisites

- You have installed and authorized a node manager. See the topics "Installing a node manager" and "Trusting a node" in the TIBCO® Spotfire Server Installation and Administration Help.

  Do not install Spotfire Service for Python for different operating systems for one Spotfire Server.
- Your deployment area contains the Spotfire package (SPK) for Spotfire Service for Python.
- Spotfire Server and the node manager are up and running.
- Optional: You have created and imported a custom configuration for Spotfire Service for Python.

Procedure

1. Log in to Spotfire Server, select your deployment area, and click **Nodes & Services**.
2. In the **Nodes** view, select the node to which you want to add Spotfire Service for Python.

   A running service displays a green circle with a check mark next to the selected node manager.

   In the **Installed services** area, the name of the node manager is displayed in the lower-right pane of the window.
3. Click **Install new service**.
4. Make your selections in the Install new service dialog box:
   a) Under **Deployment area**, select the area where you deployed 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 *TIBCO Spotfire® Server Installation and Administration Help* for more information on creating a custom configuration.
   d) Under **Number of instances**, leave the option set to `1`.

      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.
   e) Under **Port**, you can change the default as needed.
   f) Under **Name**, provide a name for this service.
5. Click **Install and start**.

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

   The service is installed and started.

What to do next

For information on the remaining setup tasks, see the topic "Post-installation steps" in the Spotfire Server help.

Configuring Spotfire Service for Python

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 Spotfire Server where you have installed the Spotfire Service for Python.

For general information about configuring services for Spotfire Server, see https://docs.tibco.com/products/tibco-spotfire-server.

Procedure

1. Open a command line as administrator and change the directory to the location of the command-line config tool (`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 for more information.)
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 *TIBCO Spotfire® Server and Environment Installation and Administration* 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 components, select **Nodes**, and then select the Python service.
12. Click **Edit**.
    The Edit service dialog box 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 Spotfire Service for Python logs.

To change the new configuration, you 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
`engine.session.max`	*<one less than the number of logical processors available on the node>*	The 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 engine session.
`engine.queue.size`	*<one quarter of the number of logical processors on the host, constrained to a minimum of 1 and a maximum of 10.>*	The number of Python engines preallocated and available for new sessions in the Spotfire Service for Python engine queue. The service always starts with enough engines to keep the queue at the requested level. The total number of engines that can run at the same time is the sum of `engine.session.max` + `engine.queue.size`. This number can be set manually to a value higher than 10.

For more information on how engine resources can be managed, see Service resource management scenarios.

Engine pruning

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

Configuration property	Default setting	Description
`engine.prune`	`10`	The time, in seconds, that a Spotfire Service for Python engine can be idle before the Spotfire Service for Python prunes it.
`dynamic.prune.threshold`	`60`	The Spotfire Service for Python capacity at which idle pruning is engaged, as a percentage value. By default, when the Spotfire Service for Python reaches 60% capacity of usage, then it begins the idle-pruning process as specified by `engine.prune`. • Set to `0` to always prune when a Spotfire Service for Python engine is idle. • Set to `100` to never prune when a Spotfire Service for Python engine is idle.

For more information, see Service resource management scenarios.

Engine timeout

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

Configuration property	Default setting	Description
`engine.execution.timeout`	`600`	The length of time, in seconds, that the Spotfire Service for Python allows a Spotfire Service for Python engine to execute a request before stopping the execution with a timeout error.
`engine.session.maxtime`	`1800`	The length of time, in seconds, that the Spotfire Service for Python allows a Spotfire Service for Python engine session to exist before killing it. Set to -1 to disable session pruning.

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
`spring.servlet.multipart.max-file-size`	`100 MB`	The total file size for upload cannot exceed the value for this setting.
`spring.servlet.multipart.max-request-size`	`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 INFO (provides 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 `DEBUG` 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.

Package library location

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

Configuration property	Default setting	Description
`packagePath`	none	The absolute path to the shared package library location. When specifying the path, you must use a forward slash regardless of operating system. • If you install packages into your TIBCO Spotfire® deployment using the SPK process, then this setting is not required. • If you install packages onto the server manually using pip, then set this path to the package installation location. See Installing Python packages manually for more information.  Avoid installing packages that are already 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.

