IBM Maximo PQI On-Premises Visual Insights

User Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 143.

First edition (March 2019)

Contents

Chapter 1. Product overview

IBM® Maximo® PQI On-Premises Visual Insights is a quality monitoring and alerting solution that can take in images of in-process and finished products and assemblies, and classify them into defect categories.

Roles

To understand Maximo PQI On-Premises Visual Insights, it is helpful to understand how the different roles interact with the product.

Table 1. Maximo PQI On-Premises Visual Insights roles

Role	Description
Model Manager	Manages defect types and models, uploads image sets for specific defect types, trains models, and distributes executable models to edges.
Inspector	Verifies the inspection results that are produced by the product, changes defect types if necessary, marks unknown defect types and passes them to the Inspector Supervisor for further evaluation.
Inspector Supervisor	Double-checks the Inspector's inspection results. Reviews and classifies unknown defect types. Reviews the KPI dashboard, which includes defect per unit and defect rate.

What's new in this release

The following new features are available in IBM Maximo PQI On-Premises Visual Insights.

New in this release

January 2019

- The product name was changed to IBM Maximo PQI On-Premises Visual Insights.
- Support for multiple edge clusters in one tenant. Models can be deployed to multiple edge clusters.
- Support for training server clusters to scale out during multiple training job submissions to reduce user's wait time and improve overall performance.
- Added an image collection policy to specify what kinds of images are archived and used to retrain a model.
- Fixed a training curve issue when training classification models in parallel. Fixed an issue when retraining YOLO models. Fixed an issue when validating and scoring FRCNN-VGG16 models.
- Improved error messaging with details during scoring to help users identify problems.
- Updated pricing metrics to count training file number and scoring file number.

New in previous releases

September 2018

- Support for online model training. You can choose different model types, algorithms, networks, and hyper parameters. Model files are generated automatically without the need for other training tools.

- Training dashboard that shows a live chart of model loss and accuracy values with iterations and detailed log files.
- Three-step API guidance to help service integration for model registration, deployment, and testing.
- Ability to automatically label training images through a small annotated data set, which reduces the cost and time required to annotate the data.
- Support for a stand-alone edge to be deployed in a private company network.
  - Enables you to store inspection results in a local database.
  - Enables you to synchronize data with the center application when the edge connects to the internet.
- Support for the training server and edge running on the Linux on Power Systems Servers platform.

December 2017

- You can now use image groups to represent the same type of images by using one or more compressed image files.
- Added support for multiple model versions that share the same image groups, but use different image files to train the model.
- The model retrain process was added. You can automatically or manually retrain a new model version by using different image files.
- Added support for model validation. The validation process calculates and shows a model accuracy report based on validation image files.
- You can now use defect boxes and defect types on an image to mark the defect location for the inspector.
- You can now show multiple defect locations on one image. You can now add, adjust, and delete defect boxes on an image.
- Updated the KPI dashboard to include defect per unit and defect rate for the inspector supervisor.
- Added support for the object detection model to detect multiple defects in one image with the CNN model.

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products.

For information about the commitment that IBM has to accessibility, see the IBM Accessibility Center (www.ibm.com/able).

HTML documentation has accessibility features. PDF documents are supplemental and, as such, include no added accessibility features.

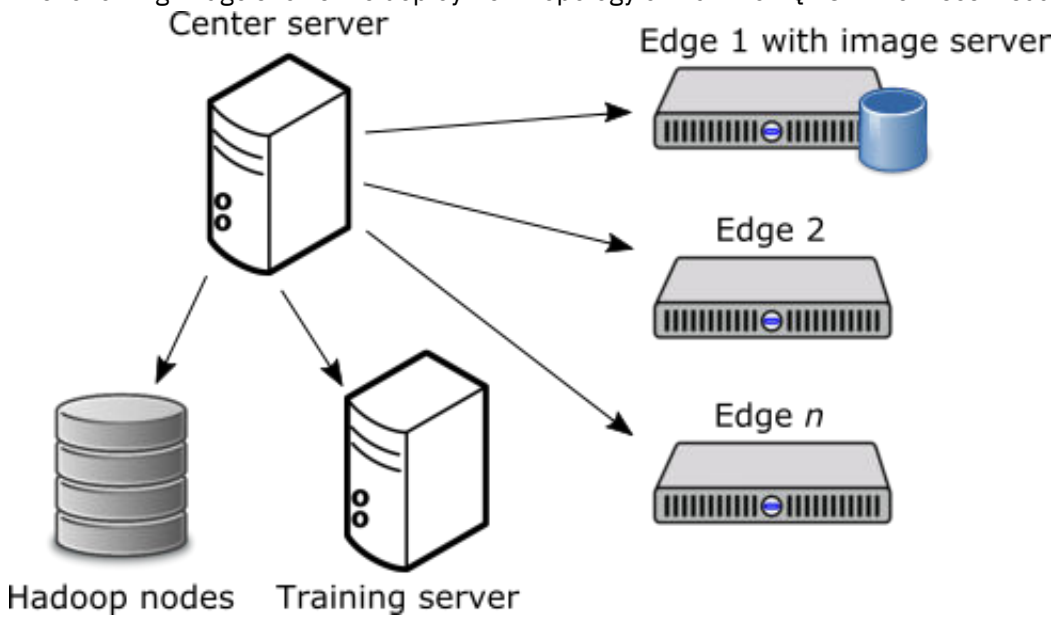
Chapter 2. Installing the product

Before using Maximo PQI On-Premises Visual Insights, you must install the server and artifacts. Follow these instructions to install the product or to upgrade an existing installation to the latest version.

Maximo PQI On-Premises Visual Insights deployment topology

The topology of Maximo PQI On-Premises Visual Insights consists of the center server, Hadoop nodes, training server, and edges.

The following image shows the deployment topology of Maximo PQI On-Premises Visual Insights.



Hadoop nodes

The Maximo PQI On-Premises Visual Insights node accesses the Hortonworks Data Platform nodes to read and write data.

Ambari 2.6.2.2 and Hortonworks Data Platform 2.6.5 are supported by Maximo PQI On-Premises Visual Insights. Hdfs, Hbase, and Zookeeper are the components for using Maximo PQI On-Premises Visual Insights.

If Kerberos is enabled on the Ambari server, specify the principal name without the cluster name. For example, use hdfs as the HDF principal name instead of hdfs-*cluster_name*, or use hbase as the Hbase principal name instead of hbase-*cluster_name*.

For more information about the hardware requirements for the Apache Hadoop, go to the Hortonworks Data Platform Hardware Recommendation website.

For more information about how to install the Hortonworks Data Platform 2.6.5 cluster, go to the Hortonworks Data Platform Documentation website.

Center server

The center server is an application server. The Maximo PQI On-Premises Visual Insights center and the provisioning console run on the server. Ensure that your system that you use for the center server meets the requirements.

- 4-core processor
- 32GB memory

- 1TB hard disk
- Red Hat Enterprise Linux Server Edition version 7.5, x86_64.
- Red Hat Enterprise Linux Server Edition version 7.5, ppc64le.

Training server

The training server is used to train models. Before using Maximo PQI On-Premises Visual Insights, you must create the training server. Before creating the training server, ensure that your system meets the requirements.

- Ubuntu 16.04
- Red Hat Enterprise Linux Server Edition version 7.5, x86_64.
- Red Hat Enterprise Linux Server Edition version 7.5, ppc64le.
- 4-core processor
- 64GB memory
- 2TB hard disk drive
- One or more NVIDIA GPU cards

For more information about the training server, go to the Creating the training server topic in the IBM Knowledge Center.

Edge Server

The edge server is used for scoring images. You must select one edge server to store images.

Ensure that your edge system meets the system requirements.

- One of the following operating systems:
  - Ubuntu 16.04 on x86_64
  - Red Hat Enterprise Linux 7.5 on x86_64
  - Red Hat Enterprise Linux 7.5 on IBM Power System
- CPU architecture: x86_64 or IBM Power System
- 4-core processor
- 64GB memory
- 2TB hard disk drive
- One or more NVIDIA GPU cards

System requirements

Before installing Maximo PQI On-Premises Visual Insights, ensure that the target system meets the system requirements.

The computer where you run the installation must have sufficient disk space that is allotted to the / directory. The minimum disk space for the / directory is 100GB.

The other system requirements for the installation server are as follows:

- 4-core processor
- 32GB memory
- Red Hat Enterprise Linux Server Edition version 7.5, x86_64.
- Red Hat Enterprise Linux Server Edition version 7.5, ppc64le.

Preparing for installation

Complete the following steps before you install Maximo PQI On-Premises Visual Insights.

Procedure

1. On the system on which you will install Maximo PQI On-Premises Visual Insights, open the `/etc/hosts` file.
2. Ensure that your Hortonworks Data Platform nodes are listed in the file, for example:
```
127.0.0.1 localhost.localdomain localhost
##.##.##.## hdpmgmt01.domain.com hdpmgmt01
##.##.##.## hdpmgmt02.domain.com hdpmgmt02
##.##.##.## hdpslave01.domain.com hdpslave01
##.##.##.## hdpslave02.domain.com hdpslave02
##.##.##.## hdpslave03.domain.com hdpslave03
```
3. Save and close the file.
4. Access the Ambari web user interface from a web browser by using the server name (the fully qualified domain name) on which you installed the Ambari server, and port 8080. For example, enter the following string in your browser:
HTTP://*node1.example.com*:8080
5. Select **Add New Hosts** and go to the **Install Options** page. In **Target Hosts**, list the node where you will install Maximo PQI On-Premises Visual Insights.
6. In **Host Registration Information**, select one of the following options:
   - Provide your SSH Private Key to automatically register the host:

     Click **SSH Private Key**. The private key file is `/root/.ssh/id_rsa`, if the root user installed the Ambari server. If you installed as a non-root user, then the default private key is in the `.ssh` directory in the non-root user's home directory.

     Click **Register and Confirm**.
   - Register the host manually and do not use SSH.
7. Verify that the hdfs and hbase services are available on the application node.
   a) If Kerberos is enabled, run the following command:
   ```
   kinit –kt /etc/security/keytabs/hdfs.headless.keytab hdfs
   ```
   b) Run the following command:
   ```
   hdfs dfs -ls /
   ```
   The result should list the directories under the root `hdfs` directory.
   c) If Kerberos is enabled, run the following command:
   ```
   kinit –kt /etc/security/keytabs/hbase.headless.keytab hbase
   ```
   d) Run the following commands:
   ```
   hbase shell
   list
   ```
   The result should not contain error messages. There are no tables inititally.
8. On the image server system:
   a) Create the `/imageserver` directory by using the following command:
   ```
   mkdir /imageserver
   ```
   Make sure the SSH user that is used to register the edge has read and write permission to this folder. You can run this command by using that SSH user.
   b) Install the NFS service by using the following command.
   In Ubuntu:
   ```
   apt-get install nfs-server
   ```
   In Redhat:
   ```
   yum install nfs-utils
   ```

c) Edit the `/etc/exports` file by using the following command:
`vi /etc/exports`
Add the following line to the file:
`/imageserver ##.##.##.##/24(rw,sync,no_root_squash,no_all_squash)`
where ##.##.##.## is the IP address of the center server.

d) Restart the NFS server by using the following command:
`service nfs-server restart`

9. On the training server system:

a) Install the NFS service by using the following command.
In Ubuntu:
`apt-get install nfs-server`
In Redhat:
`yum install nfs-utils`

b) Edit the `/etc/hosts` file:
`vi /etc/hosts`
Add the application node to the file:
##.##.##.## viappnode1.domain`.com` *viappnode1*
where *viappnode1* is the hostname of the center server.

c) Create the specified directories and then download pretrained models:

```
mkdir /home/pmqopsadmin/monitorfolder
mkdir /home/pmqopsadmin/monitorfolder/models
mkdir /home/pmqopsadmin/monitorfolder/models/weights
wget -P /home/pmqopsadmin/monitorfolder/models/weights/ http://
dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel
wget -P /home/pmqopsadmin/monitorfolder/models/weights/ http://
dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel
wget -O /home/pmqopsadmin/monitorfolder/models/weights/SSD.caffemodel
http://cs.unc.edu/~wliu/projects/ParseNet/
VGG_ILSVRC_16_layers_fc_reduced.caffemodel
wget https://dl.dropbox.com/s/gstw7122padlf0l/imagenet_models.tgz?dl=0 -
O /home/pmqopsadmin/monitorfolder/models/weights/imagenet_models.tgz
tar zxvf /home/pmqopsadmin/monitorfolder/models/weights/
imagenet_models.tgz -C /home/pmqopsadmin/monitorfolder/models/weights/
rm -f /home/pmqopsadmin/monitorfolder/models/weights/imagenet_models.tgz
```

Configuring passwordless authentication

Before you install Maximo PQI On-Premises Visual Insights, you must configure passwordless SSH authentication from the center server to all Hortonworks Data Platform nodes in your environment and to the training server.

