Migrating from IzODA Anaconda to the Open Enterprise SDK and Python AI Toolkit on IBM z/OS

Introduction

The Anaconda component of the IzODA offering is a collection of 3 major components that provide a foundation for Python-based AI and machine learning processing:

- A Python interpreter to run code from Python packages.
- The Anaconda package manager.
- A runtime library of Python packages that implement AI and Machine Learning functions.

IzODA Anaconda has been split into the IBM Open Enterprise SDK for Python (the SDK), plus the Python AI Toolkit for IBM z/OS (the Toolkit). This separation allows users who wish to use the Python interpreter for purposes other than AI or machine learning (e.g. for DevSecOPs purposes) to avoid the overhead of installing AI and ML code that they won't use. It also allows for more agile handling of service issues between the Python language (SDK), and the runtime library of Python packages (Toolkit). Each product offering can progress independently of the other, and no longer have to be kept in lockstep.

Like many other open source languages, Python has an integrated package manager (pip), which is part of the SDK. It doesn't need to be installed separately. While the Anaconda package manager has functional advantages over pip, the simplicity of the pip environment allows us to eliminate Anaconda and the dependencies associated with it – like bash, and curl. Pip can run from the default Unix System Services (USS) shell without dependencies on any other shell-based utilities.

The Toolkit fundamentally changes the code deployment model from the conventional z/OS SMP/E install to use the package manager integrated with the Python interpreter. It assembles and handles sets of packages in the same way as the open source community, while retaining key z/OS qualities of service. There are several advantages to this approach:

- The Toolkit provides a truly continuous delivery model. Updates are posted to a secure server for download as soon as they are available. This greatly enhances security and version currency.
- All packages are built as Python wheels. There are no source distributions that require a compiler environment during package installation, and there is no opportunity for bad actors to build malware into a package during the install process.
- System administrators can install a common, secure copy of the Toolkit locally for users to reference in their installs.
- Users can easily install the whole Toolkit, or individual parts. This substantially reduces the Toolkit's installed footprint in the zFS filesystem.
- There are no post-install tools to run to set file ownership, permissions, or encodings. Everything needed to install a Python package for the Toolkit is handled by pip, with no other actions required by the administrator. This greatly reduces the number of places where an environment variable may be set incorrectly, or where a file permission is not set properly.
- The set of packages provided at the Toolkit's secure server are actively managed to remove vulnerable versions with critical and high-severity issues. Prior package versions that remain secure will continue to be hosted to preserve key dependencies, but dangerous packages will be removed to prevent accidental download. Package version currency will be much closer to the rest of the open source community, with less exposure to problems due to down level code.

Please note that the package list available through the toolkit is significantly different from IzODA. There are several reasons for this:

- Separation of the SDK from the Toolkit, as mentioned above. This means that none of the parts for the Python interpreter, or pip package manager are part of the toolkit.
- The packages of the Toolkit are much more current than the versions available through IzODA. The upstream open source communities have in some cases evolved their projects to use different dependencies.
- Some packages provided in IzODA were necessary to support the Anaconda environment. These are no longer required.

- IzODA Anaconda provided all versions of every package. Old, vulnerable packages were never deprecated or removed. This caused IzODA Anaconda to grow in an open-ended fashion.
- There is no R language support.

As a result, there are approximately 175 Python packages that make up the Python AI Toolkit for IBM z/OS, in contrast to IzODA Anaconda, which currently contains about 300 packages.

As with the IzODA offering, this Toolkit runs under Unix System Services. The recommended means to access the USS environment is through the ssh command line, or a client like PuTTY that supports the ssh protocol. It is not recommended that you install the Toolkit through an interface like the OMVS command, because the installation process can generate several pages of output from different commands.

The examples below also utilize the bash shell because it is more convenient. However it is not required, and the download and installation processes will function properly with the default shell (`/bin/sh`).

Remove IzODA Anaconda

IzODA Anaconda is installed as a licensed program product alongside other products on your system. This includes a large set of software located in several directories under `/usr/lpp/IBM/izoda/anaconda`. This forms a local channel that serves as a reference location for software. The `conda` command line (sometimes referred to as miniconda) references this channel to create and manage subsets of channel packages in user-defined environments. Once a user creates an Anaconda environment, they can activate it to provide a tailored runtime for any software than they intend to run. Conda requires the bash shell to run properly, and for this reason, bash ships with IzODA Anaconda.

There are a few primary sets of activities required to remove IzODA Anaconda from your system in preparation for the Python SDK and AI Toolkit:

- Ensure that any dependencies on tools or utilities that ship with IzODA Anaconda, but which are not strictly AI functions remain available. See the section below for your options to preserve these tools.
- Remove all user dependencies and references to IzODA Anaconda.
- Un-install the IzODA Anaconda product.

Assuming you have IzODA Anaconda installed on your system according to the documented instructions, here are the steps to remove these from your system.

Preserve Key Tools and Utilities

There are several tools included in IzODA Anaconda that provided functions not available as part of USS. Open source-based packages often assume the presence of these tools from the Linux environment in order to operate properly. z/OS is not alone in needing to supplement the default functional environment in order to support popular open source packages, like those found in the Toolkit.

Key among these is the `bash` shell, although other tools like `curl` and `make` may be user favorites that you should think about retaining. There are 3 options you have for preserving these key functions:

- Install z/OS Open Tools. This contains a useful superset of tools and utilities that you will likely need from IzODA Anaconda, and in most cases, they are at the latest version of a given package. However, this is provided by an open source community focused on providing Linux-based functions natively on z/OS. There is no formal support agreement available for it.
- Install Rocket Software's Open AppDev for Z offering. The package versions here are not generally as current as those provided by z/OS Open Tools, but Rocket does provide a service contract for a fee.

- Save the binaries from the IzODA installation to another shared location, like `/usr/bin`. This is a generally bad option, since it preserves the oldest versions of these key tools and utilities. With the end-of-service announced for IzODA at the end of February 2024, this should be seen as a temporary solution.