Example

```
packagePath: /opt/python/library
```

For more information, see Package management for Spotfire Service for Python.

Safeguarding your environment

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

Configuration property	Default setting	Description
`disable.spotfire.trust.checks`	`false`	By default, Spotfire Service for Python checks whether a data function has come from a trusted source. Set to `true` to not check for the data function trust status of any data function run on Spotfire Service for Python. ⚠ Setting this value to `true` 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. For more information about script and data function trust, see the *TIBCO Spotfire® Analyst User's Guide* and *TIBCO Spotfire® Administration Manager User's Guide*.

Startup script

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

Configuration property	Default setting	Description
`startup.hook.script`	none	The path and name of a startup hook script that runs before any container or Spotfire Service for Python engine is started. This value can be empty (for no script) or a relative path from the Spotfire Service for Python working directory. Place the startup script in the directory with the `custom.properties` file (by default *<server-installation-dir>*`/tomcat/spotfire-bin/config/root/conf/`). The Spotfire Service for Python runs a `.sh` file format. The script must have appropriate permissions before the Spotfire Service for Python executes it. 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 your deployment area, or you can run another script. See Containerized Python service for more information.)

Example

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

Python engine ports

Spotfire Service for Python engines running under the Spotfire Service for Python require open ports to communicate. The first available port, and range to the last available port are determined by these two settings. (The defaults specify a range of 62000 to 63000.)

Configuration property	Default setting	Description
`engine.port.min`	`62000`	The first specified available port set for a Spotfire Service for Python engine.
`engine.port.range`	`1000`	This value, added to the value specified in `engine.port.min`, indicates the range of the ports available for the Spotfire Service for Python engines.

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.

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
`jmx.rmi.username`	None	Set this value to the JMX user name.
`jmx.rmi.password`	None	Set this value to the JMX password.
`jmx.rmi.host`	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.)
`jmx.rmi.port`	None	Set this value to an available port for the RMI connection with JMX.
`jmx.active`	`TRUE`	Set this value to `TRUE` to activate JMX.

For more information, see Monitoring Spotfire Service for Python using JMX.

Containerized configuration

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

Configuration property	Default setting	Description
`use.engine.containers`	`TRUE`	Runs the Python engine inside a container when set this value to `TRUE` (the default).
`ram.limit`	`1000`	The amount of RAM and SWAP memory to which the Python engine containers are constrained, in megabytes.
`docker.image.name`	`centos:7.7.1908`	When you use containers, the Spotfire Service for Python builds a custom image based on a starting image. This property is used by the Dockerfile as the `FROM` line. See https://docs.docker.com/engine/reference/builder/#from for more information. The Spotfire Service for Python default is CentOS.

For more information, see Containerized Python service.

Spotfire and Python data type mapping

To create Python data functions in Spotfire, you need to know how the data types in each application map to each other. This table provides that mapping, including data type mapping to Pandas column dtype, and for mapping columns and tables.

Spotfire	Python	Pandas column dtype
Integer	int	Int32
LongInteger	int	Int64
Real	float	float64
SingleReal	float	float32
Currency	decimal.Decimal	object
Date	datetime.date	object
DateTime	datetime.datetime	object
Time	datetime.time	object
TimeSpan	datetime.timedelta	object
Boolean	bool	object
String	str	object
Binary	bytes	object

- Spotfire columns map in Python to Pandas `Series` type.
- Spotfire tables map in Python to Pandas `DataFrame` type.

Package management for 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.

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.