Procedure

1. Log in as `root` to the server that hosts Maximo PQI On-Premises Visual Insights.
2. Edit the `/etc/hosts` file by using the following command:
`vi /etc/hosts`
Add the following line to the file:
ip_address ambari_host_name `ambari`
For example, `9.123.19.23 hdp.ibm.com ambari`
3. Generate an SSH authentication key pair:
`ssh-keygen -t rsa -b 2048`
4. Copy the public key to an Hortonworks Data Platform node and edit the `/etc/sudoers` file for that node. The following steps use the `ambari` node as an example:

a) Run the following command to copy the public key to the node:

```
ssh-copy-id -i ~/.ssh/id_rsa.pub root@ambari
```

b) Run the following command to log in without using strict host key checking:
```
ssh -o "StrictHostKeyChecking no" root@ambari
```

c) Launch vi with superuser privileges. If you cannot use `visudo`, use `sudo`.
```
visudo su
```

d) Change the permissions of the `/etc/sudoers` file:
```
chmod 640 /etc/sudoers
```

e) Edit the `/etc/sudoers` file:
```
vi /etc/sudoers
```

f) Find the following line in the file:
```
Defaults requiretty
```
Change this line to:
```
#Defaults requiretty
```

g) Quit vi.

h) Change the permissions of the `/etc/sudoers` file:
```
chmod 440 /etc/sudoers
```

Repeat these steps to copy the public key to all Hortonworks Data Platform nodes, the training server, and the server where you set up the installer. Repeat these steps on your training server to copy the public key for your training server to all Hortonworks Data Platform nodes, the server that hosts Maximo PQI On-Premises Visual Insights, and the server where you set up the installer..

Creating the training server

The training server is used to train models. Before using Maximo PQI On-Premises Visual Insights, you must create the training server.

The training server uses the Caffe deep-learning framework. Caffe is a dedicated artificial neural network (ANN) training environment. Deep learning requires significant processing resources. Deep learning can be performed efficiently by using a graphics processing unit (GPU). Although most deep learning frameworks also support CPU processing, GPU processing provides reasonable performance for production environments.

Related concepts

Troubleshooting the Caffe installation
If an error message displays in the log when you begin the Caffe build and installation, you can take steps to try to resolve the problem.

Opening training server ports

Before using the training server, you must open the firewall ports that are used by the training server.

About this task

If uncomplicated firewall (UFW) is enabled and active on the training server, open UFW for network file system (NFS) service and open the following firewall ports on the training server:

- 22
- 5001
- 5005
- 5060
- 5061

Installing NVIDIA GPU packages for Ubuntu

Use this task to install NVIDIA GPU packages for Ubuntu systems. To enable GPU processing, you must install the required NVIDIA GPU packages.

Procedure

1. Download and install the drivers for your NVIDIA GPU. The NVIDIA driver list for Ubuntu is available at the following link: Binary Driver How to - Nvidia. The following command is an example:
   ```
   sudo apt-get install ubuntu-drivers-common
   sudo ubuntu-drivers devices
   sudo apt-get install nvidia-384
   ```
2. Use the following command to check if your NVIDIA driver installed correctly:
   ```
   sudo nvidia-smi
   ```
3. Download and install the NVIDIA CUDA toolkit and corresponding CUDNN library. CUDA 8.0, CUDA 9.0 and CUDA 10.0 are supported. The following command is an example for CUDA 8.0.
   ```
   wget
   http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/
   cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
   ```
4. Install the CUDA file on the target server using the following commands:
   ```
   sudo dpkg -i cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
   sudo apt-get update
   sudo apt-get install cuda
   ```
   As NVIDIA upgrades the toolkit, you may get a newer version of the toolkit by using this command. It is recommended that you install CUDA 8.0 since that is the tested version. If you find that a newer version has been installed you can downgrade to CUDA 8.0 by using the following commands:
   ```
   sudo apt-get remove cuda
   sudo apt-get install cuda-8-0
   sudo ln -s /usr/local/cuda-8.0 /usr/local/cuda
   ```
5. Download the NVIDIA CUDA Deep Neural Network library `cudnn-8.0-linux-x64-v6.0.tgz` from the following link: https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v6/prod/8.0_20170307/cudnn-8.0-linux-x64-v6.0-tgz. You may need to create an account and log in before downloading the file.
6. Unpack the `cudnn-8.0-linux-x64-v6.0.tgz` file to the cuda installation directory using the following command:
   ```
   sudo tar -xvf cudnn-8.0-linux-x64-v6.0.tgz -C /usr/local
   ```
7. Set the environment variable using the following commands:
   ```
   export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
   export PATH=/usr/local/cuda/bin:$PATH
   ```
   Also add these commands to the `~/.bashrc` script.
8. Install the NVIDIA NCCL package using the following commands:
   ```
   git clone https://github.com/NVIDIA/nccl.git
   cd nccl
   sudo make install -j4
   ```

Installing NVIDIA GPU packages for Linux on NVIDIA Tesla K80 Power Systems Servers

Use this task to install NVIDIA GPU packages for Linux on NVIDIA Tesla K80 Power Systems Servers. To enable GPU processing, you must install the required NVIDIA GPU packages.

About this task

For Power8 systems, use CUDA 8 as described in the following task. For Power9 systems, substitute CUDA 10 for CUDA 8 in the following task.

Procedure

1. Download and install the drivers for your NVIDIA GPU. The NVIDIA driver list for Linux on Power Systems Servers is available at the following link: NVIDIA Driver Downloads. Follow the installation instructions on the download page.
2. Download the CUDA repository file and install CUDA 8 by using the following commands:
   ```
   wget https://developer.download.nvidia.com/compute/cuda/repos/rhel7/ppc64le/cuda-repo-rhel7-8.0.61-1.ppc64le.rpm
   rpm -i cuda-repo-rhel7-8.0.61-1.ppc64le.rpm
   yum clean all
   yum install cuda
   ```
3. For CUDA 8, download the NVIDIA CUDA Deep Neural Network library `cudnn-8.0-linux-ppc64le-v6.0-tgz` from the following URL: https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v6/prod/8.0_20170307/cudnn-8.0-linux-ppc64le-v6.0-tgz. For CUDA 10, download the NVIDIA CUDA Deep Neural Network library `cudnn-10.0-linux-ppc64le-v7.3.1.20.tgz` from the following URL: https://developer.download.nvidia.com/compute/cuda/repos/rhel7/ppc64le/cuda-10.0.130-1.ppc64le.rpm. You might need to create an account and log in before downloading the file.
4. Unpack the `cudnn-8.0-linux-ppc64le-v6.0.tgz` file to the cuda installation directory by using the following command:
   ```
   sudo tar -xvf cudnn-8.0-linux-ppc64le-v6.0.tgz -C /usr/local
   ```
5. Set the environment variable by using the following command:
   ```
   export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
   ```
   Also add this command to the ~/.bashrc script.
6. Install the NVIDIA NCCL package using the following commands:
   ```
   git clone https://github.com/NVIDIA/nccl.git
   cd nccl
   sudo make install -j4
   ```

Installing NVIDIA GPU packages for Linux on NVIDIA Tesla V100 Power Systems Servers

Use this task to install NVIDIA GPU packages for Linux on NVIDIA Tesla V100 Power Systems Servers. To enable GPU processing, you must install the required NVIDIA GPU packages.

Procedure

1. Download and install NVIDIA CUDA 9.2.148 from developer.nvidia.com/cuda-92-download-archive.
   a) Select Operating System: Linux.
   b) Select Architecture: ppc64le.
   c) Select Distribution: RHEL.
   d) Select Version: 7.
   e) Select Installer Type: rpm (local). The local rpm is preferred over the network rpm as it ensures the version that is installed is the version that is downloaded. With the network rpm, the `yum install cuda` command always installs the latest version of the CUDA Toolkit.
   f) Click **Download** to download the base installer.
   g) Click **Download** to download patch 1.
   h) Follow the Linux on POWER installation instructions in the CUDA Quick Start Guide, including the steps that describe how to set up the CUDA development environment by updating PATH and `LD_LIBRARY_PATH`.
2. Download NVIDIA driver 410.104 from http://www.nvidia.com/Download/index.aspx.
   a) Select Product Type: Tesla.
   b) Select Product Series: V-Series.
   c) Select Product: Tesla V100.

d) Select Operating System: Linux POWER LE RHEL 7.

e) Select CUDA Toolkit:10.0.

f) Click **Search** to go to the download link and then click **Download**.

3. **Note:** For IBM Power® System AC922 systems, operating system and system firmware updates are required before you install the latest GPU driver.

   Install CUDA and the GPU driver:

   a) Install the CUDA Base repository rpm.

   b) Install the CUDA Patch 1 repository rpm.

   c) Install the GPU driver repository rpm.

   d) Run the following command to install CUDA, patch, and GPU driver:
   ```
   sudo yum install cuda
   ```

   e) Restart the system to activate the driver.

4. Enable NVIDIA system-persistenced services by using the following shell command:
   ```
   systemctl enable nvidia-persistenced
   ```
5. Check NVIDIA drivers by using the following shell command:
   ```
   nvidia-smi
   ```
6. Download NVIDIA cuDNN v7.4.2 for CUDA 10.0 (cuDNN v7.4.2 Library for Linux (Power8/Power9)) from developer.nvidia.com/cudnn. Registration in the NVIDIA Accelerated Computing Developer Program is required.
7. Download NVIDIA NCCL v2.3.7 for CUDA 10.0 (NCCL 2.3.7 O/S agnostic and CUDA 10.0 and IBM Power) from developer.nvidia.com/nccl Registration in the NVIDIA Accelerated Computing Developer Program is required.
8. Install the cuDNN v7.4.2 and NCCL v2.3.7 packages and then refresh the shared library cache by using the following commands:
   ```
   sudo tar -C /usr/local --no-same-owner -xzvf cudnn-9.2-linux-ppc64le-
   v7.4.2.tgz
   sudo tar -C /usr/local --no-same-owner -xzvf nccl_2.3.7+cuda10.0_ppc64le.tgz
   sudo ldconfig
   ```

Installing Caffe for Ubuntu

You must install the Caffe deep-learning framework and related packages. Caffe is used for model training and defect classification.

Procedure

1. Install the packages that are required for Caffe by using the following commands:
   ```
   sudo apt-get update
   sudo apt-get upgrade
   sudo apt-get install -y build-essential cmake git pkg-config
   sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev
   libhdf5-serial-dev protobuf-compiler
   sudo apt-get install -y libatlas-base-dev libjasper-dev
   sudo apt-get install -y --no-install-recommends libboost-all-dev
   sudo apt-get install -y libgflags-dev libgoogle-glog-dev liblmdb-dev
   sudo apt-get install -y python-pip
   sudo apt-get install -y python-dev
   sudo apt-get install -y python-numpy python-scipy
   sudo apt-get install -y libopencv-dev
   sudo pip install opencv-python
   sudo pip install flask_httpauth
   sudo pip install gevent
   sudo pip install pyinotify
   sudo pip install tornado
   ```

2. Download the Caffe source code by using the following command:
   `wget https://github.com/BVLC/caffe/archive/1.0.zip`
3. Unpack the package and enter the package directory by using the following commands:
   ```
   unzip 1.0.zip
   cd ./caffe-1.0
   ```
4. Make a copy of the make configuration file by using the following command:
   `cp Makefile.config.example Makefile.config`
5. Add the following variables in the `Makefile.config` file:
   ```
   USE_CUDNN := 1
   CUDA_DIR := /usr/local/cuda
   PYTHON_INCLUDE := /usr/include/python2.7 \
   /usr/lib/python2.7/dist-packages/numpy/core/include
   PYTHON_LIB := /usr/lib/x86_64-linux-gnu
   WITH_PYTHON_LAYER := 1
   INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include \
   /usr/include/hdf5/serial
   LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib \
   /usr/lib/x86_64-linux-gnu /usr/lib/x86_64-linux-gnu/hdf5/serial
   ```
6. In the `caffe-1.0` directory, run the following command:
   ```
   find . -type f -exec sed -i -e 's^"hdf5.h"^"hdf5/serial/hdf5.h"^g' -e
   's^"hdf5_hl.h"^"hdf5/serial/hdf5_hl.h"^g' '{}' \;
   ```
7. Run the following commands:
   ```
   cd /usr/lib/x86_64-linux-gnu
   sudo ln -s libhdf5_serial.so.10.1.0 libhdf5.so
   sudo ln -s libhdf5_serial_hl.so.10.0.2 libhdf5_hl.so
   ```
8. Install the required Python packages in the `caffe-1.0/python` directory by using the following commands:
   ```
   cd {caffe-installation-path}/caffe-1.0/python
   for req in $(cat requirements.txt); do sudo -H pip install $req --upgrade;
   done
   ```
   where `{caffe-installation-path}` is the Caffe deployment path.
9. Open the makefile in the `{caffe-installation-path}` directory and change the parameter NVCCFLAGS to the following setting:
   ```
   NVCCFLAGS += -D_FORCE_INLINES -ccbin=$(CXX) -Xcompiler -fPIC $
   (COMMON_FLAGS)
   ```
10. In the main Caffe directory `caffe-1.0`, begin the Caffe build and installation by using the following commands:
    ```
    make all
    make test
    make runtest
    make pycaffe
    make distribute
    ```
11. Add the following line to the `~/.bashrc` script:
    ```
    export PYTHONPATH="/usr/lib/python2.7:{caffe-installation-path}/caffe-1.0/
    python:$PYTHONPATH"
    ```
    where `{caffe-installation-path}` is the Caffe deployment path.