Perhaps the worst of all options is to do nothing, and simply remove the IzODA Anaconda installation. This can have unpredictable results on your developers who may have built dependencies into their workflows and pipelines. Replacing these key functions with z/OS Open Tools or Open AppDev for Z makes the most sense. If you want to bridge to one of these solutions by preserving parts of your current IzODA Anaconda environment, consider copying the following set of binaries from `/usr/lpp/IBM/izoda/anaconda/bin` to another common location like `/usr/local/bin` that users will likely have in their PATH:

```
bash                bzip2               curl                gettext
gzip                make                openssl             perl
php                 sed                 sqlite3             xz
```

Please note that this is the set of packages from IzODA Anaconda. Both other solutions provide additional packages with more function that your developers and testers are likely to find useful. Remember, the IzODA versions of these packages may have identified security vulnerabilities, and updated versions should be installed as soon as possible.

Archive Important Anaconda User Environments

If your users have existing Anaconda environments that they intend to replicate as pip environments with the Toolkit, they need preserve the package lists they are currently using:

1. List all your existing Anaconda environments, and choose the important ones:

```
~ > cat .conda/environments.txt
~/.conda/envs/jup
~/.conda/envs/pyspeed
```

2. In this example, the user happens to have 2 Anaconda environments already defined - `jup`, and `pyspeed`. We'll preserve the Anaconda environment named jup (`~/.conda/envs/jup`). Activate the environment and list all the packages in it:

```
~ > conda activate jup
(jup) ~ > conda list | tee jup_env_list.txt
# packages in environment at ~/.conda/envs/jup:
#
# Name                    Version   Build   Channel
backcall                  0.1.0     py37_1  file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA
bleach                    2.1.3     py37_0  file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA
bzip2                     1.0.6          2  file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA

(many files in this environment ...)

zeromq                    4.2.1          2  file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA
zlib                      1.2.11         1  file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA
```

3. Save the `jup_env_list.txt` file in a safe location so the user can create a pip recipe that will generate an equivalent environment later in this process. Don't leave your environment package lists under `~/.conda`, because this directory will be removed.

4. Deactivate the jup environment

```
(jup) ~ > conda deactivate
```

```
~ >
```

5. Remove the jup environment

```
~ > conda remove -y -n jup --all

Remove all packages in environment ~/.conda/envs/jup:


## Package Plan ##

  environment location: ~/.conda/envs/jup


The following packages will be REMOVED:

    backcall:  0.1.0-py37_1 file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA
    bleach:    2.1.3-py37_0 file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA
    bzip2:     1.0.6-2      file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA

. . .

    zeromq:    4.2.1-2      file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA
    zlib:      1.2.11-1     file:///usr/lpp/IBM/izoda/anaconda/channels/IzODA


~ >
```

6. Repeat steps 2 through 5 for every Anaconda environment that needs to be preserved.

After all the important environments have been archived and removed, there may be additional environments and miscellaneous packages that remain. Since there is no more need for the contents of the `~/.conda` directory, remove it to ensure there are no unused parts left over:

```
~ > rm -Rf ~/.conda
```

Clean up Your Bash Profile

In order to use the conda command line, each IzODA Anaconda user had to set up their shell session appropriately. Anaconda makes this easy by putting all required setup in a shell script that users can either run interactively, or more often, they include in their .bashrc file. This shell script is run in the current shell session like this:

```
. /usr/lpp/IBM/izoda/anaconda/etc/profile.d/conda.sh
```

Users should check their own .bashrc files for this setup, and other environmental settings that may be related to using the conda command line. In particular, look for references to path variables that include the IzODA path (`/usr/lpp/IBM/izoda/anaconda`). These may be in `PATH` or `LIBPATH`. Also look for any references to the PYTHON environment variables that can be set to control the behavior of the interpreter directly. Look for `PYTHONHOME` and `PYTHONPATH` to make sure that they aren't referencing IzODA Anaconda any longer.

Failing to clean up your runtime environment can result in unpredictable results because you may inadvertently use a down-level, unsupported version of Python. You may also inadvertently include old package versions that contain severe security vulnerabilities, or lack key function that you require.

Remove the System IzODA Anaconda Installation.

Once all users have removed their references to IzODA Anaconda and cleaned up their own environments, the system administrator can remove the IzODA Anaconda product itself via SMP/E. This should be the last step in the process to avoid breaking any users who have a valid requirement for IzODA Anaconda.

Install the IBM Open Enterprise SDK

The first step re-establishing your Python-based AI/ML environment is to install the Python compiler and interpreter, as well as the Python standard library functions, which are all part of the software developer kit (SDK) that can be found at https://www.ibm.com/products/open-enterprise-python-zos. This is a no-cost feature of z/OS with optional service and support.

This provides a current and comprehensive Python environment that can be used for any type of workload, not just AI or machine learning. It has a number of key advantages over the interpreter packaged with IzODA Anaconda:

- *Currency* - this SDK is maintained at the latest level. New versions are provided within weeks of the cross-platform open source community. This enhances both the security and function of the SDK.
- *Performance* - The SDK automatically inherits performance enhancements from the open source community with new Python versions.
- *zIIP offload* - interpreter operations and certain key compiled libraries are marked zIIP eligible. See the product solution page above for details.
- *Pip package management* - as mentioned above, pip is used for package management within the Python AI Toolkit. Since pip is included with the SDK, there is no need to install or manage it separately. Pip can be used to access python packages on pypi.org for those pure Python packages that can install and run on z/OS. For those organizations that allow access to pypi.org in their enterprises, this opens up the Python environment to a large library of available software.

The Python SDK makes Python a first-tier, well-supported language on z/OS, similar to C or Java. Please see the content solution page at the link above for detailed instructions about how to install it.

Install the Python AI Toolkit

The Python SDK and Toolkit are complementary products. Although the Toolkit pre-reqs the SDK, the Toolkit and its associated deployment channel provides functions that extend the function of the Python standard library that is a part of the SDK.

The Toolkit Deployment Channel

The Toolkit (5698-PAL), is available for no cost from ShopZ with optional service and support (5698-PLS) for a fee. Installing the Toolkit offering establishes your entitlement to use the code, and to order service and support. However, the deployment model for the Toolkit closely follows that of the open source community. The code itself does not install through SMP/E. This departure from the typical z/OS installation process allows the toolkit to remain much more current, and align Python use on z/OS with other platforms that most enterprises employ.

The Python AI Toolkit for IBM z/OS deployment model is a secure IBM version of the Python Package Index at pypi.org.