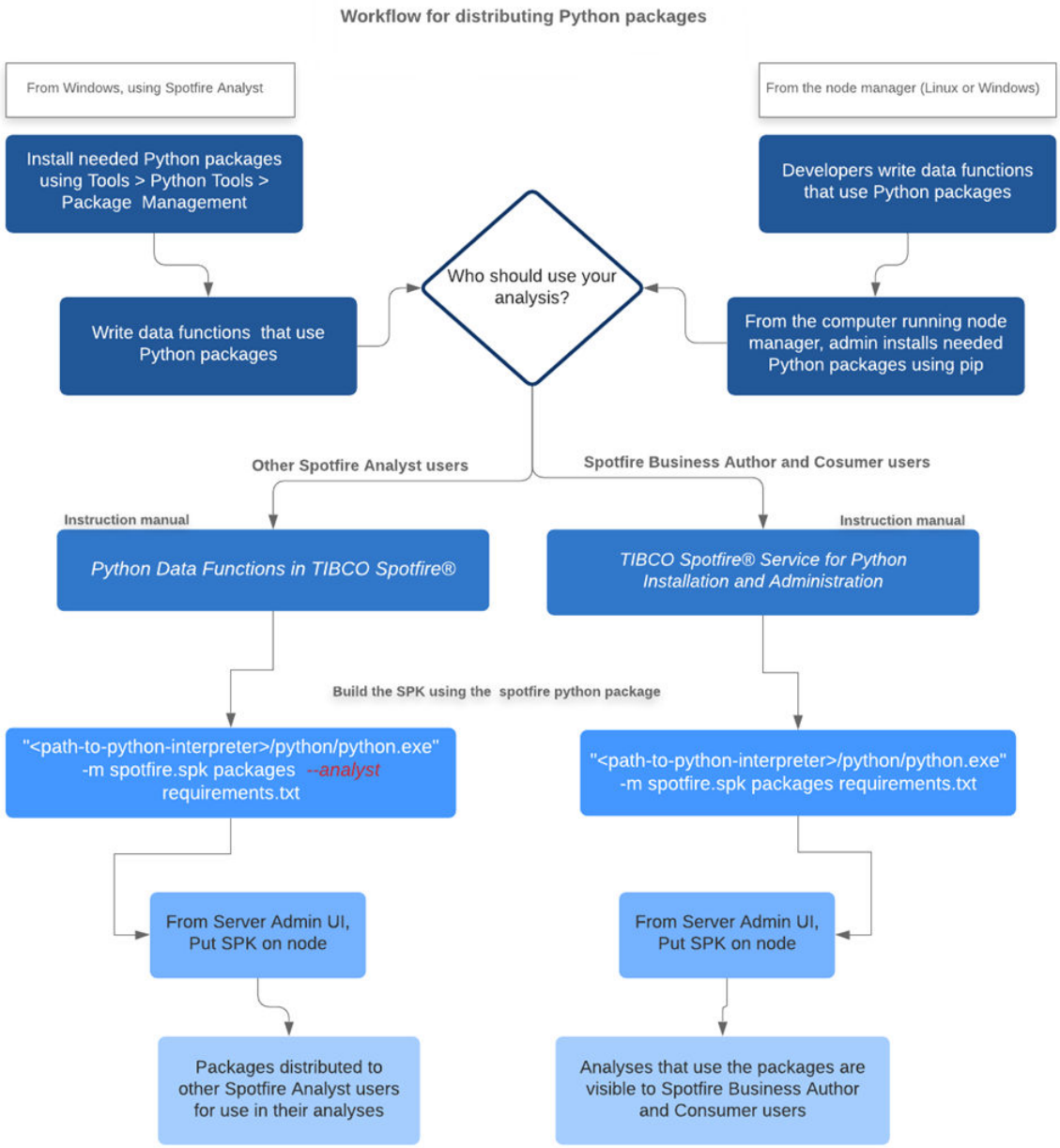
Spotfire analysts can develop data functions that use these packages for either other Spotfire Analyst installations or by users of Spotfire Business Author and Consumer.

- To share an SPK with Spotfire Analyst installations, see the instructions in the guide *Python Data Functions in TIBCO Spotfire®*. Packages deployed in this manner are distributed to other Spotfire Analyst installations in your organization.
- To share an SPK with Spotfire Business Author and Consumer users through Spotfire Web Player, see SPKs for Python packages. Packages deployed in this manner are on Spotfire Server.