Installing Caffe for Linux on Power Systems Servers

Use this task to install Caffe for Linux on Power Systems Servers systems. You must install the Caffe deep-learning framework and related packages. Caffe is used for model training and defect classification.

Procedure

1. Install the packages that are required for Caffe by using the following commands:
   ```
   sudo yum clean all
   ```

```
sudo yum update
sudo yum install upgrade
sudo yum install -y libboost-*
sudo yum install -y gflags-devel glog-devel lmdb-devel
sudo yum install -y python-pip
sudo yum install -y python-devel
sudo yum install -y opencv-devel
sudo yum makecache
sudo yum install -y protobuf-devel leveldb-devel lmdb-devel snappy-devel
opencv-devel boost-devel hdf5-devel atlas-devel glog-devel gflags-devel
sudo yum install libpng-devel
sudo yum install freetype-devel
sudo yum install libjpeg-turbo-devel
sudo yum install opencv-python
sudo rpm -e --nodeps numpy
sudo pip install numpy
pip install --upgrade pip
sudo pip install flask_httpauth
sudo pip install gevent
sudo pip install pyinotify
ln -s /usr/local/cuda-10.0 /usr/local/cuda
pip install scikit-image
sudo pip install tornado
```

2. Link the Atlas library by using the following commands:
```
ln -fs /usr/lib64/atlas/libsatlas.so /usr/lib64/libatlas.so
ln -fs /usr/lib64/atlas/libsatlas.so /usr/lib64/libcblas.so
```
3. Download the Caffe source code by using the following command:
```
wget https://github.com/BVLC/caffe/archive/1.0.zip
```
4. Unpack the package and enter the package directory by using the following commands:
```
unzip 1.0.zip
cd ./caffe-1.0
```
5. Replace the following Caffe files:
   - In the `/include/caffe/util/cudnn.hpp` directory, replace the `cudnn.hpp` file with the newest `cudnn.hpp` file that is in the Caffe repository on GitHub: github.com/BVLC/caffe.git
   - Replace all of the cudnn files that are in the `/src/caffe/layers` folder with the newest cudnn files that are in the Caffe repository on GitHub: github.com/BVLC/caffe.git

   For example, run the following commands:
```
cp -rf /root/source/caffe-git/caffe-master/include/caffe/util/
cudnn.hpp /usr/local/caffe-1.0/include/caffe/util/
cp -rf /root/source/caffe-git/caffe-master/src/caffe/layers/cudnn_* /usr/
local/caffe-1.0/src/caffe/layers/
cp -rf /root/source/caffe-git/caffe-master/include/caffe/layers/
cudnn_* /usr/local/p/usr/local/caffe-1.0/include/caffe/layers/
```
6. Make a copy of the make configuration file by using the following command:
```
cp Makefile.config.example Makefile.config
```
7. Add the following variables in the Makefile.config file:

```
USE_CUDNN := 1
CUDA_DIR := /usr/local/cuda
PYTHON_INCLUDE := /usr/include/python2.7 \
     /usr/lib64/python2.7/site-packages/numpy/core/include/
PYTHON_LIB := /usr/lib/gcc/ppc64le-redhat-linux/4.8.5/
WITH_PYTHON_LAYER := 1
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include
/usr/local/cuda-10.0/targets/ppc64le-linux/include/
LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib \
    /usr/lib64 /usr/local/lib64
```

Change CUDA_ARCH to the following text:

```
CUDA_ARCH := -gencode arch=compute_30,code=sm_30
-gencode arch=compute_35,code=sm_35
-gencode arch=compute_50,code=sm_50
-gencode arch=compute_52,code=sm_52
-gencode arch=compute_60,code=sm_60
-gencode arch=compute_61,code=sm_61
-gencode arch=compute_61,code=compute_61
```

8. Install the required Python packages in the caffe-1.0/python directory by using the following commands:
   `cd` *`caffe-installation-path`*`/caffe-1.0/python`
   `for req in $(cat requirements.txt); do sudo -H pip install $req --upgrade; done`
   where *caffe-installation-path* is the Caffe deployment path.
9. Open the Makefile in the *caffe-installation-path* directory and change the parameter NVCCFLAGS to the following setting:
   `NVCCFLAGS += -D_FORCE_INLINES -ccbin=$(CXX) -Xcompiler -fPIC $(COMMON_FLAGS)`
10. In the main Caffe directory `caffe-1.0`, begin the Caffe build and installation by using the following commands:
    `make all`
    `make test`
    `make runtest`
    `make pycaffe`
    `make distribute`
11. Add the following line to the ~/.bashrc script:
    `export PYTHONPATH="/usr/lib/python2.7:`*`caffe-installation-path`*`/caffe-1.0/python:$PYTHONPATH"`
    where *caffe-installation-path* is the Caffe deployment path.
12. Run the following post-installation tests:
    a) `make runtest | tee -a runtest.out`
    b) `grep -i OK runtest.out | wc –l`
       Caffe test output must be 2101
    c) `python -c "import caffe"`
       to test the Pycaffe installation
    d) `tail -n 2 runtest.out`
       The contents of `runtest.out` should contain the following text:
       [==========] 2101 tests from 277 test cases ran. (291548 ms total)
       [PASSED] 2101 tests

Troubleshooting the Caffe installation

If an error message displays in the log when you begin the Caffe build and installation, you can take steps to try to resolve the problem.

Symptoms 1

When you began the Caffe build and installation, the following message displays:

```
    1.    In file included from ./include/caffe/util/device_alternate.hpp:40:0,
    2.                     from ./include/caffe/common.hpp:19,
    3.                     from src/caffe/common.cpp:7:
    4.    ./include/caffe/util/cudnn.hpp: In function ‘void
caffe::cudnn::createPoolingDesc(cudnnPoolingStruct**,
        caffe::PoolingParameter_PoolMethod, cudnnPoolingMode_t*, int, int, int, int, int,
int)’:
    5.    ./include/caffe/util/cudnn.hpp:127:41: error: too few arguments to function
‘cudnnStatus_t
        cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t, cudnnPoolingMode_t,
```

```
cudnnNanPropagation_t, int,
        int, int, int, int, int)'
    6.             pad_h, pad_w, stride_h, stride_w));
    7.                                               ^
    8.    ./include/caffe/util/cudnn.hpp:15:28: note: in definition of macro ‘CUDNN_CHECK’
    9.         cudnnStatus_t status = condition; \
    10.                                ^
    11.    In file included from ./include/caffe/util/cudnn.hpp:5:0,
    12.                     from ./include/caffe/util/device_alternate.hpp:40,
    13.                     from ./include/caffe/common.hpp:19,
    14.                     from src/caffe/common.cpp:7:
    15.    /usr/local/cuda-7.5//include/cudnn.h:803:27: note: declared here
    16.     cudnnStatus_t CUDNNWINAPI cudnnSetPooling2dDescriptor(
    17.                               ^
    18.    make: *** [.build_release/src/caffe/common.o] Error 1
    19.
```

Resolving the problem 1

To fix the error, refer to the following steps:

1. In the `/include/caffe/util/cudnn.hpp` directory, replace the `cudnn.hpp` file with the newest `cudnn.hpp` file that is in the Caffe repository on GitHub.
2. In the `/src/caffe/layers` folder, replace all of the `cudnn` files that are in the `/src/caffe/layers` folder with the newest `cudnn` files that are in the Caffe repository on GitHub.

Symptoms 2

When you install the required Python packages in the `caffe-1.0/python` directory, the following message displays:

```
Traceback (most recent call last):
  File "/usr/bin/pip", line 11, in <module>
    sys.exit(main())
  File "/usr/lib/python2.7/dist-packages/pip/__init__.py", line 215, in main
    locale.setlocale(locale.LC_ALL, '')
  File "/usr/lib/python2.7/locale.py", line 581, in setlocale
    return _setlocale(category, locale)
locale.Error: unsupported locale setting
Traceback (most recent call last):
  File "/usr/bin/pip", line 11, in <module>
    sys.exit(main())
  File "/usr/lib/python2.7/dist-packages/pip/__init__.py", line 215, in main
    locale.setlocale(locale.LC_ALL, '')
  File "/usr/lib/python2.7/locale.py", line 581, in setlocale
    return _setlocale(category, locale)
locale.Error: unsupported locale setting
```

Resolving the problem 2

To resolve this error, run the following command:
`export LC_ALL=C`

Symptoms 3

When you begin the Caffe build and installation, the following message displays:

```
nvcc fatal   : Unsupported gpu architecture 'compute_20'
Makefile:595: recipe for target '.build_release/cuda/src/caffe/layers/prelu_layer.o' failed
make: *** [.build_release/cuda/src/caffe/layers/prelu_layer.o] Error 1
```

Resolving the problem 3

Comment out `-gencode arch=compute_20` in `Makefile.config`.

Symptoms 4

When you begin the Caffe build and installation, the following message displays:

```
PROTOC src/caffe/proto/caffe.proto
make: protoc: Command not found
Makefile:639: recipe for target '.build_release/src/caffe/proto/caffe.pb.cc' failed
make: *** [.build_release/src/caffe/proto/caffe.pb.cc] Error 127
```

Resolving the problem 4

Run the following command to install the protoc program:

sudo apt install protobuf-compiler

Symptoms 5

When you begin the Caffe build and installation, the following message displays:

```
src/caffe/layers/hdf5_data_layer.cpp:13:30: fatal error: hdf5/serial/hdf5.h: No such file or
directory
compilation terminated.
Makefile:582: recipe for target '.build_release/src/caffe/layers/hdf5_data_layer.o' failed
make: *** [.build_release/src/caffe/layers/hdf5_data_layer.o] Error 1
```

Resolving the problem 5

You might need to install a Caffe dependency package using the following commands:

sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler
sudo apt-get install --no-install-recommends libboost-all-dev
sudo apt-get install libopenblas-dev liblapack-dev libatlas-base-dev
sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev

Installing object detection libraries on the training server

After you install the artifacts, install an object detection library on the training server. The object detection library enables you to validate models on the training server.

About this task

IBM Maximo PQI On-Premises Visual Insights supports the following object detection libraries: you only look once (YOLO), Faster R-CNN, and Single Shot MultiBox Detector (SSD).

Procedure

1. Install the related Python packages by using the following commands:
   sudo apt-get install python-numpy
   sudo apt-get install python-scipy
   sudo pip install cython
   sudo pip install easydict
   sudo pip install uuid
   sudo pip install multiprocessing
2. Install all of the following libraries:

Library	Installation Instructions
YOLO version 2 library	a. Run the following commands to get the YOLO source code: **git clone --recursive https://github.com/pjreddie/darknet.git** **cd darknet**

Library	Installation Instructions
	git checkout 691debd b. Edit the Makefile file by enabling the **GPU**, and select the correct **GPU ARCH** parameter according to your machine configuration: **vi Makefile** **GPU=1** c. Run the following command to compile YOLO: **make**

Faster-RCNN Python library	a. Run the following command to get the Faster R-CNN source code:
	git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git
	b. In the py-faster-rcnn directory, in the lib folder, run the following command to Compile Cython:
	make
	c. Go to the caffe-fast-rcnn directory under the py-faster-rcnn directory and make a copy of the make configuration file by using the following commands:
	cd caffe-fast-rcnn
	cp Makefile.config.example Makefile.config
	d. Add the following variables to the Makefile.config file:
	`USE_CUDNN := 1`
	`CUDA_DIR := /usr/local/cuda`
	`PYTHON_INCLUDE := /usr/include/python2.7`
	`\`
	`/usr/lib64/python2.7/site`
	`-`
	`packages/numpy/core/include/`
	`PYTHON_LIB:=/usr/lib64/`
	`WITH_PYTHON_LAYER := 1`
	`INCLUDE_DIRS := $(PYTHON_INCLUDE) /`
	`usr/local/include`
	`LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib`
	`\`
	`/usr/lib64 /usr/local/lib64`
	e. Run the following command to compile Caffe:
	make
	f. Run the following command to compile pycaffe by using the Python layer:
	make pycaffe

SSD library	a. Run the following command to get the SSD source code:
	git clone --recursive https://github.com/weiliu89/caffe.git ~/ssd-caffe
	cd ~/ssd-caffe
	git checkout ssd
	b. Make a copy of the Makefile configuration file by using the following command:
	cp Makefile.config.example Makefile.config
	c. Compile the default OpenBLAS by using the following commands:
	`git clone git://github.com/xianyi/OpenBLAS.git`
	`cd OpenBLAS`
	`make TARGET=POWER9`
	`make install`

Library	Installation Instructions

	d. Edit the `Makefile.config` file as follows:
	`CUDA_ARCH := -gencode arch=compute_30,code=sm_30`
	`-gencode arch=compute_35,code=sm_35`
	`-gencode arch=compute_50,code=sm_50`
	`-gencode arch=compute_52,code=sm_52`
	`-gencode arch=compute_61,code=sm_61`
	`BLAS_INCLUDE := /opt/OpenBLAS/include`
	`BLAS_LIB := /opt/OpenBLAS/lib`
	`PYTHON_INCLUDE := /usr/include/python2.7 \`
	`/usr/lib64/python2.7/site-packages/numpy/core/include/`
	e. Run the following command on one line:
	`find . -type f -exec sed -i -e 's^"hdf5.h"^"hdf5/serial/hdf5.h"^g' -e 's^"hdf5_hl.h"^"hdf5/serial/hdf5_hl.h"^g' '{}' \;`
	f. Compile the code by using following command:
	`make -j8`
	g. Compile the Python layer by using the following command:
	`make py`
	h. Compile the test by using the following command:
	`make test -j8`

3. Add the following environment variables in the `/root/.bashrc` file: *YOLO_HOME*, *FRCNN_HOME*, and *SSD_HOME*. The following text is an example of adding environment variables: `YOLO_HOME=~/darknet/ FRCNN_HOME=~/py-faster-rcnn/ SSD_HOME=~/ssd-caffe/`.