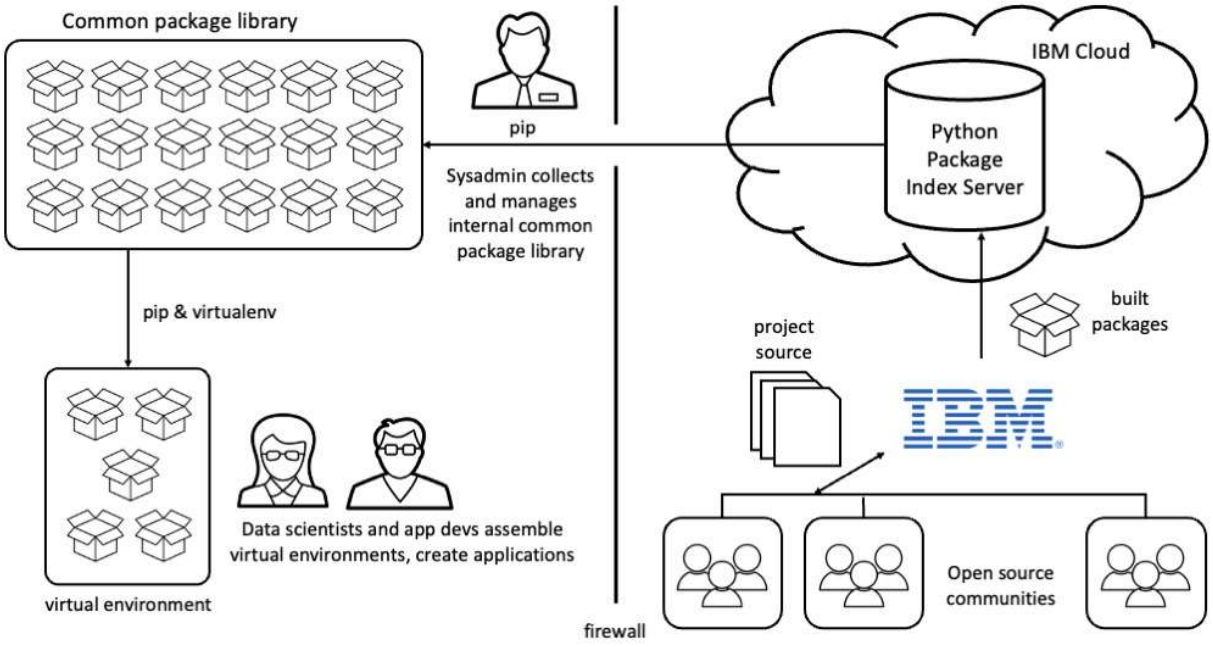

Figure 1 - Suggested Toolkit Architecture

IBM collects and builds all of the packages that are part of the Toolkit from the open source communities that develop the original upstream code. IBM incorporates several security validation and vetting steps during the build and test of the packages posted to the python package index server.

The system administrator uses the pip command interface to collect the packages from the secure IBM server and form a common package library that is internal to the enterprise. If there are additional security vetting steps required by the enterprise, they can be performed on this library before release to the internal developer community.

The pip command is a highly configurable interface that allows users to easily specify which package index server to use. The common ways a user can specify the server to work with include:

- On the pip command line, using the `--index-url`, and `--trusted-host` arguments.
- As settings in a pip configuration file. For an individual user, this is located in $HOME/.config/pip.conf. This is the most flexible option that allows the user to provide server details that can save some time over specifying the options on the command line.
- In a requirements file. For downloading or installing large sets of packages, pip can accept a text file where the full list can be specified. Pip allows many command line arguments to be specified in this file as well, effectively making the package list into a complete recipe for what to install, where to acquire it, and how to complete the request.

See the online documentation at https://pip.pypa.io/en/stable/ for all the details about how to control the actions performed by pip.

If no configuration information is specified, pip will default to https://pypi.python.org. The IBM server is provided at https://downloads.pyaitoolkit.ibm.net

Installing Individual Packages, or the Whole Toolkit

In addition to the Python Package Index server, IBM provides a web interface at https://ibm-z-oss-oda.github.io/python_ai_toolkit_zos/ that lists details about all the packages available in the Toolkit. System administrators can install or download individual packages from the information provided here. Here is an example of the information provided for the Pandas package:

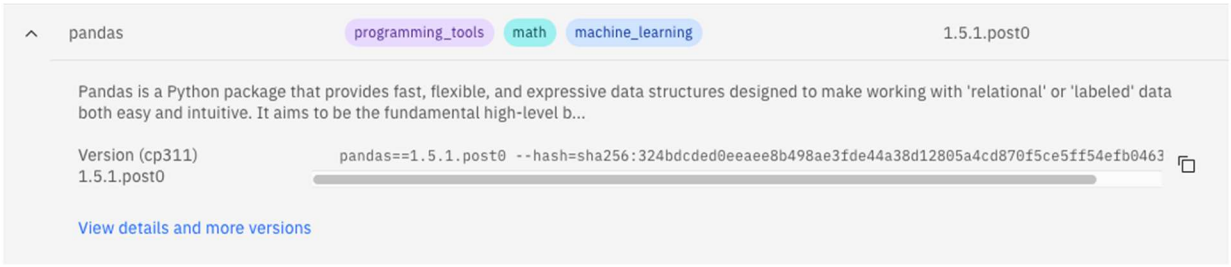

Figure 2 - details about an individual Project

Additionally, the entire Toolkit can be downloaded or installed as a whole using a reference requirements file available at:

https://github.com/ibm-z-oss-oda/python_ai_toolkit_zos/tree/main/requirements

There is a link to this github repository from the ***Get Started*** section of the Toolkit web interface. Here is an example of a Toolkit requirements file.

```
#####
#
# Python AI Toolkit for IBM z/OS
#
# This is the complete list of packages that make up the Python AI
# Toolkit for IBM z/OS.
#
# Example use:  pip install --no-deps -r <toolkit_requirements.txt>
#
# pip command options:
--index-url https://downloads.pyaitoolkit.ibm.net/repository/python_ai_toolkit_zos/simple
--trusted-host downloads.pyaitoolkit.ibm.net
--require-hashes
--only-binary :all:

#
# This requirements file was generated for a cp311 environment.
#

Cython==0.29.32 --hash=sha256:6e747bc6bb238fb732718c9a2ab60cc84285a08f429e2d725ee6712bff632f2f
Flask-RESTful==0.3.9 --hash=sha256:d0a42c2a37cc486511f5bc7eaa291aaf0f679ffb0eac7ed0fc096629bb921325
Flask==2.2.5 --hash=sha256:620f433f84d00ec3b5375357fd7f9f7e7e53087552d5a5618e2acfb89bf3fad4
HeapDict==1.0.1 --hash=sha256:8faa6f10f7581117cad3195cb00d80792b08f7c534c99b1e294b869b6eb0c94f

. . .
```

Example 1 - a sample pip requirements file

You can see the pip configuration settings for reaching the IBM Python Package Index server, along with some other settings provided to enhance security.