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

- If your node manager is running on a Windows server, then you can create the SPK either from a Windows computer running Spotfire Analyst, or you can create the SPK from the computer running the node manager where Spotfire Service for Python is installed.
- If your node manager is running on a Linux server, then you must create the SPK from the computer running the node manager where Spotfire Service for Python is installed.

The following image shows the workflow for creating an SPK to share Python packages with other analysts, or with Spotfire Business Author and Consumer users.



Included Packages

Your installation of Spotfire includes a Python interpreter and several packages it needs to run under Spotfire. To run the Python interpreter, you must have the 64-bit version of Spotfire.

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

Avoid installing packages that are already included in the Spotfire Service for Python. Installing a different version of an included package can cause unexpected results.

Package name	Version	Description	More information
bitstring	3.1.6	A pure Python module that makes the creation, manipulation and analysis of binary data as simple and natural as possible.	https://pypi.org/project/bitstring/ https://pythonhosted.org/bitstring/
numpy	1.17.4	Provides the following. • An N-dimensional array object. • Broadcasting functions. • Tools for integrating C/C++ and Fortran code. • Linear algebra, Fourier transform, and random number capabilities.	https://pypi.org/project/numpy/ https://docs.scipy.org/doc/numpy/
pandas	0.25.3	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/ https://pandas.pydata.org/pandas-docs/stable/
pip	19.3.1	Provides support for installing packages.	https://pip.readthedocs.io/en/stable/reference/pip_install/#description
python-dateutil	2.8.1	Provides extensions to the datetime module in Python.	https://pypi.org/project/python-dateutil/ https://dateutil.readthedocs.io/en/stable/
pytz	2019.3	Provides a platform for cross-platform time zone calculations.	https://pypi.org/project/pytz/
six	1.13.0	Provides utility functions for smoothing over the differences between the Python versions 2 and 3.	https://pypi.org/project/six/ https://six.readthedocs.io/
setuptools	42.0.2	Provides tools for building, installing, upgrading, and uninstalling Python packages.	https://pypi.org/project/setuptools/ https://setuptools.readthedocs.io/en/latest/

Package name	Version	Description	More information
wheel	0.33.6	The reference implementation of the Python wheel packaging standard, as defined in PEP 427.	https://pypi.org/project/wheel/ https://wheel.readthedocs.io/en/stable/

Manage packages through roles

Working with packages in a deployment that includes Spotfire and Spotfire Server can add some complexity to management policies.

The job of synchronizing package versions among your development computers, your testing computers, and your servers is an important package management concern for an organization. You can reduce the risk of confusion and streamline your processes by defining roles in your organization for dealing with packages. Ensure processes and rules are established to manage packages.

Administrator role

The Spotfire administrator manages packages on the Spotfire Server.

The responsibilities for the administrator role include the following.

- Deploys the SPK containing Python packages that are distributed to Spotfire Analyst users.
- Assigns licenses for access to the Data Functions feature in Spotfire Analyst.
- Uploads, maintains, and removes packages. (Might assign server permissions to the curator for this task.)

Developer role

The developer is a Python programmer or statistician who develops packages or writes data functions using Python.

The package developer accomplishes the following tasks using the Spotfire tools.

- Develops and tests Python packages or data functions using the local Python interpreter available from Spotfire Analyst.
- Reviews and recommends packages to be included on the Spotfire Server for data functions to use.

Curator role

The curator maintains the standards and lists of officially-sanctioned packages. The curator keeps all of the package versions synchronized. The curator might be the same person who fills the developer role.

The approval process for adding a package is up to your organization, and might vary from minimal to extensive, depending on your usual practices. Designate a developer familiar with Python packages and package versioning to be the package curator. The package curator works with package developers and server administrators to perform the following management tasks.

- Maintains the list of tested and sanctioned package versions (the gold standard), which would be the set of packages available for general use under Spotfire applications.
- Creates a Spotfire SPK containing the Python packages, and then gives it to the administrator who manages Spotfire distributions on Spotfire Server. Packages uploaded to Spotfire Server are distributed to other Python users who write data functions using Python in Spotfire Analyst.
- Ensures that the SPK containing the "gold standard" package versions are placed on the Spotfire Server for distribution to Spotfire Analyst clients, or are deployed to a Spotfire Server for use by analyses available to Spotfire Business Author and Consumer users.