Related concepts

Troubleshooting the Faster-RCNN Python library installation

If an error message displays when you install the Faster-RCNN Python library, you can take steps to try to resolve the problem.

Troubleshooting the Faster-RCNN Python library installation

If an error message displays when you install the Faster-RCNN Python library, you can take steps to try to resolve the problem.

Symptoms

When you install the Faster-RCNN Python library, the following cudnn compile error displays:

```
CXX src/caffe/layers/hdf5_data_layer.cpp
./include/caffe/util/cudnn.hpp: In function 'const char* cudnnGetErrorString(cudnnStatus_t)':
./include/caffe/util/cudnn.hpp:21:10: warning: enumeration value
'CUDNN_STATUS_RUNTIME_PREREQUISITE_MISSING' not handled in switch
./include/caffe/util/cudnn.hpp:15:28: note: in definition of macro 'CUDNN_CHECK'
     cudnnStatus_t status = condition;
Makefile:563: recipe for target '.build_release/src/caffe/layers/hdf5_data_layer.o' failed
make: *** [.build_release/src/caffe/layers/hdf5_data_layer.o] Error 1
```

On Power9 systems with CUDA 10, the following cudnn compile error displays:

```
./include/caffe/util/cudnn.hpp:127:41: error: too few arguments to function
'cudnnStatus_t cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t, cudnnPoolingMode_t,
cudnnNanPropagation_t, int, int, int, int, int, int)' pad_h, pad_w, stride_h, stride_w));
^./include/caffe/util/cudnn.hpp:15:28: note: in definition of macro 'CUDNN_CHECK'
cudnnStatus_t status = condition; \ ^In file included from ./include/caffe/util/
cudnn.hpp:5:0, from ./include/caffe/util/device_alternate.hpp:40, from ./include/
caffe/common.hpp:19, from ./include/caffe/blob.hpp:8, from src/caffe/blob.cpp:4:/
usr/local/cuda/include/cudnn.h:991:1: note: declared here cudnnSetPooling2dDescriptor
(cudnnPoolingDescriptor_t poolingDesc, ^
```

Resolving the problem

To fix the error, find your Caffe installation and run the following commands:
cp -rf ~/caffe-1.0/include/caffe/util/cudnn.hpp ~/py-faster-rcnn/caffe-fast-rcnn/include/caffe/util/
cp -rf ~/caffe-1.0/src/caffe/layers/cudnn_* ~/py-faster-rcnn/caffe-fast-rcnn/src/caffe/layers/
cp -rf ~/caffe-1.0/include/caffe/layers/cudnn_* ~/py-faster-rcnn/caffe-fast-rcnn/include/caffe/layers/

For Power9 systems with CUDA 10, edit the `Makefile.config` file by using the following command:
`vim ~/py-faster-rcnn/caffe-fast-rcnn/Makefile.config`
Change the CUDA_ARCH value as follows:

```
CUDA_ARCH := -gencode arch=compute_30,code=sm_30 \
  -gencode arch=compute_35,code=sm_35 \
  -gencode arch=compute_50,code=sm_50 \
  -gencode arch=compute_50,code=compute_50
```

Patching Faster R-CNN

Before using the product you must edit the Faster R-CNN files.

About this task

Procedure

1. Open imdb.py by using vi:
   `vi FRCNN_HOME/lib/datasets/imdb.py`
2. Under the function `append_flipped_images()` and before the code `assert (boxes[:, 2] >= boxes[:, 0]).all()`, add the following codes:

   ```
   for b in range(len(boxes)):
     if boxes[b][2]< boxes[b][0]:
       boxes[b][0] = 0
   ```
3. Open pascal_voc.py by using vi:
   `vi FRCNN_HOME/lib/datasets/pascal_voc.py`
4. Under the function `_load_pascal_annotation(,)`, change the following codes:

   ```
   x1 = float(bbox.find('xmin').text)-1
   y1 = float(bbox.find('ymin').text)-1
   x2 = float(bbox.find('xmax').text)-1
   y2 = float(bbox.find('ymax').text)-1
   ```

   to this:

   ```
   x1 = float(bbox.find('xmin').text)
   y1 = float(bbox.find('ymin').text)
   x2 = float(bbox.find('xmax').text)
   y2 = float(bbox.find('ymax').text)
   ```
5. Open train.py by using vi:
   `vi FRCNN_HOME/lib/fast_rcnn/train.py`
6. Add the following line of code:
   `import google.protobuf.text_format`
7. Open minibatch.py by using vi:
   `vi FRCNN_HOME/lib/roi_data_layer/minibatch.py`
8. In line 26, change
   `fg_rois_per_image = np.round(cfg.TRAIN.FG_FRACTION * rois_per_image)`
   to

```
fg_rois_per_image = np.round(cfg.TRAIN.FG_FRACTION *
rois_per_image).astype(np.int)
```

9. In line 173, change
   `start = 4 * cls`
   to
   `start = int(4 * cls)`

Installing the PostgreSQL database

The training server uses a PostgreSQL database to record the job execution status to support the training server cluster. You must install and configure PostgreSQL database on one training server.

Installing PostgreSQL for Ubuntu

Use this task to install PostgreSQL for Ubuntu systems. The training server and stand-alone edge use a PostgreSQL database.

Procedure

1. Install PostgreSQL, PHPPgadmin and Apache2 by using the following command:
   `sudo apt-get -y install postgresql postgresql-contrib phppgadmin`
2. Configure the PostgreSQL user.
   a) Log in as the PostgreSQL user by using the following commands:
   ```
   sudo su
   su - postgres
   psql
   ```
   b) Configure the password for user postgres by using the following commands:
   ```
   password postgres
   password
   \q
   ```
3. Configure Apache2 by editing the `nano phppgadmin.conf` file:
   ```
   cd /etc/apache2/conf-available/
   nano phppgadmin.conf
   ```
   Delete the following line: `Require local`. Add the following line to the file:
   `Require all granted`
4. Configure the PHPPgadmin by editing the `config.inc.php` file:
   ```
   cd /etc/phppgadmin/
   nano config.inc.php
   ```
   Find the following line in the file:
   `$conf['extra_login_security'] = true`
   Change `true` to `false`.
5. Restart PostreSQL and Apache2 by using the following commands:
   ```
   systemctl restart postgresql
   systemctl restart apache2
   ```
6. Verify that you can access the user interface on the stand-alone edge by accessing the following URL:
   `http://`*`standalone_edge_IP`*`/phppgadmin`
   where *standalone_edge_IP* is the IP address of the stand-alone edge.
7. Create the database schema in PostgreSQL.
   a) Run the following command on the SQL console on PHPPgadmin:
   ```
   create database edge with owner postgres encoding='UTF-8'
   lc_collate='en_US.utf8' lc_ctype='en_US.utf8' template template0;
   ```
   b) In the database, create the following tables:
   ```
   CREATE TABLE vi_tenant_inspectionresult(id text, info jsonb);
   CREATE TABLE vi_tenant_notification(id text, info jsonb);
   CREATE TABLE vi_tenant_defectsummary(id text, info jsonb);
   CREATE TABLE vi_tenant_uploaddataset(id text, info jsonb);
   ```

```
CREATE TABLE vi_tenant_syncprocess(id text, info jsonb);
CREATE TABLE vi_tenant_model(id text, info jsonb);
CREATE TABLE vi_tenant_datagroup(id text, info jsonb);
```
where *tenant* is the tenant for the operation user in the Maximo PQI On-Premises Visual Insights center. Get the tenant value from the user profile in the user interface for the center application.

Installing PostgreSQL for Linux on Power Systems Servers

Use this task to install PostgreSQL for Linux on Power Systems Servers. The training server and stand-alone edge use a PostgreSQL database.

Procedure

1. su to root by using the following command:
   ```
   sudo su
   ```
2. Download the source for PostgreSQL:
   ```
   wget https://ftp.postgresql.org/pub/source/v9.5.13/postgresql-9.5.13.tar.gz
   ```
3. Install PostgreSQL by using the following commands:
   ```
   tar -zxvf postgresql-9.5.13.tar.gz
   cd postgresql-9.5.13/
   yum -y install readline-devel
   ./configure --prefix=/usr/local/postgresql
   make
   make install
   ```
4. Create user postgres and change owner for the `postgres` directory:
   ```
   useradd postgres
   chown -R postgres:postgres /usr/local/postgresql/
   ```
5. Change to user postgres:
   ```
   su postgres
   ```
6. Configure the system path for postgres:
   ```
   vi ~/.bashrc
   PGHOME=/usr/local/postgresql
   export PGHOME
   PGDATA=/usr/local/postgresql/data
   export PGDATA
   PATH=$PATH:$HOME/.local/bin:$HOME/bin:$PGHOME/bin
   export PATH
   ```
7. Source the configuration:
   ```
   source ~/.bashrc
   ```
8. Initialize the PostgreSQL database:
   ```
   initdb
   ```
9. Configure the database. Open `postgresql.conf` in vi:
   ```
   vi /usr/local/postgresql/data/postgresql.conf
   ```
   Change:
   ```
   #listen_address='localhost'
   #port = 5432
   ```
   to:
   ```
   listen_address='*'
   port = 5432
   ```
   Open the `pg_hba.conf` file in vi:
   ```
   vi /usr/local/postgresql/data/pg_hba.conf
   ```
   Add the following line to the file:
   ```
   host all all 0.0.0.0/0 trust
   ```
10. Restart postgresql:

```
pg_ctl -D /usr/local/postgresql/data -l logfile restart
```

11. Change the password for user postgres in the PostgreSQL database:
    ```
    psql
    ALTER USER postgres WITH PASSWORD 'password';
    \q
    ```
    If the postgresql service is not started, run the following commands:
    ```
    su postgres
    vi ~/.bashrc
    ```
    Add `/usr/local/pgsql/bin/` to the file:
    ```
    export PATH=/usr/local/cuda-8.0/bin:$PATH:/usr/local/pgsql/bin/
    ```
    Run the following command:
    ```
    source ~/.bashrc
    ```
12. Create the database schema in PostgreSQL. Run the following command on the psql console:
    ```
    create database edge with owner postgres encoding='UTF-8'
    lc_collate='en_US.utf8' lc_ctype='en_US.utf8' template template0;
    ```
    In the database, create the following tables:

    ```
    CREATE TABLE vi_tenant_inspectionresult(id text, info jsonb);
    CREATE TABLE vi_tenant_notification(id text, info jsonb);
    CREATE TABLE vi_tenant_defectsummary(id text, info jsonb);
    CREATE TABLE vi_tenant_uploaddataset(id text, info jsonb);
    CREATE TABLE vi_tenant_syncprocess(id text, info jsonb);
    CREATE TABLE vi_tenant_model(id text, info jsonb);
    CREATE TABLE vi_tenant_datagroup(id text, info jsonb);
    ```

    where *tenant* is the tenant for the operation user in the Maximo PQI On-Premises Visual Insights center. You can get the tenant value from the user profile in the user interface for the center application.

Installing Python modules

To create a training server or stand-alone edge, you must install the required Python module.

Procedure

Install the Python module by using the following commands:

```
sudo apt-get update
sudo apt -y install postgresql
sudo apt -y install libpq-dev
sudo pip install PyGreSQL
sudo pip install DBUtils
```

Installing the server

Perform this task to install the server component.

Procedure

1. Download the server files to the system that you will use for the server. For Ubuntu, download `pqi_server_installer_1.4_l86-64_en.tar.gz` and `pqi_server_middleware_1.4_l86-64_en.tar.gz`. For Linux on Power Systems Servers, download `pqi_server_installer_1.4_ppc64_en.tar.gz` and `pqi_server_middleware_1.4_ppc64_en.tar.gz`. You can get these files from the `IBM Maximo PQI On-Premises Visual Insights V1.4 Linux Multilingual eAssembly` package.
2. Copy the `pqi_server_installer_1.4_l86-64_en.tar.gz` or `pqi_server_installer_1.4_ppc64_en.tar.gz` file to the installation directory (*VI_install_dir*) and decompress the file.