In both figure 2, and example 1 you can see that the `--hash` argument is included. This is a security measure that we strongly recommend always using, because it specifies the exact instance of the package you are installing or

downloading. It prevents you from inadvertently installing the wrong version, or using an unintended python package index server, instead of the IBM server.

The hash argument is only honored when using a requirements file with the pip install or download command. It can't be specified from the command line. If you are installing only one package, or a small collection of packages, we recommend that you build your own requirements file, and pass it to the pip command with the `-r` argument. Although passing your requests to the pip command like this involves extra steps, it's the most secure way to acquire Python software from outside your firewall.

Note that when installing packages, pip does much more than just copy bytes over from the package server. By default, it checks for all dependencies that a given package may have on other packages, and it brings those along too. Python packages have a well-developed mechanism for specifying package dependencies on other package versions, and the level of the Python interpreter. This works well when installing small numbers of packages at once, but the dependency checker can take a long time to work with larger numbers of packages. In some cases, pip's checker can simply fail because the dependencies are too complex, even though all the packages successfully install individually.

The Python AI Toolkit for IBM z/OS has a closed dependency graph. It contains the packages necessary to satisfy all the dependency requirements for all other packages in the Toolkit. For this reason, we can tell pip to avoid dependency checking when installing the full toolkit. This makes installation much faster and avoids any potential problems with pip's dependency checker.

The example requirements file above has a comment that shows use of the pip command using the `--no-deps` argument. This disables pip command dependency checking. Be sure to use this if installing the entire toolkit, or else pip is likely to flag a dependency error.

Virtual Environments

Virtual environments are a built-in feature of Python and are implemented in the Python SDK. Although not required, we recommend using them because they help a lot with managing groups of specific packages. These serve the same function as Anaconda environments in IzODA. If you don't download or install packages using a virtual environment, they will be installed under your `$HOME/.local/lib/python3.xx` directory (where python3.11 is the version of your python interpreter) and can be managed from there. All the install and download examples below use virtual environments because they make handling all the packages as a group easier.

Full details on how to use pip virtual environments can be found at https://docs.python.org/3/library/venv.html. Here is a simple example of how we set up the virtual environment for this installation when using the bash shell. If you are using the default USS shell in `/bin/sh`, please substitute the dot command (".") for the source command below. Either way, you want to activate the virtual environment in the current shell process in order to make the settings of the new virtual environment take effect :

```
> bash
> python3 -m venv myenv
> source myenv/bin/activate
(myenv) >
```

The python3 command creates a virtual environment named `myenv` (we could have called it anything), and the source command makes the environment active. Note that the activate command also changes your command prompt to remind you that you are working in the `myenv` virtual environment. Use the `exit` command to exit the environment and the shell session you used to activate the environment. Note that if you don't start a nested shell like `bash` or `sh` as we have in the example above, the exit command will end your USS session with the host, and you will have to log in again.

Python Requirements - Your Download and Install Recipe

The toolkit is stored as a collection of built packages called ***wheels***. These are complete builds of each package that simplify installation and reduce opportunities for bad actors to corrupt the package set. The full recipe for managing the packages of the Toolkit, called a requirements file, can be downloaded from the Toolkit's public Github repository. This requirements file is published on a regular cadence (at least quarterly) to list the latest versions and hashes for all the packages of the Toolkit. You pass this file to pip during the download or installation process.

You can download this file to a workstation, and then transfer it to your USS environment on z/OS.

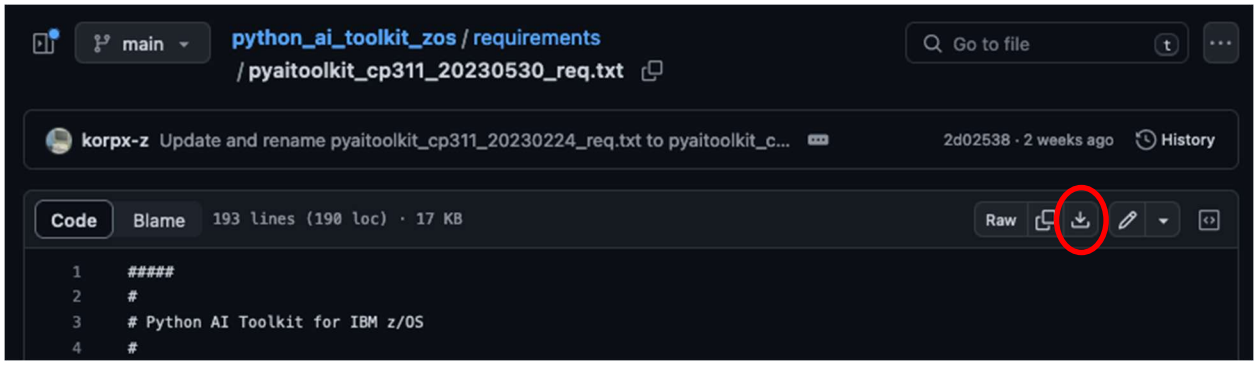

Figure 3 - a requirements file in the Toolkit public repository

There are many ways to perform the file transfer to your host system, including cut/paste of the text in the requirements file. However you manage to get this file to z/OS, it has to be in ASCII form, and tagged as an ascii file. If you find that your file is in EBCDIC, you can convert it to ascii like this:

```
(myenv) > iconv -f IBM-1047 -t ISO8859-1 pyaitoolkit_cp311_20230530_req.txt > temp.txt
(myenv) > mv -f temp.txt pyaitoolkit_cp311_20230530_req.txt
```

Now tag this as an ASCII file. If you transferred the file up from your workstation using a command like sftp, the file will most likely already be in ASCII, and all you have to do is tag it:

```
(myenv) > chtag -tc ISO8859-1 pyaitoolkit_cp311_20230530_req.txt
```

You can check the file tag like this:

```
(myenv) > ls -alT pyaitoolkit_cp311_20230530_req.txt
t ISO8859-1   T=on  -rw-r-----   1 BOSTIAN  DEPTD60    17425 Jun 23 15:18
pyaitoolkit_cp311_20230530_req.txt
```

The tag will be ISO8859-1 rather than IBM-037 or IBM-1047. Now you're ready to download the Toolkit.