- 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.

See SPK Versioning for more information.

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 Package library location 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.

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 `tibco/tsnm/nm/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 Spotfire Service for Python 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 Spotfire Service for Python.

The Spotfire 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 Spotfire Analyst users, or distributed to the Spotfire Server node for use by another service.

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

- The Python package (usually downloaded from a repository, such as PyPI) contains Python modules.
- The SPK is a means to deploy extensions to the Spotfire Server, which either distributes its contents to Spotfire Analyst users, or installs a service, such as the Spotfire Service for Python, to use from the Spotfire Server node.

This Spotfire installation provides a specialized Python package that creates an SPK to hold packages.

SPKs for Python packages

You can distribute Python packages by using the Spotfire SPK mechanism for either Spotfire Server installation running on Windows OS, or Spotfire Server installation running on a Linux OS.

An SPK containing packages must be created using the operating system that the node manager runs under.

- If your node manager is running on a Windows server, then you can create the SPK containing Python packages from a Windows computer running Spotfire Analyst, or you can create the SPK from the computer running the node manager where Spotfire Service for Python is installed.
- If your node manager is running on a Linux server, then you must create the SPK containing Python packages from the Linux computer that is running the node manager where Spotfire Service for Python is installed.

This section provides instruction for creating an SPK for both of those cases.

Creating an SPK for Python Packages from Spotfire Analyst on Windows

An installation of Spotfire Analyst includes a Python interpreter and a set of packages to enable using Python in Spotfire.

One of these packages, `spotfire.zip`, provides tools for building SPKs to share Python packages with other data function authors in an organization, or to enable Spotfire Business Author and Consumer users to see Python-enabled analyses in a web browser. For more information about package workflow, see Package management for Spotfire Service for Python.

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.zip` package supplied with your Windows installation of Spotfire Analyst. If you are installing the SPK on a Linux node, see Creating an SPK for Python Packages on the node manager.

Spotfire Analyst relies on pip, the Python command-line application for Python package installation. Spotfire Analyst 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
```

Avoid installing packages that are already 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.

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

Python users perform this task from a command prompt on the Windows computer where Spotfire Analyst is installed.

An installation of Spotfire Analyst relies on the Python packages included in the installation. Removing any of these packages causes your Spotfire Analyst installation to not work with the included Python interpreter. See Included Packages for more information.

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 == 1.3.3
matplotlib == 3.1.2
statsmodels == 0.10.2
```

Procedure

1. At the command prompt, set the Python path (`PYTHONPATH`) to the directory where the Python package `spotfire.zip` is installed.

```
set PYTHONPATH=%SPOTFIRE_HOME%\Modules\Core_<core-build-version>\python
\spotfire.zip
```

   For example purposes, we refer to the directory where Spotfire Analyst is installed as *%SPOTFIRE_HOME%*.

   The `PYTHON_PATH` environment variable is set to the directory where the `spotfire.zip` package is installed.

2. From the command line, type the following command.

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

Option	Description
`spotfire.spk`	The package containing the Python code to download and bundle the needed packages specified in `requirements.txt`.  From the command prompt, you can view help for the package `spotfire.zip` by typing the following command: `"<path-to-python-interpreter>\python\python.exe" -m spotfire.spk packages --help`
`packages`	The subcommand that creates the package bundle.
`--name`	A string that sets the internal name of the generated package (for deploying multiple SPK files).  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, or, if it is missing entirely, it reverts to `"Python Packages Windows"` for server packages.
`--analyst`	This argument specifies that the Spotfire Server should distribute the packages in the SPK to other Spotfire Analyst clients connected to the Spotfire Server. (If you do not use this option, then the packages are placed on the server for Spotfire Business Author and Consumer users who connect using the Spotfire Web Player.)
`name.spk`	The name of the SPK file that is created by this task.
`requirements.txt`	The full path to the file `requirements.txt`.

