ModelOp Center v2.x:
Customizing the ModelOp Runtime

AUGUST 2020

DOCUMENT PURPOSE

The following document provides an overview of how to customize the ModelOp runtime to incorporate specific libraries/versions required to execute your model. The ModelOp Field Support team will work with you to integrate into your specific production logging systems and processes.

TABLE OF CONTENTS

Customizing Python Packages

In order to make particular Python packages available to Python models which execute in the ModelOp Center Runtime, you must install those packages on the Runtime. This is typically done via a Docker container built using the Runtime containers as a base. In the vast majority of cases, minor variations on the following Dockerfile will suffice in defining the requisite container:

```
FROM modelop/bofa-engine:2.1-rc9-centos

USER 0
COPY requirements.txt .
RUN pip install -r requirements.txt
USER 1000
```

Here the requirements.txt file must reside in the same directory as the Dockerfile. The requirements.txt lists the Python packages to be installed as long as requirements on the version numbers. For example,

```
scikit-learn==0.21.3
shap==0.35.0
aequitas==0.38.0
xgboost==1.1.1
```

It is a common pattern to have the base image (modelop/bofa-engine:2.1-rc9-centos) reside in an internal repository such as Artifactory rather than pulling the image from DockerHub each time. Similarly, it is very common to have a curated internal repository of Python libraries which are pre-approved for use by data scientists and other users. pip can be configured to use such an internal repository simply by adding the "--index-url" flag in the pip install command in the Dockerfile with the correct URL for the internal repository or by setting the PIP_INDEX_URL environment variable before running the pip install command (so inserting the line:

```
ENV PIP_INDEX_URL=<your-repo-url>
```

just before the

```
RUN pip install -r requirements.txt
```

line.

In a small number of cases, a Python package may require the presence of certain non-Python dependencies such as particular compiled C binaries. In these cases, the base container's package manager (e.g. yum for CentOS) can be utilized to ensure that the prerequisites are present. This is an exceedingly rare occurrence.

Customizing R Packages

The Dockerfile in the case of adding R packages is structured similarly:

```
FROM modelop/bofa-engine:2.1-rc4-centos8

USER 0
COPY requirements.R .
RUN Rscript requirements.R
USER 1000
```

Here requirements.R is a script which performs the installation. Here is an example of such a script:

```
r <-  getOption("repos")
r["CRAN"] <- "cran.r-project.org"
options(repos=r)
install.packages(c("zoo", “dplyr”, “caret”), dependencies=T)
```

Here “cran.r-project.org” can be replaced with a local CRAN repository just as PyPI can be replaced with a local Python repository. You can also specify which version of the package to use by specifying the URL to pull it from e.g.:

```
packageurl <- "http://cran.r-project.org/src/contrib/Archive/ggplot2/ggplot2_0.9.1.tar.gz"
install.packages(packageurl, repos=NULL, type="source")
```

Here packages are being installed from source. This can be desirable in many situations, but packages can also be added as precompiled binaries. This can be more efficient if the same package is used often across various models.