   The `ServerInstallation` directory is created.

3. Copy the `pqi_server_middleware_1.4_l86-64_en.tar.gz` or `pqi_server_middleware_1.4_ppc64_en.tar.gz` file to the *VI_install_dir*/`ServerInstallation/SolutionInstaller/NodeRoot/Downloads/Software` directory.
4. Run the following script to decompress the files:
   `sh PQIF_1.4_UncompressTarball.sh`
5. Edit `/etc/bashrc` by using the following command:
   `vi /etc/bashrc`
   Add the following lines to the file:

```
export JAVA_HOME=jdk_installation_path
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

6. To set up the installer, go to the installer setup folder *VI_install_dir*/`ServerInstallation/SolutionInstaller` and then run the following script:
   `./setup.sh`
7. Access the installer by going to the following URL:
   `https://`*hostname*`:8080/UI/index.html`
   where *hostname* is the host name for installer. Make sure the language setting for your browser is English (en). Review the license information and then click **Accept**.
8. Click **Predefined Configuration** to load a predefined configuration that includes one node, WebSphere® Application Server Liberty, Provisioning Console, IBM JDK Artifact, and Service Framework. Then click **OK** to confirm your choice.
9. Change the values in the **Property Editor** to the configuration of your server.
10. Click **Validate** to validate the configuration.
11. Click **Run** to run the installation.
12. In the installer setup folder, run the following script to clean up the installation files:
    `./cleanup.sh`
13. Restart the server.

Installing the artifacts

Perform this task to install the product artifacts.

Before you begin
Before installing the product artifacts, you must create the training server.

Procedure

1. Download the `ArtifactsInstallation.tar.gz` file to the system that you will use for the server. You can get `ArtifactsInstallation.tar.gz` from the `IBM Visual Insights Artifact Installer 1.4 Linux x86-64 English` package.
2. Copy the ArtifactsInstallation.tar.gz file to the *VI_install_dir* directory and decompress the file.
3. To set up the installer, go to the installer setup folder and then run the following script:
   `./setup.sh`
4. Access the installer by going to the following URL:
   `https://`*hostname*`:8080/UI/index.html`
   where *hostname* is the host name for installer. Review the license information and then click **Accept**.
5. Click **Predefined Configuration** to load a predefined configuration that includes one node and the **VI Artifact** component.
6. Change the values in the **Property Editor** to the configuration of the VI Artifact.
   - **Cluster name** refers to the cluster name of Hortonworks Data Platform. You can find the cluster name in the Hortonworks Data Platform admin console, or you can click **Admin** > **Manage Ambari** and check Clusters.

- **Ambari host name** refers to the name of host where the Ambari server is installed. The artifact installer connects to the Ambari server to get Hortonworks Data Platform service status and connection information. If Kerberos is enabled in Hortonworks Data Platform, set **Kerberos is set as security authentication** to **True**.
- The hbase keytab file and hdfs keytab file are files in the `/etc/security/keytabs` folder.

7. Change the values in the **Property Editor** to the configuration of the deployed training server.

   The **subnet mask** refers to the network that can access the shared folder of the training server. You can set this value as *IP_of_VI_center_server*/29.
8. Click **Validate** to validate the configuration.
9. Click **Run** to run the installation.
10. In the installer setup folder, run the following script to clean up the installation files:
    `./cleanup.sh`

Post-installation tasks

Complete these tasks after you install Maximo PQI On-Premises Visual Insights.

Compiling the YOLO library with Maximo PQI On-Premises Visual Insights artifacts

After you install Maximo PQI On-Premises Visual Insights on the training server, you can compile the YOLO library again with Maximo PQI On-Premises Visual Insights artifacts.

About this task

Procedure

1. Add the `/home/pmqopsadmin/vi_obj_detection_retrain/model_library/yolo/detectorobj.c` file to the `darknet/examples` folder.
2. Edit the `darknet/Makefile` file and indicate that `EXECOBJA=detectorobj.o`. The following code is an example of the code in the `Makefile` file:
   `EXECOBJA=detectorobj.o captcha.o lsd.o super.o voxel.o art.o tag.o cifar.o go.o rnn.o rnn_vid.o compare.o segmenter.o regressor.o classifier.o coco.o dice.o yolo.o detector.o writing.o nightmare.o swag.o darknet.o`
3. In the `Makefile` file, add `$(EXECOBJ)` for the `$(SLIB)` and `$(ALIB)` objects. The following code is an example of the code in the `Makefile` file:

```
$(ALIB): $(EXECOBJ) $(OBJS)
        $(AR) $(ARFLAGS) $@ $^
$(SLIB): $(EXECOBJ) $(OBJS)
        $(CC) $(CFLAGS) -shared $^ -o $@ $(LDFLAGS)
```

4. Run the following command:
   make
5. Copy the `iotmyolo.py` file and add it to `YOLO_HOME` directory.

Creating system-level users

Before you first login to the product, you must create operating system-level users and groups on each Hortonworks Data Platform management node and slave node.

Procedure

1. Run the following command on each Hortonworks Data Platform management node and slave node:
   `groupadd vi`
   `useradd -g vi -s /bin/bash -p pw@VI VI`
2. If Kerberos is enabled in the Hortonworks Data Platform cluster, distribute the `/etc/security/keytabs/VI.keytab` keytab file from the Ambari node to all other Hortonworks Data Platform nodes.

Verifying the training services on the training server

Verify that the training services have started on the training server.

Procedure

1. Log in to the training server using secure shell (SSH).
2. Run the following command:
   `ps aux | grep python`
   The result should include the following Python services for classification model training:
   - /home/pmqopsadmin/selflearning/services/run.py
   - /home/pmqopsadmin/selflearning/selflearning/runScheduler.py
   - /home/pmqopsadmin/selflearning/selflearning/runJobRunner.py

   If any of the Python services do not exist, check the log files, fix any errors, and restart the services. The log files are as follows:
   - `/home/pmqopsadmin/selflearning/selflearning.log`
   - `/home/pmqopsadmin/selflearning/jobrunner.log`
   - `/home/pmqopsadmin/selflearning/scheduler.log`

   If you find the error "no such table: T_Queue_Job," run the following commands to create table:

   ```
   cd /home/pmqopsadmin/selflearning
   python selflearning/initdb.py
   ```

   The command to restart the classification model training server is as follows:

   ```
   source ~/.bashrc
   cd /home/pmqopsadmin/selflearning
   ./restartSelflearning.sh
   ```

   The result should also include the following Python services for object detection model training:
   - /home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/run.py 5060
   - /home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/run.py 5061

   If any of the Python services do not exist, check the log files, fix any errors, and restart the services. The log files are as follows:
   - `/home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/frcnn_log.txt`
   - `/home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/ssd_log.txt`
   - `/home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/jobDaemon.log`

   If you see a "No module named iotmyolo" error message, copy the `iotmyolo.so` or `iotmyolo.py` file from the `/home/pmqopsadmin/vi_obj_detection_retrain/model_library/yolo` directory to the YOLO_HOME directory.
   The command to restart the object detection model training server is as follows:

   ```
   source ~/.bashrc
   cd /home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/
   ./restartTrainingServer.sh
   ```

Logging in

You can log in to the product by accessing the Web interface.

Before you begin

Before you log in to the product, make sure the Liberty server provisioning console is valid.

Procedure

1. To log in to the system, go to the following URL:
   `https://<appnode IP>:9447/ibm/iotm/solution/`
2. Enter your user name and password.

Default users

Maximo PQI On-Premises Visual Insights has two default users that you can use to access the product.

The following users are provided with the product.

Table 2. Default users

User	Role
demouser	Inspector Supervisor
demomgr	Model Manager

You can change the passwords for these users by editing the `server.xml` file in the Maximo PQI On-Premises Visual Insights server installation folder.

If you cannot log in to Maximo PQI On-Premises Visual Insights, open the server.xml file and uncomment the user and password sections.

Product license files

After you manually install IBM Maximo PQI On-Premises Visual Insights, you must ensure that the appropriate `swidtag` file and `license` directory exists on each computer on which you installed a Maximo PQI On-Premises Visual Insights component.

License files

The `swidtag` file and `license` directory must exist on the Maximo PQI On-Premises Visual Insights node computer.

Note: Do not rename the directory or files.

If you installed on a Red Hat Enterprise Linux operating system by using the solution installer, the license files are automatically copied to `/opt/IBM/VI_1.4/license` and the swidtag files are automatically copied to `/opt/IBM/VI_1.4/prod/iso-swid` or /opt/IBM/VI_1.4/non-prod/iso-swid on the `Maximo PQI On-Premises Visual Insights` node.

Application licensing and the slmtag file

A licensing application runs on the Maximo PQI On-Premises Visual Insights Integration Bus node. The application periodically logs the number of images that are classified by the Maximo PQI On-Premises Visual Insights system into a file with the `.slmtag` extension in the `/opt/IBM/VI_1.4/prod/iso-swid` or `/opt/IBM/VI_1.4/non-prod/iso-swid` folder.

Here is a sample `slmtag` file with the following code.

```
<SchemaVersion>2.1.1</SchemaVersion>
<SoftwareIdentity>
       <PersistentId>3ad60a8beaf2496a8b940813bba7417d</PersistentId>
       <Name>IBM Maximo PQI On-Premises Visual Insights</Name>
       <InstanceId>/IBM/PQI/VI</InstanceId>
</SoftwareIdentity>
<Metric logTime="2018-12-14T06:00:33+08:00">
       <Type>EDGE</Type>
       <SubType>NO_EDGES</SubType>
       <Value>1</Value>
       <Period>
              <StartTime>2018-12-13T06:00:33+08:00</StartTime>
              <EndTime>2018-12-14T06:00:33+08:00</EndTime>
       </Period>
```