Also call the command, passing in the same packages and using the `--analyst` flag, to create an SPK to distribute the same packages with other Spotfire Analyst users in your organization. Everyone should use the same packages and package versions.

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

```
"%SPOTFIRE_HOME%\Modules\Python Interpreter_3.7.5.0\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 == 1.3.3
matplotlib == 3.1.2
statsmodels == 0.10.2

## spotfire.spk: {"BuiltBy":"3.7.5 (tags/v3.7.5:e09359112e, Nov  30 2019,
## spotfire.spk:  20:34:20) [MSC v.1916 64 bit (AMD64)]","BuiltAt":"Thu
## spotfire.spk: Dec 12 15:41:32 2019","BuiltFile":"my-data.spk","BuiltN
## spotfire.spk: ame":"Python Packages","BuiltId":"b8b9e4ec-324d-4da1-bd
## spotfire.spk: 4b-6a3509c7877d","BuiltVersion":"1.0.0.0","BuiltPackage
## spotfire.spk: s":{"cycler":"0.10.0","kiwisolver":"1.1.0","matplotlib"
## spotfire.spk: :"3.1.2","numpy":"1.17.4","pandas":"0.25.3","patsy":"0.
## spotfire.spk: 5.1","pyparsing":"2.4.5","python-dateutil":"2.8.1","pyt
## spotfire.spk: z":"2019.3","scipy":"1.3.3","setuptools":"42.0.2","six"
## spotfire.spk: :"1.13.0","statsmodels":"0.10.2"}}
```

3. Locate the SPK you created in the working directory where you ran the command.
4. Deploy the SPK to the Spotfire Server.

Result

After the SPK is placed into the deployment area, the packages are deployed to the Spotfire Server. Business Author and Consumer users can see analyses that use the functions in the packages in a web browser. For an overview of the entire process, see Package management for Spotfire Service for Python.

Installing the 'spotfire' package on a Linux computer

Before you create an SPK to deploy on a Linux computer where Spotfire Service for Python is installed, you must first install the 'spotfire' package that is included in the download bundle `.whl` file, along with Spotfire Service for Python.

This step is not required for Windows installations because your Spotfire Service for Python installation already includes the package that builds the SPK.

Prerequisites

- You must have write access to the Linux computer running the node manager where Spotfire Service for Python is installed.
- You must have Python 3.7 installed on the system. The instructions below assume "python" will point to the system-installed Python 3.7.

Procedure

1. Locate the `.whl` file included in the Spotfire Service for Python distribution bundle.

   This file not in the SDN. You can find it in the zipped archive at `python/spotfire-<version#>-py2.py3-none-any.whl`
2. Copy the `.whl` file to the Linux computer running the node manager where Spotfire Service for Python is installed.
3. 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 `tibco/tsnm/nm/services/<Spotfire Service for Python name>/python`.

4. Run the following command.

```
python -m pip install --target=<path-to-.whl> spotfire-<version#>-py2.py3-none-any.whl
```

Where <*path-to-.whl*> is the location where you placed the `.whl` file, and <*version#*> is the version number that is part of the file name.

The 'spotfire' package from the `.whl` file is installed into the Python interpreter.

What to do next

You can now use the 'spotfire' package to create SPK files containing Python packages from the node manager computer.

Creating an SPK 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 Spotfire Business Author and Consumer users to see Python-enabled analyses in a web browser.

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 supplied with download bundle. (You can find it in the zipped archive at `python/spotfire-<version#>-py2.py3-none-any.whl`.)

- 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 Business Author and Consumer users.
- If you are building an SPK for Windows only, then the SPK can be deployed to the node manager, to other Spotfire Analyst users, or to both. Packages deployed to the node manager are used by data functions in analyses accessed through a web browser by Business Author and Consumer users. (Alternatively, you can build the SPK from an installation of Spotfire Analyst. For those details, see Creating an SPK for Python Packages from Spotfire Analyst on Windows.