Download the Toolkit

As shown in figure 1 above, a best practice is to build a common package library in your enterprise from which all users can build their virtual environments. This minimizes traffic through the firewall to the IBM server and allows further testing and vetting that your organization may require before deployment to a production environment.

Here is an example that uses the pip interface to download all the packages of the Toolkit to a common location. It assumes that you have system administrator capabilities to create and manage files in certain system locations. ***Please note that the directories used here are for illustration purposes and represent one useful configuration.***

Other locations can be used for download and installation of the Toolkit, and you should use the convention that fits your enterprise best.

On Unix-based systems `/usr/local` is often used to provide content that is shared between users. First, create a directory under here called `pyaitoolkit` that will be home to all the packages and other assets to create a common Toolkit installation. As mentioned above, all the packages of the toolkit are built into installable wheel files. We will create a directory under `/usr/local/pyaitoolkit` called `wheels` that will hold all the downloaded packages. You can continue to use the `myenv` environment that we have been using above to run the pip command to download all the wheels. Here is what the whole process looks like:

```
(myenv) > cd /usr/local
(myenv) /usr/local/> mkdir pyaitoolkit
(myenv) /usr/local/> cd pyaitoolkit
(myenv) /usr/local/pyaitoolkit/>
(myenv) /usr/local/pyaitoolkit/> mkdir wheels
(myenv) /usr/local/pyaitoolkit/> cd wheels
(myenv) /usr/local/pyaitoolkit/wheels>
```

Make sure that the permissions are set for the `/usr/local/pyaitoolkit` directory that allow shared read-only access to the downloaded wheels. Failure to do this will result in users not being able to access the toolkit.

Move the Python requirements file for the Toolkit to the wheels directory (this example assumes that you downloaded or created this file in your `$HOME` directory). This file should have read-only permissions for other users as well.

```
(myenv) /usr/local/pyaitoolkit/wheels/> mv $HOME/pyaitoolkit_cp311_20230530_req.txt .
```

There needs to be about 3 GB of free space available in the USS filesystem for `/usr/local/pyaitoolkit` to download the whole Toolkit and perform other necessary installation steps. Now we are ready to download the Toolkit via pip:

```
(myenv) /usr/local/pyaitoolkit/wheels> pip download --no-deps --platform s390x --python-version 311 -r
/usr/local/pyaitoolkit_cp311_20230530_req.txt
adding trusted host: 'downloads.pyaitoolkit.ibm.net' (from line 12 of
/usr/local/pyaitoolkit_cp311_20230530_req.txt)
Looking in indexes: https://downloads.pyaitoolkit.ibm.net/repository/python_ai_toolkit_zos/simple
Collecting Cython==0.29.32
  Downloading
https://downloads.pyaitoolkit.ibm.net/repository/python_ai_toolkit_zos/packages/cython/0.29.32/Cython-
0.29.32-cp311-none-any.whl (3.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 10.1 MB/s eta
0:00:00
Collecting Flask-RESTful==0.3.9
  Downloading https://downloads.pyaitoolkit.ibm.net/repository/python_ai_toolkit_zos/packages/flask-
restful/0.3.9/Flask_RESTful-0.3.9-py2.py3-none-any.whl (28 kB)
Collecting Flask==2.2.5
  Downloading
https://downloads.pyaitoolkit.ibm.net/repository/python_ai_toolkit_zos/packages/flask/2.2.5/Flask-
2.2.5-py3-none-any.whl (117 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 117.4/117.4 kB 24.4 MB/s eta
0:00:00

. . .

Saved ./Cython-0.29.32-cp311-none-any.whl
Saved ./Flask_RESTful-0.3.9-py2.py3-none-any.whl
Saved ./Flask-2.2.5-py3-none-any.whl
Saved ./HeapDict-1.0.1-py3-none-any.whl

. . .

Successfully downloaded Cython Flask-RESTful Flask HeapDict JPype1 JayDeBeApi Jinja2 Mako Markdown-
Editor Markdown MarkupSafe Pillow PyWavelets PyYAML Pygments QtPy SQLAlchemy Send2Trash Werkzeug Whoosh
alembic aniso8601 argon2-cffi-bindings argon2-cffi asttokens async-generator attrs backcall
beautifulsoup4 bleach bottle bz2file cairocffi certifi cffi chardet charset-normalizer click
```

```
cloudpickle cycler cytoolz dask debugpy decorator defusedxml distlib docutils ebcdic entrypoints
executing fastjsonschema filelock fontconfig fonttools freetype2 fsspec future glib graphviz harfbuzz
html5lib idna imageio importlib-metadata importlib-resources iniconfig ipykernel ipython-genutils
ipython ipywidgets itsdangerous jedi joblib jsonschema jupyter-client jupyter-cms jupyter-console
jupyter-kernel-gateway jupyter jupyter_core jupyterlab-pygments jupyterlab-widgets kiwisolver libcairo
libjpeg-turbo libpng libtiff libxml2 libxslt libyaml locket lxml matplotlib-inline matplotlib mistune
mock msgpack nbclient nbconvert nbformat nest-asyncio networkx notebook numpy oauthlib packaging pamela
pandas-datareader pandas pandocfilters parso partd patsy pexpect pickleshare platformdirs pretend
prometheus-client prompt-toolkit ptyprocess pure-eval pyasn1 pycparser pyftpdlib pymdown-extensions
pyparsing pyrsistent python-dateutil python-json-logger pytz pyzmq qtconsole readme-renderer requests-
file requests-ftp requests-toolbelt requests ruamel.yaml.clib ruamel.yaml scikit-learn scipy seaborn
semantic-version setuptools-rust six sklearn-pandas sortedcontainers soupsieve sqlparse stack-data
statsmodels tblib tdqm terminado threadpoolctl tinycss2 toolz toree tornado tqdm traitlets
typing_extensions uWSGI urllib3 virtualenv wcwidth webencodings widgetsnbextension xgboost zict zipp
zos-util setuptools
```

In this example, the pip download command is run with the following parameters:

- `--no-deps`: do not check the dependencies for each package
- `--platform`: download packages for the S/390 hardware platform
- `--python-version`: use the python 3.11 version of packages that have been also built for other python versions (e.g. python 3.10)
- `-r`: the name of the requirements file. The name of the requirements file in the Toolkit public repository will contain the python version, and a date so that you can choose the version you need. You should generally choose the latest version available to get the most current code.