```
</Metric>
<Metric logTime="2018-12-14T06:00:33+08:00">
       <Type>IMAGE</Type>
       <SubType>NO_TRAIN_IMAGES</SubType>
       <Value>1</Value>
       <Period>
              <StartTime>2018-12-13T06:00:33+08:00</StartTime>
              <EndTime>2018-12-14T06:00:33+08:00</EndTime>
       </Period>
</Metric>
```

Troubleshooting

Troubleshooting topics can help you identify and recover from common issues.

Server validation fails with check_requiretty.sh error

Validation fails during the server installation, and the log file shows an error about check_requiretty.sh.

About this task

This issue occurs because requiretty is enabled on the target machine. Complete the following steps to comment out the requiretty line in the `/etc/sudoers` file on the target system.

Procedure

1. Run the following command:
   `visudo`
2. Comment out the requiretty line as follows:
   `#Defaults requiretty`
3. Save the file and exit.
4. Retry the server installation and validation.

setup.sh fails with US-ASCII error

On a MacBook, setup.sh fails and the log file contains the following message: ERROR: ArgumentError: invalid byte sequence in US-ASCII.

About this task

This error occurs because the default encoding of a MacBook is US-ASCII rather than UTF-8. Complete the following steps to change the encoding of a MacBook to UTF-8.

Procedure

1. Run the following command on the MacBook:
   `vi ~/.bash_profile`
2. Add the following lines to the file:
   `export LC_ALL=en_US.UTF-8`
   `export LANG=en_US.UTF-8`
3. Save the file.
4. Run the following command:
   `source ~/.bash_profile`
5. Run the following command:
   `echo $LANG`
   If the result is `en_US.UTF-8`, the issue should be resolved.

Chapter 3. Provisioning the product

The provisioning console enables you to manage instances, tenants and users.

When Maximo PQI On-Premises Visual Insights is installed to your system, one instance, one tenant, and two users are created. You can find them in the provisioning console user interface. If you need to add more users or tenants, you can add them by using the provisioning console.

Creating tenants

A tenant is a group of users that are granted access to instances of one or more products. There is no limit to the number of users that can be assigned to a tenant. You can create one tenant for all your users, or you can create multiple tenants to group users and control their access to product instances.

Procedure

1. Log in to the provisioning console user interface by using the following URL: `http://hostname:port/`.
2. Click **Tenants**.
3. Click **Add Tenant**.
4. Enter an ID for the tenant. The ID must begin with a letter, the ID must contain only letters and numbers, and letters must be capitalized.
5. Enter an email address for the tenant.

   The tenant email address is used only as an identifier for the tenant. No email is sent to this address.
6. Enter a password for the tenant.
7. For tenant type, specify `internal`.
8. Leave the **Note** field blank.
9. Select **vi** in the instance list, select **Paid User**, and input the expiration date.
10. Click **Add**.
11. If your Hortonworks Data Platform has multiple nodes, after you create a tenant in the user interface, you must create OS-level users on each Hortonworks Data Platform management node and slave node, other than the Ambari node. To do this, complete the following steps:

    a) Run the following command on each Hortonworks Data Platform management node and slave node other than the Ambari node:
    ```
    useradd -g pq -s /bin/bash -p <tenantpassword> <tenant>
    usermod -a -G qews <tenant>
    ```

    b) If Kerberos is enabled in the Hortonworks Data Platform cluster, distribute the `/etc/security/keytabs/<tenant>.keytab` file from the Ambari node to all other Hortonworks Data Platform nodes.

Creating users

You can add users to tenants. A user can be added to only one tenant. After a user is added, the user's username and password can be used to log in to the Maximo PQI On-Premises Visual Insights user interface.

Procedure

1. Log in to the provisioning console user interface by using the following URL: `http://hostname:port/`.

2. Click **Users**.
3. Click **Add User**.
4. Enter the username.
5. Enter the user's email address.
6. Enter a password for the user.
7. Select one existing tenant.
8. Click **Next**.
9. In **Available solutions**, select **vi** and then click **add** to add it to the **In use solutions** list.
10. In the **In use solutions** list, edit the user properties:
    a. Specify one or more roles for the user: `modelmanager`, `inspector`, or `inspectorsupervisor`. If you specify multiple roles, separate them using a comma.
    b. For the cell list, add the cells one by one. One cell name includes the plant name, line name and cell name separated by comma, for example `plant1,line1,cell1` or `plant2,line2,cell2`.
11. Click **Create**.

Chapter 4. Creating edge systems

Maximo PQI On-Premises Visual Insights consists of the center application and the edges. Edges are Linux systems that are used to perform runtime defect detection.

Edge systems use the Caffe deep-learning framework. Caffe is a dedicated artificial neural network (ANN) training environment. Deep learning requires significant processing resources. Deep learning can be performed efficiently by using a graphics processing unit (GPU). Although most deep learning frameworks also support CPU processing, GPU processing provides reasonable performance for production environments.

Edges are clustered for load balancing. A cluster consists of one master edge and multiple slave edges. When a cell sends an image for processing, the master edge receives the image and then sends it to a slave edge that has GPU resources available.

The edge clustering architecture is scalable. If you need more processing resources, you can add more edges.

When you create an edge, you specify whether the edge is the master edge or a slave edge. The first edge that you create must be the master edge. After an edge is created, you cannot change the edge type. You can create only one master edge per tenant. The master edge cannot be deleted until all the slave edges are deleted.

Master edges can be connected or standalone. Connected means when an image is scored on an edge, the inspection result is sent to the center application immediately. Standalone means the inspection result is stored on the edge until you select **Pull results** from the **Preview Edge** dialog box or call the service that is deployed on the edge.

Edge system requirements

Before creating an edge system, ensure that your system meets the requirements.

- One of the following operating systems:
  - Ubuntu 16.04 on x86_64
  - Red Hat Enterprise Linux 7.5 on x86_64
  - Red Hat Enterprise Linux 7.5 on IBM Power System
- CPU architecture: x86_64 or IBM Power System
- 4-core processor
- 64GB memory
- 2TB hard disk drive
- One or more NVIDIA GPU cards

Opening edge ports

Before using edges, you must open the firewall ports that are used by the edge systems.

About this task

If uncomplicated firewall (UFW) is enabled and active on the edges, use the following commands to open UFW for network file system (NFS) service and open the following firewall ports on the edges:

```
sudo ufw enable
sudo ufw allow nfs
sudo ufw allow 22
sudo ufw allow 5005
```

```
sudo ufw allow 5070:5090/tcp
sudo ufw allow 6005
sudo ufw allow 8449
```

Installing NVIDIA GPU packages for Ubuntu

Use this task to install NVIDIA GPU packages for Ubuntu systems. To enable GPU processing, you must install the required NVIDIA GPU packages.

Procedure

1. Download and install the drivers for your NVIDIA GPU. The NVIDIA driver list for Ubuntu is available at the following link: Binary Driver How to - Nvidia. The following command is an example:
   ```
   sudo apt-get install ubuntu-drivers-common
   sudo ubuntu-drivers devices
   sudo apt-get install nvidia-384
   ```
2. Use the following command to check if your NVIDIA driver installed correctly:
   ```
   sudo nvidia-smi
   ```
3. Download and install the NVIDIA CUDA toolkit and corresponding CUDNN library. CUDA 8.0, CUDA 9.0 and CUDA 10.0 are supported. The following command is an example for CUDA 8.0.
   ```
   wget
   http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/
   cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
   ```
4. Install the CUDA file on the target server using the following commands:
   ```
   sudo dpkg -i cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
   sudo apt-get update
   sudo apt-get install cuda
   ```
   As NVIDIA upgrades the toolkit, you may get a newer version of the toolkit by using this command. It is recommended that you install CUDA 8.0 since that is the tested version. If you find that a newer version has been installed you can downgrade to CUDA 8.0 by using the following commands:
   ```
   sudo apt-get remove cuda
   sudo apt-get install cuda-8-0
   sudo ln -s /usr/local/cuda-8.0 /usr/local/cuda
   ```
5. Download the NVIDIA CUDA Deep Neural Network library `cudnn-8.0-linux-x64-v6.0.tgz` from the following link: https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v6/prod/8.0_20170307/cudnn-8.0-linux-x64-v6.0-tgz. You may need to create an account and log in before downloading the file.
6. Unpack the `cudnn-8.0-linux-x64-v6.0.tgz` file to the cuda installation directory using the following command:
   ```
   sudo tar -xvf cudnn-8.0-linux-x64-v6.0.tgz -C /usr/local
   ```
7. Set the environment variable using the following commands:
   ```
   export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
   export PATH=/usr/local/cuda/bin:$PATH
   ```
   Also add these commands to the `~/.bashrc` script.
8. Install the NVIDIA NCCL package using the following commands:
   ```
   git clone https://github.com/NVIDIA/nccl.git
   cd nccl
   sudo make install -j4
   ```

Installing NVIDIA GPU packages for Linux on NVIDIA Tesla K80 Power Systems Servers

Use this task to install NVIDIA GPU packages for Linux on NVIDIA Tesla K80 Power Systems Servers. To enable GPU processing, you must install the required NVIDIA GPU packages.

About this task

For Power8 systems, use CUDA 8 as described in the following task. For Power9 systems, substitute CUDA 10 for CUDA 8 in the following task.

Procedure

1. Download and install the drivers for your NVIDIA GPU. The NVIDIA driver list for Linux on Power Systems Servers is available at the following link: NVIDIA Driver Downloads. Follow the installation instructions on the download page.
2. Download the CUDA repository file and install CUDA 8 by using the following commands:
   ```
   wget https://developer.download.nvidia.com/compute/cuda/repos/rhel7/ppc64le/cuda-repo-rhel7-8.0.61-1.ppc64le.rpm
   rpm -i cuda-repo-rhel7-8.0.61-1.ppc64le.rpm
   yum clean all
   yum install cuda
   ```
3. For CUDA 8, download the NVIDIA CUDA Deep Neural Network library `cudnn-8.0-linux-ppc64le-v6.0-tgz` from the following URL: https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v6/prod/8.0_20170307/cudnn-8.0-linux-ppc64le-v6.0-tgz. For CUDA 10, download the NVIDIA CUDA Deep Neural Network library `cudnn-10.0-linux-ppc64le-v7.3.1.20.tgz` from the following URL: https://developer.download.nvidia.com/compute/cuda/repos/rhel7/ppc64le/cuda-10.0.130-1.ppc64le.rpm. You might need to create an account and log in before downloading the file.
4. Unpack the `cudnn-8.0-linux-ppc64le-v6.0.tgz` file to the cuda installation directory by using the following command:
   ```
   sudo tar -xvf cudnn-8.0-linux-ppc64le-v6.0.tgz -C /usr/local
   ```
5. Set the environment variable by using the following command:
   ```
   export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
   ```
   Also add this command to the ~/.bashrc script.
6. Install the NVIDIA NCCL package using the following commands:
   ```
   git clone https://github.com/NVIDIA/nccl.git
   cd nccl
   sudo make install -j4
   ```

Installing NVIDIA GPU packages for Linux on NVIDIA Tesla V100 Power Systems Servers

Use this task to install NVIDIA GPU packages for Linux on NVIDIA Tesla V100 Power Systems Servers. To enable GPU processing, you must install the required NVIDIA GPU packages.

Procedure

1. Download and install NVIDIA CUDA 9.2.148 from developer.nvidia.com/cuda-92-download-archive.
   a) Select Operating System: Linux.
   b) Select Architecture: ppc64le.
   c) Select Distribution: RHEL.
   d) Select Version: 7.

e) Select Installer Type: rpm (local). The local rpm is preferred over the network rpm as it ensures the version that is installed is the version that is downloaded. With the network rpm, the `yum install cuda` command always installs the latest version of the CUDA Toolkit.

f) Click **Download** to download the base installer.

g) Click **Download** to download patch 1.

h) Follow the Linux on POWER installation instructions in the CUDA Quick Start Guide, including the steps that describe how to set up the CUDA development environment by updating PATH and `LD_LIBRARY_PATH`.

2. Download NVIDIA driver 410.104 from http://www.nvidia.com/Download/index.aspx.
   a) Select Product Type: Tesla.
   b) Select Product Series: V-Series.
   c) Select Product: Tesla V100.
   d) Select Operating System: Linux POWER LE RHEL 7.
   e) Select CUDA Toolkit:10.0.
   f) Click **Search** to go to the download link and then click **Download**.
3. **Note:** For IBM Power System AC922 systems, operating system and system firmware updates are required before you install the latest GPU driver.

   Install CUDA and the GPU driver:
   a) Install the CUDA Base repository rpm.
   b) Install the CUDA Patch 1 repository rpm.
   c) Install the GPU driver repository rpm.
   d) Run the following command to install CUDA, patch, and GPU driver:
   ```
   sudo yum install cuda
   ```
   e) Restart the system to activate the driver.
4. Enable NVIDIA system-persistenced services by using the following shell command:
   ```
   systemctl enable nvidia-persistenced
   ```
5. Check NVIDIA drivers by using the following shell command:
   ```
   nvidia-smi
   ```
6. Download NVIDIA cuDNN v7.4.2 for CUDA 10.0 (cuDNN v7.4.2 Library for Linux (Power8/Power9)) from developer.nvidia.com/cudnn. Registration in the NVIDIA Accelerated Computing Developer Program is required.
7. Download NVIDIA NCCL v2.3.7 for CUDA 10.0 (NCCL 2.3.7 O/S agnostic and CUDA 10.0 and IBM Power) from developer.nvidia.com/nccl Registration in the NVIDIA Accelerated Computing Developer Program is required.
8. Install the cuDNN v7.4.2 and NCCL v2.3.7 packages and then refresh the shared library cache by using the following commands:
   ```
   sudo tar -C /usr/local --no-same-owner -xzvf cudnn-9.2-linux-ppc64le-v7.4.2.tgz
   sudo tar -C /usr/local --no-same-owner -xzvf nccl_2.3.7+cuda10.0_ppc64le.tgz
   sudo ldconfig
   ```

Installing Caffe for Ubuntu

You must install the Caffe deep-learning framework and related packages. Caffe is used for model training and defect classification.

Procedure

1. Install the packages that are required for Caffe by using the following commands:
   ```
   sudo apt-get update
   ```

```
sudo apt-get upgrade
sudo apt-get install -y build-essential cmake git pkg-config
sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev
libhdf5-serial-dev protobuf-compiler
sudo apt-get install -y libatlas-base-dev libjasper-dev
sudo apt-get install -y --no-install-recommends libboost-all-dev
sudo apt-get install -y libgflags-dev libgoogle-glog-dev liblmdb-dev
sudo apt-get install -y python-pip
sudo apt-get install -y python-dev
sudo apt-get install -y python-numpy python-scipy
sudo apt-get install -y libopencv-dev
sudo pip install opencv-python
sudo pip install flask_httpauth
sudo pip install gevent
sudo pip install pyinotify
sudo pip install tornado
```

2. Download the Caffe source code by using the following command:
   `wget https://github.com/BVLC/caffe/archive/1.0.zip`
3. Unpack the package and enter the package directory by using the following commands:
   ```
   unzip 1.0.zip
   cd ./caffe-1.0
   ```
4. Make a copy of the make configuration file by using the following command:
   `cp Makefile.config.example Makefile.config`
5. Add the following variables in the `Makefile.config` file:
   ```
   USE_CUDNN := 1
   CUDA_DIR := /usr/local/cuda
   PYTHON_INCLUDE := /usr/include/python2.7 \
   /usr/lib/python2.7/dist-packages/numpy/core/include
   PYTHON_LIB := /usr/lib/x86_64-linux-gnu
   WITH_PYTHON_LAYER := 1
   INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include \
   /usr/include/hdf5/serial
   LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib \
   /usr/lib/x86_64-linux-gnu /usr/lib/x86_64-linux-gnu/hdf5/serial
   ```
6. In the `caffe-1.0` directory, run the following command:
   ```
   find . -type f -exec sed -i -e 's^"hdf5.h"^"hdf5/serial/hdf5.h"^g' -e
   's^"hdf5_hl.h"^"hdf5/serial/hdf5_hl.h"^g' '{}' \;
   ```
7. Run the following commands:
   ```
   cd /usr/lib/x86_64-linux-gnu
   sudo ln -s libhdf5_serial.so.10.1.0 libhdf5.so
   sudo ln -s libhdf5_serial_hl.so.10.0.2 libhdf5_hl.so
   ```
8. Install the required Python packages in the `caffe-1.0/python` directory by using the following commands:
   ```
   cd {caffe-installation-path}/caffe-1.0/python
   for req in $(cat requirements.txt); do sudo -H pip install $req --upgrade;
   done
   ```
   where `{caffe-installation-path}` is the Caffe deployment path.
9. Open the makefile in the `{caffe-installation-path}` directory and change the parameter NVCCFLAGS to the following setting:
   ```
   NVCCFLAGS += -D_FORCE_INLINES -ccbin=$(CXX) -Xcompiler -fPIC $
   (COMMON_FLAGS)
   ```
10. In the main Caffe directory `caffe-1.0`, begin the Caffe build and installation by using the following commands:
    ```
    make all
    make test
    ```