The package builder relies on pip, the Python command-line application for Python package installation. 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
```

Avoid installing packages that are already 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.

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

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

Prerequisites

- You must have write access to the computer running the node manager where 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 specified versions from PyPI.

  ```
  #
  ####### example-requirements.txt #######
  #
  scipy == 1.3.3
  matplotlib == 3.1.2
  statsmodels == 0.10.2
  ```
- Linux servers only: You must first install the 'spotfire' package into the Python interpreter.

Procedure

1. From the command line, type the following command.

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

   where *"%Python_Service_Home%* is the Python Service installation location. By default, this is usually `/opt/tibco/tsnm/<server-version#>/nm/services/Python Service Linux-<installed-service-guid>`

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

Option	Description
`spotfire.spk`	The package containing the Python code to download and bundle the needed packages specified in `requirements.txt`.  From the command prompt, you can view help for the package `spotfire.whl` by typing the following command: `"%Python_Service_Home%/python/python" -m spotfire.spk packages --help`
`packages`	The subcommand that creates the package bundle.

Option	Description
`--name`	A string that sets the internal name of the generated package (for deploying multiple SPK files). 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, or, if it is missing entirely, it reverts to `"Python Packages Linux"` for server packages.
--analyst	This option specifies that the Spotfire Server should distribute the packages in the SPK to other Spotfire Analyst clients connected to the Spotfire Server. (If you do not use this option, then the packages are placed on the server for Spotfire Business Author and Consumer users who connect using the Spotfire Web Player.)
`name.spk`	The name of the SPK file that is created by this task.
`requirements.txt`	The full path to the file `requirements.txt`.

Also call the command, passing in the same packages and using the `--analyst` flag, to create an SPK to distribute the same packages with other Spotfire Analyst users in your organization. Everyone should use the same packages and package versions.

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
"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 `requirements.txt`. For example:

```
#
####### example-requirements.txt #######
#
scipy == 1.3.3
matplotlib == 3.1.2
statsmodels == 0.10.2

## spotfire.spk: {"BuiltBy":"3.7.5 (tags/v3.7.5:e09359112e, Nov  30 2019,
## spotfire.spk:  20:34:20) [MSC v.1916 64 bit (AMD64)]","BuiltAt":"Thu
## spotfire.spk: Dec 12 15:41:32 2019","BuiltFile":"my-data.spk","BuiltN
## spotfire.spk: ame":"Python Packages","BuiltId":"b8b9e4ec-324d-4da1-bd
## spotfire.spk: 4b-6a3509c7877d","BuiltVersion":"1.0.0.0","BuiltPackage
## spotfire.spk: s":{"cycler":"0.10.0","kiwisolver":"1.1.0","matplotlib"
## spotfire.spk: :"3.1.2","numpy":"1.17.4","pandas":"0.25.3","patsy":"0.
## spotfire.spk: 5.1","pyparsing":"2.4.5","python-dateutil":"2.8.1","pyt
## spotfire.spk: z":"2019.3","scipy":"1.3.3","setuptools":"42.0.2","six"
## spotfire.spk: :"1.13.0","statsmodels":"0.10.2"}}
```

2. Locate the SPK you created in the working directory where you ran the command.
3. Deploy the SPK to the Spotfire Server.

Result

After the SPK is placed into the deployment area, the packages are deployed to the Spotfire Server; Business Author and Consumer users can see analyses that use the functions in the packages in a web browser.

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 an SPK for Python Packages from Spotfire Analyst on Windows. 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.

Versioning rules

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.
Adding package names to an existing `requirements.txt`.	Edit the `requirements.txt`, 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 `requirements.txt`. Spotfire Server registers the SPK as changed and distributes it to the users.

Task	Procedure	Version result	Version example	Comment
Removing package names from an existing `requirements.txt`.	Edit the `requirements.txt`, 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 `requirements.txt`. Spotfire Server registers the SPK as changed and distributes it to the users.
Assigning a specific version number to a `requirements.txt`.	Run the Python script and pass in the `requirements.txt` in the command, along with the `version`, 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, `"--version = 1.2.3.4"` or `"-v 1.2.3.4"`).

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 Spotfire Service for 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 Spotfire Service for 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 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 `engine.execution.timeout`, 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 `dynamic.prune.threshold` value of `100`. The user can access this same engine for up to `engine.session.maxtime` (in this case `120` seconds).
3A: Four users submit jobs that execute for 5 seconds each	Because `dynamic.prune.threshold` is set to `100` (specifying never idle prune), the first three user jobs claim the engines (`engine.session.max`) for the entire duration of `engine.session.maxtime`, or `120` seconds. The fourth user must wait until that time expires to claim a new engine, which are then 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 `dynamic.prune.threshold` of `0`. • If the user does not submit a new job within 10 seconds defined by the `engine.prune` idle timeout, then the used engine is destroyed. • If the user submits a new job before the timeout occurs, then the second job uses the same engine. • If the user submits a new job after the timeout occurs, then another engine is pulled 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 `engine.session.max` has been reached. However, `dynamic.prune.threshold` is set to always idle prune (`0`), so after submitting a job and getting results, if a user sits idle for longer than `engine.prune` 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 for more information about these custom properties. See Configuring Spotfire Service for Python for information about setting all custom properties.

Spotfire Service for Python 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
`service-<guid>-stdout.log`	Prints information about the service startup, shut down, and any exceptions.
`python-service-<guid>.log`	Prints configuration options that the Spotfire Service for Python starts with. Provides granular level of the individual engines, job execution details.

Monitoring Spotfire Service for Python 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. (See Using JConsole from the Oracle Java documentation site.) Alternatively, you can install and use VisualVM to monitor the Spotfire Service for Python using JMX.

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 Python service>
   jmx.rmi.port: 1099
   jmx.active: TRUE
   ```