The toolkit is now downloaded. If you list the files in this directory, you will see one `.whl` file for each package of the Toolkit.

Create a full Toolkit Installation

Just as we saw that the entire Toolkit package set can be downloaded at once, environments can be created with all the Toolkit packages installed. Generally this will be the case for AI/ML offerings higher in the stack, such as IBM Watson Machine Learning for z/OS (WMLz). Note that this Toolkit is a pre-requisite for WMLz.

First, exit the `myenv` virtual environment we've been using so that we can create a new common environment (`commenv`) in `/usr/local/pyaitoolkit` for shared use.

```
(myenv) /usr/local/pyaitoolkit/wheels> exit
/usr/local/pyaitoolkit/wheels> cd ..
/usr/local/pyaitoolkit> bash
/usr/local/pyaitoolkit> python3 -m venv commenv
/usr/local/pyaitoolkit> source commenv/bin/activate
(commenv) /usr/local/pyaitoolkit/> ls
commenv  wheels
(commenv) /usr/local/pyaitoolkit/>
```

Note that we now have 2 directories under `/usr/local/pyaitoolkit` - one for the package wheel files, and a new common virtual environment where we will install the packages. Here is the pip command to install the wheels to commenv:

```
(commenv) /usr/local/pyaitoolkit/wheels> pip install --no-deps --no-index --find-links ./wheels -r
pyaitoolkit_cp311_20230530_req.txt
adding trusted host: 'downloads.pyaitoolkit.ibm.net' (from line 12 of
pyaitoolkit_cp311_20230530_req.txt)
Looking in links: ./wheels
Processing ./wheels/Cython-0.29.32-cp311-none-any.whl
Processing ./wheels/Flask_RESTful-0.3.9-py2.py3-none-any.whl
Processing ./wheels/Flask-2.2.5-py3-none-any.whl
Processing ./wheels/HeapDict-1.0.1-py3-none-any.whl

. . .
```

```
Installing collected packages: zos-util, Whoosh, webencodings, wcwidth, uWSGI, toree, tdqm, stack-data,
sortedcontainers, sklearn-pandas, Send2Trash, requests-ftp, requests-file, pytz, pyftpdlib, pure-eval,
ptyprocess, pretend, pickleshare, pexpect, patsy, pamela, msgpack, mistune, Markdown-Editor, libyaml,
libxslt, libxml2, libtiff, libpng, libjpeg-turbo, libcairo, jupyter-cms, jupyter, JayDeBeApi, ipython-
genutils, HeapDict, harfbuzz, glib, freetype2, fontconfig, Flask-RESTful, fastjsonschema, executing,
ebcdic, distlib, charset-normalizer, cffi, bz2file, bottle, backcall, asttokens, aniso8601, zipp, zict,
xgboost, widgetsnbextension, Werkzeug, virtualenv, urllib3, typing_extensions, traitlets, tqdm,
tornado, toolz, tinycss2, threadpoolctl, terminado, tblib, statsmodels, sqlparse, SQLAlchemy,
soupsieve, six, setuptools-rust, setuptools, semantic-version, seaborn, scipy, scikit-learn,
ruamel.yaml.clib, ruamel.yaml, requests-toolbelt, requests, readme-renderer, QtPy, qtconsole, pyzmq,
PyYAML, PyWavelets, python-json-logger, python-dateutil, pyrsistent, pyparsing, pymdown-extensions,
Pygments, pycparser, pyasn1, prompt-toolkit, prometheus-client, platformdirs, Pillow, partd, parso,
pandocfilters, pandas-datareader, pandas, packaging, oauthlib, numpy, notebook, networkx, nest-asyncio,
nbformat, nbconvert, nbclient, mock, matplotlib-inline, matplotlib, MarkupSafe, Markdown, Mako, lxml,
locket, kiwisolver, jupyterlab-widgets, jupyterlab-pygments, jupyter-kernel-gateway, jupyter_core,
jupyter-console, jupyter-client, jsonschema, JPype1, joblib, Jinja2, jedi, itsdangerous, ipywidgets,
ipython, ipykernel, iniconfig, importlib-resources, importlib-metadata, imageio, idna, html5lib,
graphviz, future, fsspec, fonttools, Flask, filelock, entrypoints, docutils, defusedxml, decorator,
debugpy, dask, cytoolz, Cython, cycler, cloudpickle, click, chardet, certifi, cairocffi, bleach,
beautifulsoup4, attrs, async-generator, argon2-cffi-bindings, argon2-cffi, alembic
  Attempting uninstall: setuptools
    Found existing installation: setuptools 65.6.3
    Uninstalling setuptools-65.6.3:
      Successfully uninstalled setuptools-65.6.3
Successfully installed Cython-0.29.32 Flask-2.2.5 Flask-RESTful-0.3.9 HeapDict-1.0.1 JPype1-1.4.1
JayDeBeApi-1.2.3 Jinja2-3.1.2 Mako-1.2.4.dev0 Markdown-3.4.1.post0 Markdown-Editor-1.0.7.post1
MarkupSafe-2.1.1 Pillow-9.3.0 PyWavelets-1.3.0 PyYAML-6.0.post0 Pygments-2.14.0 QtPy-2.1.0 SQLAlchemy-
2.0.12.dev0 Send2Trash-1.8.0 Werkzeug-2.3.4 Whoosh-2.7.4 alembic-1.10.4.dev0 aniso8601-9.0.1 argon2-
cffi-21.3.0 argon2-cffi-bindings-21.2.0.post0 asttokens-2.2.1 async-generator-1.10 attrs-22.2.0
backcall-0.2.0 beautifulsoup4-4.11.1 bleach-6.0.0.post1 bottle-0.12.23 bz2file-0.98 cairocffi-
1.3.0.post1 certifi-2022.12.7 cffi-1.14.6 chardet-5.0.0 charset-normalizer-3.0.1.post0 click-
8.1.3.post0 cloudpickle-2.0.0 cycler-0.11.0 cytoolz-0.12.0 dask-2021.12.0 debugpy-1.6.3.post1
decorator-5.1.1 defusedxml-0.7.1.post0 distlib-0.3.6 docutils-0.19.post0 ebcdic-1.1.1 entrypoints-0.4
executing-1.2.0 fastjsonschema-2.16.2 filelock-3.12.0 fontconfig-2.14.1.post0 fonttools-4.33.3.post0
freetype2-2.12.2.post0 fsspec-2022.7.0 future-0.18.3.post1 glib-2.56.4.post0 graphviz-0.20.1 harfbuzz-
7.0.0 html5lib-1.1.post1 idna-3.4 imageio-2.13.5 importlib-metadata-6.0.0 importlib-resources-5.8.0
iniconfig-2.0.0 ipykernel-6.9.1 ipython-8.13.2.post0 ipython-genutils-0.3.0 ipywidgets-8.0.6
itsdangerous-2.1.2 jedi-0.18.2 joblib-1.2.0.post0 jsonschema-4.17.3 jupyter-1.1.0 jupyter-client-
7.3.5.post1 jupyter-cms-0.7.0 jupyter-console-6.4.4 jupyter-kernel-gateway-2.5.1 jupyter_core-5.2.0
jupyterlab-pygments-0.2.2.post0 jupyterlab-widgets-3.0.0 kiwisolver-1.4.4 libcairo-1.17.6 libjpeg-
turbo-2.1.4.post0 libpng-1.6.39.post0 libtiff-4.4.0.post0 libxml2-2.10.3.post0 libxslt-1.1.37.post0
libyaml-0.2.5.post0 locket-1.0.0 lxml-4.9.1.post0 matplotlib-3.5.0.post1 matplotlib-inline-0.1.6
mistune-0.8.3 mock-5.0.1 msgpack-1.0.4 nbclient-0.7.0 nbconvert-6.5.0.post0 nbformat-5.4.0 nest-
asyncio-1.5.5 networkx-3.0b1.post0 notebook-6.4.12 numpy-1.23.4+3.gc311554fd oauthlib-3.2.2 packaging-
23.0 pamela-1.0.0 pandas-1.5.1.post0 pandas-datareader-0.10.0 pandocfilters-1.5.0 parso-0.8.3 partd-
1.2.0 patsy-0.5.3 pexpect-4.8.0 pickleshare-0.7.5 platformdirs-3.5.1 pretend-1.0.9 prometheus-client-
0.14.0 prompt-toolkit-3.0.36 ptyprocess-0.7.0 pure-eval-0.2.2 pyasn1-0.4.9 pycparser-2.20 pyftpdlib-
1.5.7 pymdown-extensions-10.0.1 pyparsing-3.0.10 pyrsistent-0.19.3 python-dateutil-2.8.2 python-json-
logger-2.0.7 pytz-2022.4 pyzmq-24.0.1.post0 qtconsole-5.3.2 readme-renderer-37.3 requests-2.28.2
requests-file-1.5.1 requests-ftp-0.3.1 requests-toolbelt-0.10.1 ruamel.yaml-0.17.24 ruamel.yaml.clib-
0.2.7 scikit-learn-1.2.1.post1 scipy-1.3.3 seaborn-0.11.0 semantic-version-2.10.0 setuptools-67.1.0
setuptools-rust-1.6.0 six-1.16.0 sklearn-pandas-2.2.0 sortedcontainers-2.4.0 soupsieve-2.4.post1
sqlparse-0.4.4 stack-data-0.6.2 statsmodels-0.13.5 tblib-1.7.0 tdqm-0.0.1 terminado-0.15.0
threadpoolctl-3.1.0 tinycss2-1.1.1 toolz-0.12.0 toree-0.5.0.post0 tornado-6.2 tqdm-4.64.1 traitlets-
5.9.0 typing_extensions-4.4.0 uWSGI-2.0.21 urllib3-1.26.14 virtualenv-20.23.0 wcwidth-0.2.6
webencodings-0.5.1 widgetsnbextension-4.0.7 xgboost-1.6.2 zict-2.2.0 zipp-3.11.0 zos-util-1.0.1
```