   See Configuring Spotfire Service for Python 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.

   ```
   2018-06-08T21:03:11,520 | 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 displayed indicating `the connection could not be made using SSL. Would you like to try without SSL?`, then click **Insecure connection**.

   JConsole should now display information from the service.
7. To view metrics specific to the Spotfire Service for Python, click the tab **MBeans**.

8. In the left panel, expend the group labeled **metrics**.
   Metrics are listed, including many JVM metrics. Some of the metrics specific for Spotfire Service 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 Spotfire Service for Python

If you have problems with the Spotfire Service for Python, review these tips.

Problems with the `Dockerfile` on a Linux node manager

After the service runs, you can view the `Dockerfile` that the Spotfire Service for Python writes. You can find the file in the root service directory (for example, `/opt/tibco/tsnm/<server version_#>/nm/services/Python service Linux-<version_#_ID>/dockerfile/Dockerfile`.

Try test building the Docker image in the environment before starting the Spotfire Service for Python.

Building an image takes time, so it can take a few minutes for the web UI to display a possible image build failure. If a build failure occurs, the retry mechanism is triggered automatically.

Problems with the startup script

Check your script line endings.

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

Release Notes

Visualizations that are created in TIBCO Spotfire® Analyst that take advantage of Spotfire Service for Python scripts are available for users on TIBCO Spotfire® Web Player through TIBCO Spotfire® Service for Python version 1.0.0.

This is the first version of the Spotfire Service for Python.

New Features

This is the first version of the TIBCO Spotfire® Service for Python.

Changes in Functionality

There are no changes in functionality in version 1.0.0 of the Spotfire Service for Python.

Migration and Compatibility

There are no migration or compatibility issues in version 1.0.0 of the Spotfire Service for Python.

Deprecated and Removed Features

There are no deprecated or removed features in version 1.0.0 of the Spotfire Service for Python.

Closed Issues

There are no closed issues in version 1.0.0 of the Spotfire Service for Python.

Known Issues

This topic lists the known issues in version 1.0.0 of the Spotfire Service for Python.

Plot created with pyplot persists between data functions

A plot created in a data function using the function pyplot from matplotlib can persist between data functions unless the plot is explicitly cleared.

Workaround: If the buffer retains the plot, you can clear it explicitly by including the command `plt.clf()` in your data function.

Index