You can now list all the packages that have been installed to `commenv`:

```
(commenv) /usr/local/pyaitoolkit> pip list
Package                Version
---------------------- -------------------
alembic                1.10.4.dev0
aniso8601              9.0.1
argon2-cffi            21.3.0
argon2-cffi-bindings   21.2.0.post0
asttokens              2.2.1

. . .
```

This environment is now available for Python users to activate.

Custom Virtual Environments for Individuals

Application developers and data scientists who write applications directly to the interfaces provided by this Toolkit will generally create virtual environments with selected sets of packages installed rather than the full contents of the Toolkit. Any users who archived their Anaconda environments from the IzODA installation can now convert those to a pip requirements file. One way to accomplish this is to make a copy of the requirements file from the Toolkit wheels directory that we used during download and edit it to include only the packages that are listed in their archived Anaconda environments. This way they can be sure to request the proper versions of the packages they need, since these version numbers will be different from IzODA Anaconda versions.

In this example, a user has created a requirements file that includes only the subset of packages they need:

```
#####
#
# A custom requirements file to build a virtual environment for an individual user.
#
# Example use:  pip install --no-index --find-links /usr/local/pyaitoolkit/wheels -r
my_requirements.txt
#
Cython==0.29.32
Flask-RESTful==0.3.9
kiwisolver==1.4.4
pandas-datareader==0.10.0
pandas==1.5.1.post0
scikit-learn==1.2.1.post1
scipy==1.3.3
```

In this example there are a few things to note:

- Since this user is installing from the common local wheels directory, there will be no request made to an outside server. The `--no-index` and `--find-links` parameters tell pip to look in `/usr/local/pyaitoolkit/wheels/` for the packages to install.
- Hash checks were performed when the Toolkit's wheels were downloaded from IBM to verify their authenticity, so there is no need to perform them again. The hashes can be removed from the requirements file so that it contains only package names and versions.
- We want pip to check for dependencies between packages and include any packages that are not explicitly listed in the requirements file. This is the default behavior.
- These packages will be installed to a user-specific virtual environment called myenv in this example.

```
> python3 -m venv myenv
> source myenv/bin/activate
(myenv) > pip install --no-index --find-links /usr/local/pyaitoolkit/wheels/ -r my_requirements.txt
Looking in links: /usr/local/pyaitoolkit/wheels/
Processing /usr/local/pyaitoolkit/wheels/Cython-0.29.32-cp311-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/Flask_RESTful-0.3.9-py2.py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/kiwisolver-1.4.4-cp311-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/pandas_datareader-0.10.0-py2.py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/pandas-1.5.1.post0-cp311-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/scikit_learn-1.2.1.post1-cp311-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/scipy-1.3.3-cp311-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/defusedxml-0.7.1.post0-py2.py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/aniso8601-9.0.1-py2.py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/Flask-2.2.5-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/six-1.16.0-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/pytz-2022.4-py2.py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/lxml-4.9.1.post0-cp311-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/requests-2.28.2-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/python_dateutil-2.8.2-py2.py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/numpy-1.23.4+3.gc311554fd-cp311-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/joblib-1.2.0.post0-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/threadpoolctl-3.1.0-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/Werkzeug-2.3.4-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/Jinja2-3.1.2-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/itsdangerous-2.1.2-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/click-8.1.3.post0-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/charset_normalizer-3.0.1.post0-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/idna-3.4-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/urllib3-1.26.14-py2.py3-none-any.whl
```

```
Processing /usr/local/pyaitoolkit/wheels/certifi-2022.12.7-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/libxml2-2.10.3.post0-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/libxslt-1.1.37.post0-py3-none-any.whl
Processing /usr/local/pyaitoolkit/wheels/MarkupSafe-2.1.1-py2.py3-none-any.whl
Installing collected packages: pytz, libxslt, libxml2, charset-normalizer, aniso8601, urllib3,
threadpoolctl, six, numpy, MarkupSafe, lxml, kiwisolver, joblib, itsdangerous, idna, defusedxml, click,
certifi, Werkzeug, scipy, requests, python-dateutil, Jinja2, Cython, scikit-learn, pandas, Flask,
pandas-datareader, Flask-RESTful
Successfully installed Cython-0.29.32 Flask-2.2.5 Flask-RESTful-0.3.9 Jinja2-3.1.2 MarkupSafe-2.1.1
Werkzeug-2.3.4 aniso8601-9.0.1 certifi-2022.12.7 charset-normalizer-3.0.1.post0 click-8.1.3.post0
defusedxml-0.7.1.post0 idna-3.4 itsdangerous-2.1.2 joblib-1.2.0.post0 kiwisolver-1.4.4 libxml2-
2.10.3.post0 libxslt-1.1.37.post0 lxml-4.9.1.post0 numpy-1.23.4+3.gc311554fd pandas-1.5.1.post0 pandas-
datareader-0.10.0 python-dateutil-2.8.2 pytz-2022.4 requests-2.28.2 scikit-learn-1.2.1.post1 scipy-
1.3.3 six-1.16.0 threadpoolctl-3.1.0 urllib3-1.26.14
```

Custom virtual environments like this are a common way to use sets of Python packages. These allow users tune their runtime configurations to the specific needs of their applications, rather than use a single library of functions. They can be created and deleted as needed, making experimentation and testing easier to manage. The requirements files that describe virtual environments can also be passed along with the application as a complete specification of the application's needs, making the application more portable between platforms.

Isolated Production Environments

Since z/OS installations are often isolated from internet access, and can only be reached from systems internal to the enterprise, how does one create a common package library as outlined here? Such "air-gapped" systems can be accommodated with this package management architecture as long as there is another machine behind the enterprise firewall with access to the IBM python package index server (`downloads.pyaitoolkit.ibm.net`). This can be a non-production z/OS LPAR, a linux on Z system, or an x/86 workstation. The platform doesn't matter because Python is an environment that can accommodate any system that provides the pip command line and Python interpreter.

For these isolated environments, perform all the Toolkit installation steps up through the download of the wheel files onto the non-production system. You probably will want to do this in a directory created specifically for the wheel files for convenience purposes.

Once the download step is complete, transfer all the wheels from the download system to your target production z/OS machine at a common location as described in the download section above. You may want to archive or zip up all the wheels into a single file for easier transfer. Once the wheels are on your target production system, you can proceed with the rest of the installation steps as usual in the production environment.

Some enterprises that have deeper backgrounds in open source software may choose to locate their wheel files on an internal cross-platform package repository like Artifactory or Nexus Repository Manager. From there, packages can be installed into the production environment as they would be for any other platform in the enterprise. There is no more special installation process required for z/OS. These types of common repository configurations are beyond the scope of this document, but if you have such a setup, the Python AI Toolkit for IBM z/OS will fit in much better than IzODA Anaconda ever did.

Notes on Security and Maintenance

Although the deployment channel for the Python AI Toolkit for IBM z/OS does not use SMP/E to install the executable content of the product, the installation process is secure. Several key characteristics of the Python package management architecture have been leveraged to ensure the security of the package deployment channel.

- IBM controls all content posted on the Toolkit package server. This contrasts with community servers like pypi.org where any user can post a package, allowing bad actors to spoof system administrators and automated workflows into installing a compromised package mistakenly.
- IBM publishes hashes that identify the unique instance of each package to download. This prevents inadvertent download from a mis-configured pip environment. It means your download request will result you receiving only what IBM provides.
- IBM hosts the python package index server in the IBM cloud and inherits all the security characteristics of that environment.
- Every package posted to the package server has undergone several types of scans that detect known problems, viral licenses, and poor programming practices. All integrated tests for each package also run successfully before a package is posted on the server for download.

There are two key reasons for the shift from an SMP/E installation process to the pip mechanism:

- Packages can be downloaded and installed individually rather than replacing the entire Toolkit product. This greatly enhances the agility with which IBM can respond to emerging threats. When a vulnerability for a package becomes known, it can be patched and posted on the package server immediately without the overhead of re-building the entire product. Updates to the Toolkit's requirements file on a regular cadence effectively allow a periodic (usually quarterly) roll-up of recent updates.
- Any modern open language like Python should be thought of as a complete environment, and not just a language. It includes not only the compiler/interpreter and associated libraries, but it also provides the package management interfaces and environmental mechanisms to enhance the deployment of code. By adopting Python's package and environmental management mechanisms, applications can more easily move to z/OS, where key enterprise data is kept. It aligns z/OS better with the broader enterprise and lowers barriers to adoption.

The goal of the Toolkit replacement for IzODA Anaconda is to give administrators and users the means to keep their Python environments current. This is key to keeping open source configurations secure.