```
make runtest
make pycaffe
make distribute
```

11. Add the following line to the `~/.bashrc` script:
    ```
    export PYTHONPATH="/usr/lib/python2.7:{caffe-installation-path}/caffe-1.0/
    python:$PYTHONPATH"
    ```
    where `{caffe-installation-path}` is the Caffe deployment path.

Installing Caffe for Linux on Power Systems Servers

Use this task to install Caffe for Linux on Power Systems Servers systems. You must install the Caffe deep-learning framework and related packages. Caffe is used for model training and defect classification.

Procedure

1. Install the packages that are required for Caffe by using the following commands:
   ```
   sudo yum clean all
   sudo yum update
   sudo yum install upgrade
   sudo yum install -y libboost-*
   sudo yum install -y gflags-devel glog-devel lmdb-devel
   sudo yum install -y python-pip
   sudo yum install -y python-devel
   sudo yum install -y opencv-devel
   sudo yum makecache
   sudo yum install -y protobuf-devel leveldb-devel lmdb-devel snappy-devel
   opencv-devel boost-devel hdf5-devel atlas-devel glog-devel gflags-devel
   sudo yum install libpng-devel
   sudo yum install freetype-devel
   sudo yum install libjpeg-turbo-devel
   sudo yum install opencv-python
   sudo rpm -e --nodeps numpy
   sudo pip install numpy
   pip install --upgrade pip
   sudo pip install flask_httpauth
   sudo pip install gevent
   sudo pip install pyinotify
   ln -s /usr/local/cuda-10.0 /usr/local/cuda
   pip install scikit-image
   sudo pip install tornado
   ```
2. Link the Atlas library by using the following commands:
   ```
   ln -fs /usr/lib64/atlas/libsatlas.so /usr/lib64/libatlas.so
   ln -fs /usr/lib64/atlas/libsatlas.so /usr/lib64/libcblas.so
   ```
3. Download the Caffe source code by using the following command:
   ```
   wget https://github.com/BVLC/caffe/archive/1.0.zip
   ```
4. Unpack the package and enter the package directory by using the following commands:
   ```
   unzip 1.0.zip
   cd ./caffe-1.0
   ```
5. Replace the following Caffe files:
   - In the `/include/caffe/util/cudnn.hpp` directory, replace the `cudnn.hpp` file with the newest `cudnn.hpp` file that is in the Caffe repository on GitHub: github.com/BVLC/caffe.git
   - Replace all of the cudnn files that are in the `/src/caffe/layers` folder with the newest cudnn files that are in the Caffe repository on GitHub: github.com/BVLC/caffe.git

   For example, run the following commands:

```
cp -rf /root/source/caffe-git/caffe-master/include/caffe/util/
cudnn.hpp /usr/local/caffe-1.0/include/caffe/util/
cp -rf /root/source/caffe-git/caffe-master/src/caffe/layers/cudnn_* /usr/
local/caffe-1.0/src/caffe/layers/
cp -rf /root/source/caffe-git/caffe-master/include/caffe/layers/
cudnn_* /usr/local/p/usr/local/caffe-1.0/include/caffe/layers/
```

6. Make a copy of the make configuration file by using the following command:
   `cp Makefile.config.example Makefile.config`
7. Add the following variables in the Makefile.config file:

```
USE_CUDNN := 1
CUDA_DIR := /usr/local/cuda
PYTHON_INCLUDE := /usr/include/python2.7 \
     /usr/lib64/python2.7/site-packages/numpy/core/include/
PYTHON_LIB := /usr/lib/gcc/ppc64le-redhat-linux/4.8.5/
WITH_PYTHON_LAYER := 1
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include
/usr/local/cuda-10.0/targets/ppc64le-linux/include/
LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib \
    /usr/lib64 /usr/local/lib64
```

   Change CUDA_ARCH to the following text:

```
CUDA_ARCH := -gencode arch=compute_30,code=sm_30
-gencode arch=compute_35,code=sm_35
-gencode arch=compute_50,code=sm_50
-gencode arch=compute_52,code=sm_52
-gencode arch=compute_60,code=sm_60
-gencode arch=compute_61,code=sm_61
-gencode arch=compute_61,code=compute_61
```

8. Install the required Python packages in the caffe-1.0/python directory by using the following commands:
   `cd` *`caffe-installation-path`*`/caffe-1.0/python`
   `for req in $(cat requirements.txt); do sudo -H pip install $req --upgrade; done`
   where *caffe-installation-path* is the Caffe deployment path.
9. Open the Makefile in the *caffe-installation-path* directory and change the parameter NVCCFLAGS to the following setting:
   `NVCCFLAGS += -D_FORCE_INLINES -ccbin=$(CXX) -Xcompiler -fPIC $(COMMON_FLAGS)`
10. In the main Caffe directory `caffe-1.0`, begin the Caffe build and installation by using the following commands:
    ```
    make all
    make test
    make runtest
    make pycaffe
    make distribute
    ```
11. Add the following line to the ~/.bashrc script:
    `export PYTHONPATH="/usr/lib/python2.7:`*`caffe-installation-path`*`/caffe-1.0/python:$PYTHONPATH"`
    where *caffe-installation-path* is the Caffe deployment path.
12. Run the following post-installation tests:
    a) `make runtest | tee -a runtest.out`
    b) `grep -i OK runtest.out | wc –l`
       Caffe test output must be 2101
    c) `python -c "import caffe"`
       to test the Pycaffe installation
    d) `tail -n 2 runtest.out`
       The contents of `runtest.out` should contain the following text:

```
[==========] 2101 tests from 277 test cases ran. (291548 ms total)
[ PASSED ] 2101 tests
```

Troubleshooting the Caffe installation

If an error message displays in the log when you begin the Caffe build and installation, you can take steps to try to resolve the problem.

Symptoms 1

When you began the Caffe build and installation, the following message displays:

```
    1.    In file included from ./include/caffe/util/device_alternate.hpp:40:0,
    2.                     from ./include/caffe/common.hpp:19,
    3.                     from src/caffe/common.cpp:7:
    4.    ./include/caffe/util/cudnn.hpp: In function ‘void
caffe::cudnn::createPoolingDesc(cudnnPoolingStruct**,
        caffe::PoolingParameter_PoolMethod, cudnnPoolingMode_t*, int, int, int, int, int,
int)’:
    5.    ./include/caffe/util/cudnn.hpp:127:41: error: too few arguments to function
‘cudnnStatus_t
        cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t, cudnnPoolingMode_t,
cudnnNanPropagation_t, int,
        int, int, int, int, int)’
    6.             pad_h, pad_w, stride_h, stride_w));
    7.                                               ^
    8.    ./include/caffe/util/cudnn.hpp:15:28: note: in definition of macro ‘CUDNN_CHECK’
    9.         cudnnStatus_t status = condition; \
    10.                                ^
    11.    In file included from ./include/caffe/util/cudnn.hpp:5:0,
    12.                     from ./include/caffe/util/device_alternate.hpp:40,
    13.                     from ./include/caffe/common.hpp:19,
    14.                     from src/caffe/common.cpp:7:
    15.    /usr/local/cuda-7.5//include/cudnn.h:803:27: note: declared here
    16.     cudnnStatus_t CUDNNWINAPI cudnnSetPooling2dDescriptor(
    17.                               ^
    18.    make: *** [.build_release/src/caffe/common.o] Error 1
    19.
```

Resolving the problem 1

To fix the error, refer to the following steps:

1. In the `/include/caffe/util/cudnn.hpp` directory, replace the `cudnn.hpp` file with the newest `cudnn.hpp` file that is in the Caffe repository on GitHub.
2. In the `/src/caffe/layers` folder, replace all of the `cudnn` files that are in the `/src/caffe/layers` folder with the newest `cudnn` files that are in the Caffe repository on GitHub.

Symptoms 2

When you install the required Python packages in the `caffe-1.0/python` directory, the following message displays:

```
Traceback (most recent call last):
  File "/usr/bin/pip", line 11, in <module>
    sys.exit(main())
  File "/usr/lib/python2.7/dist-packages/pip/__init__.py", line 215, in main
    locale.setlocale(locale.LC_ALL, '')
  File "/usr/lib/python2.7/locale.py", line 581, in setlocale
    return _setlocale(category, locale)
locale.Error: unsupported locale setting
Traceback (most recent call last):
  File "/usr/bin/pip", line 11, in <module>
    sys.exit(main())
  File "/usr/lib/python2.7/dist-packages/pip/__init__.py", line 215, in main
    locale.setlocale(locale.LC_ALL, '')
  File "/usr/lib/python2.7/locale.py", line 581, in setlocale
```

```
    return _setlocale(category, locale)
locale.Error: unsupported locale setting
```

Resolving the problem 2

To resolve this error, run the following command:
`export LC_ALL=C`

Symptoms 3

When you begin the Caffe build and installation, the following message displays:

```
nvcc fatal   : Unsupported gpu architecture 'compute_20'
Makefile:595: recipe for target '.build_release/cuda/src/caffe/layers/prelu_layer.o' failed
make: *** [.build_release/cuda/src/caffe/layers/prelu_layer.o] Error 1
```

Resolving the problem 3

Comment out `-gencode arch=compute_20` in `Makefile.config`.

Symptoms 4

When you begin the Caffe build and installation, the following message displays:

```
PROTOC src/caffe/proto/caffe.proto
make: protoc: Command not found
Makefile:639: recipe for target '.build_release/src/caffe/proto/caffe.pb.cc' failed
make: *** [.build_release/src/caffe/proto/caffe.pb.cc] Error 127
```

Resolving the problem 4

Run the following command to install the protoc program:

sudo apt install protobuf-compiler

Symptoms 5

When you begin the Caffe build and installation, the following message displays:

```
src/caffe/layers/hdf5_data_layer.cpp:13:30: fatal error: hdf5/serial/hdf5.h: No such file or
directory
compilation terminated.
Makefile:582: recipe for target '.build_release/src/caffe/layers/hdf5_data_layer.o' failed
make: *** [.build_release/src/caffe/layers/hdf5_data_layer.o] Error 1
```

Resolving the problem 5

You might need to install a Caffe dependency package using the following commands:

sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libopencv-dev libhdf5-serial-dev protobuf-compiler
sudo apt-get install --no-install-recommends libboost-all-dev
sudo apt-get install libopenblas-dev liblapack-dev libatlas-base-dev
sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev

Installing Open CV

You can install the Open Source Computer Vision (OpenCV) 3.2 library if you need to customize models with the features of OpenCV 3.2.

Procedure

1. Get the OpenCV source code from Github:
   ```
   wget https://github.com/opencv/opencv/archive/3.2.0.zip
   ```
2. Unpack the downloaded package and change to the package directory:
   ```
   unzip 3.2.0.zip
   cd opencv-3.2.0
   ```
3. Create a building subdirectory and change to the directory:
   ```
   mkdir build
   cd build
   ```
4. Prepare and generate the building configuration:
   ```
   cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D
   WITH_TBB=ON -D WITH_V4L=ON ..
   ```
5. Compile and build the package:
   ```
   make -j $(($(nproc) + 1))
   ```
6. Install the package:
   ```
   sudo make install
   ```
7. Register the libraries and modules to the system:
   ```
   sudo /bin/bash -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'
   sudo ldconfig
   ```
8. If required, uninstall the old opencv version to avoid version collision:
   ```
   sudo apt-get autoremove libopencv-dev
   ```

Installing object detection libraries

You install an object detection library so that you can run the object detection model on an edge.

About this task

IBM Maximo PQI On-Premises Visual Insights supports the following object detection libraries: YOLO (you only look once), Faster R-CNN, and SSD (Single Shot MultiBox Detector).

Procedure

1. Install the related Python packages by using the following commands:
   ```
   sudo apt-get install python-numpy
   sudo apt-get install python-scipy
   sudo pip install cython
   sudo pip install easydict
   sudo pip install uuid
   sudo pip install multiprocessing
   ```
2. Install all of the following libraries:

Library	Installation Instructions

YOLO version 2 library	a. Run the following commands to get the YOLO source code:
	`git clone --recursive https://github.com/pjreddie/darknet.git`
	`cd darknet`
	`git checkout 691debd`

Library	Installation Instructions
	b. Edit the Makefile file by enabling the **GPU**, and select the correct **GPU ARCH** parameter according to your machine configuration: **vi Makefile** **GPU=1** c. Run the following command to compile YOLO: **make**

Faster-RCNN Python library	a. Run the following command to get the Faster R-CNN source code:
	git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git
	b. In the py-faster-rcnn directory, in the lib folder, run the following command to Compile Cython:
	make
	c. Go to the caffe-fast-rcnn directory under the py-faster-rcnn directory and make a copy of the make configuration file by using the following commands:
	cd caffe-fast-rcnn
	cp Makefile.config.example Makefile.config
	d. Add the following variables to the Makefile.config file:
	`USE_CUDNN := 1`
	`CUDA_DIR := /usr/local/cuda`
	`PYTHON_INCLUDE := /usr/include/python2.7 \`
	`/usr/lib/python2.7/dist-packages/numpy/core/include`
	`PYTHON_LIB:=/usr/lib/x86_64-linux-gnu`
	`WITH_PYTHON_LAYER := 1`
	`INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include \`
	`/usr/include/hdf5/serial`
	`LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib \`
	`/usr/lib/x86_64-linux-gnu /usr/lib/x86_64-linux-gnu/hdf5/serial`
	e. Run the following command to compile Caffe:
	make
	f. Run the following command to compile pycaffe by using the Python layer:
	make pycaffe

SSD library	a. Run the following command to get the SSD source code: **git clone --recursive https://github.com/weiliu89/caffe.git ~/ssd-caffe** **cd ~/ssd-caffe** **git checkout ssd** b. Make a copy of the Makefile configuration file by using the following command: **cp Makefile.config.example Makefile.config** c. Edit the Makefile configuration file and change the **CUDA_ARCH**, **BLAS**, and **PYTHON_INCLUDE** parameters according to your machine configuration. d. Run the following command on one line: **find . -type f -exec sed -i -e 's^"hdf5.h"^"hdf5/serial/hdf5.h"^g' -e 's^"hdf5_hl.h"^"hdf5/serial/hdf5_hl.h"^g' '{}' \;** e. Compile the code by using following command: **make -j8** f. Compile the Python layer by using the following command:

Library	Installation Instructions
	make py g. Compile the test by using the following command: **make test -j8**

3. Add the following environment variables to the ~/.bashrc file: *YOLO_HOME*, *FRCNN_HOME*, and *SSD_HOME*. The following text is an example of adding environment variables: YOLO_HOME=~/darknet/ FRCNN_HOME=~/py-faster-rcnn/ SSD_HOME=~/ssd-caffe/.

Troubleshooting the Faster-RCNN Python library installation

If an error message displays when you install the Faster-RCNN Python library, you can take steps to try to resolve the problem.

Symptoms

When you install the Faster-RCNN Python library, the following cudnn compile error displays:

```
CXX src/caffe/layers/hdf5_data_layer.cpp
./include/caffe/util/cudnn.hpp: In function 'const char* cudnnGetErrorString(cudnnStatus_t)':
./include/caffe/util/cudnn.hpp:21:10: warning: enumeration value
'CUDNN_STATUS_RUNTIME_PREREQUISITE_MISSING' not handled in switch
./include/caffe/util/cudnn.hpp:15:28: note: in definition of macro 'CUDNN_CHECK'
     cudnnStatus_t status = condition;
Makefile:563: recipe for target '.build_release/src/caffe/layers/hdf5_data_layer.o' failed
make: *** [.build_release/src/caffe/layers/hdf5_data_layer.o] Error 1
```

On Power9 systems with CUDA 10, the following cudnn compile error displays:

```
./include/caffe/util/cudnn.hpp:127:41: error: too few arguments to function
'cudnnStatus_t cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t, cudnnPoolingMode_t,
cudnnNanPropagation_t, int, int, int, int, int, int)' pad_h, pad_w, stride_h, stride_w));
^./include/caffe/util/cudnn.hpp:15:28: note: in definition of macro 'CUDNN_CHECK'
cudnnStatus_t status = condition; \ ^In file included from ./include/caffe/util/
cudnn.hpp:5:0, from ./include/caffe/util/device_alternate.hpp:40, from ./include/
caffe/common.hpp:19, from ./include/caffe/blob.hpp:8, from src/caffe/blob.cpp:4:/
usr/local/cuda/include/cudnn.h:991:1: note: declared here cudnnSetPooling2dDescriptor
(cudnnPoolingDescriptor_t poolingDesc, ^
```

Resolving the problem

To fix the error, find your Caffe installation and run the following commands:
cp -rf ~/caffe-1.0/include/caffe/util/cudnn.hpp ~/py-faster-rcnn/caffe-fast-rcnn/include/caffe/util/
cp -rf ~/caffe-1.0/src/caffe/layers/cudnn_* ~/py-faster-rcnn/caffe-fast-rcnn/src/caffe/layers/
cp -rf ~/caffe-1.0/include/caffe/layers/cudnn_* ~/py-faster-rcnn/caffe-fast-rcnn/include/caffe/layers/

For Power9 systems with CUDA 10, edit the Makefile.config file by using the following command:
vim ~/py-faster-rcnn/caffe-fast-rcnn/Makefile.config
Change the CUDA_ARCH value as follows:

```
CUDA_ARCH := -gencode arch=compute_30,code=sm_30 \
  -gencode arch=compute_35,code=sm_35 \
  -gencode arch=compute_50,code=sm_50 \
  -gencode arch=compute_50,code=compute_50
```

Configuring the image server

You configure the image server on the master edge machine so that the master edge machine can store images that are captured by an industrial camera. The image server is monitored by the edge controller.

When a new image is added, it is scored and the inspection result is sent to the center application so that an inspector can evaluate the image and the inspection results.

Procedure

1. Install and start the NFS service on the master edge system by using the following command:
   `sudo apt-get install nfs-kernel-server`
   For Linux on Power Systems Servers, use the following commands:
   ```
   sudo yum install -y nfs-utils
   sudo yum install -y rpcbind
   systemctl start nfs
   systemctl start nfslock
   systemctl start rpcbind
   ```
2. Make a directory named `imageserver` and change the folder owner using following commands:
   ```
   sudo mkdir /imageserver
   sudo chown user /imageserver
   ```
   where *user* is the user you specify when you register the edge to the center application.
3. Optional: Complete the following steps only if you have both a master edge system and a slave edge system.
   a. To export the `imageserver` directory, edit the `/etc/exports` file by using the following command:
      `vi /etc/exports`
   b. Add the following line to your export file. The IP address network is the IP subnet that can access the shared folder. The format is:
      IP address`/`*subnet mask*. All slave edge systems must be able to access the shared folder:
      `/imageserver` *IP address*`/`*network*`(rw,sync,no_root_squash,no_all_squash)`
      For example:
      `/imageserver 10.173.0.0/29(rw,sync,no_root_squash,no_all_squash)`
   c. Restart the NFS service using following command:
      `sudo service nfs-server restart`
4. On each slave edge machine, mount the `/imageserver` directory to the shared folder of master edge:
   a) Run the following command on the slave edge machine:
      `mkdir /imageserver`
   b) Edit the `/etc/fstab` file using the following command:
      `vi /etc/fstab`
   c) Add the following line to the file, specifying the IP address of the master edge:
      IP_address`:/imageserver /imageserver nfs defaults 0 0`
   d) Run the following command on the slave edge machine to make the mount take effect:
      `mount -a`

Configuring the model store

You must configure the model store on the master edge machine. The model store is a repository for executable models that are distributed from the center application.

Procedure

1. Make a `modelstore` directory and change the folder owner on the master edge system by using following commands:
   ```
   sudo mkdir /modelstore
   sudo chown user /modelstore
   ```
   where *user* is the user you specify when you register the edge to the center application.

2. Optional: Complete the following steps only if you have both a master edge system and a slave edge system.
   a. To export the `modelstore` directory, edit the `/etc/exports` file by using the following command:
      `vi /etc/exports`
   b. Add the following line to your export file. The IP address network is the IP subnet that can access the shared folder. The format is:
      `IP address/subnet mask`. All slave edge systems must be able to access the shared folder:
      `/modelstore` *`IP address/network`*`(rw,sync,no_root_squash,no_all_squash)`
      For example:
      `/modelstore 10.173.0.0/29(rw,sync,no_root_squash,no_all_squash)`
   c. Restart the NFS service by using the following command:
      `sudo service nfs-server restart`
3. On each slave edge machine, mount the `/modelstore` directory to the shared folder of master edge:
   a) Run the following command on the slave edge machine:
      `mkdir /modelstore`
   b) Edit the `/etc/fstab` file using the following command:
      `vi /etc/fstab`
   c) Add the following line to the file, specifying the IP address of the master edge:
      `IP_address``:/modelstore /modelstore nfs defaults 0 0`
   d) Run the following command on the slave edge machine to make the mount take effect:
      `mount -a`

Installing Python modules

Complete this task to install prerequisite Python modules.

Procedure

Run the following commands on the master edge system and each slave edge system to install prerequisite packages:

```
sudo pip install flask
sudo pip install gevent
sudo pip install requests
sudo pip install pyinotify
sudo pip install opencv-python
sudo pip install lmdb
```

For Ubuntu: `sudo apt-get install dos2unix`

For Linux on Power Systems Servers: `sudo yum install dos2unix`

```
sudo apt install curl
sudo pip install flask_httpauth
sudo pip install paramiko
sudo pip install tornado
```

Registering the edge to the center application

You must register edges to the center application. The procedure is different for connected and stand-alone edges.

There are two types of edges: connected edge and stand-alone edge. The connected edge has a public IP address that enables the center application to connect with the edge directly. The edge sends inspection results to the center application in real time. The stand-alone edge has no public IP, and sometimes it cannot connect to the center application. The inspection result must be stored locally on the stand-alone

edge, and the edge administrator must call the edge service explicitly to communicate with the center application when the edge connects to the internet.

Registering a connected edge to the center application

After you configure the connected edge system, you must register it in the center application. You can create an edge or edit an existing one. Edges are used to run a scoring model.

Procedure

1. In the **Model Manager**, select **Data** > **Edges**.
2. Select **Create new edge** and input the edge name.
3. Input the IP address, the SSH user name, and the password of the edge system, and specify the edge type as **Master** or **Slave**. The IP address must be accessible to the center application. The SSH user name and password are used to log in the edge system to deploy the edge controller.
4. If you are creating a master edge, specify connected for connection mode. If you are creating a slave edge, specify the corresponding master edge.
5. If you selected connection mode, select **Next** and specify the delete policy.
6. Select **Create** to create the edge. The edge controller and score engine deploy to the edge system, and the edge is added into the registered list.

Registering a stand-alone edge to the center application

After you configure a stand-alone edge system, you must register it in the center application. You can create an edge or edit an existing one. Edges are used to run a scoring model.

About this task

A stand-alone edge enables you to perform some functions of the product when you are disconnected from the center application, such as scoring images in the production line, syncing data to the center, and cleaning uploaded data. To install a stand-alone edge, you must install PostgreSQL and the required Python modules. You must also install the edge on the edge system manually.

Procedure

1. In the **Model Manager**, select **Data** > **Edges**.
2. Select **Create new edge** and input the edge name.
3. Input the IP address, the SSH user name, and the password of the edge system, and specify the edge type as **Master** or **Slave**. The IP address must be accessible to the center application. The SSH user name and password are used to log in the edge system to deploy the edge controller.
4. If you are creating a master edge, specify standalone for connection mode.
5. If you selected connection mode, select **Next** and specify the delete policy.
6. Select **Create** to create the edge. For a stand-alone edge, the edge administrator must perform an SSH login to the edge system and install the edge manually. For more information, refer to the Installing a stand-alone edge topic.

Installing a stand-alone edge

A stand-alone edge enables you to perform some functions of the product when you are disconnected from the center, such as scoring images in the production line, syncing data to the center, and cleaning uploaded data. To install a stand-alone edge, you must install PostgreSQL and the required python module. You must also install the edge onto the edge machine manually.

To create a stand-alone edge, you must choose the stand-alone connection mode when creating the edge.

You must create the stand-alone edge on the same system as the master edge.

Installing PostgreSQL for Ubuntu

Use this task to install PostgreSQL for Ubuntu systems. The training server and stand-alone edge use a PostgreSQL database.

Procedure

1. Install PostgreSQL, PHPPgadmin and Apache2 by using the following command:
   ```
   sudo apt-get -y install postgresql postgresql-contrib phppgadmin
   ```
2. Configure the PostgreSQL user.
   a) Log in as the PostgreSQL user by using the following commands:
   ```
   sudo su
   su - postgres
   psql
   ```
   b) Configure the password for user postgres by using the following commands:
   ```
   password postgres
   password
   \q
   ```
3. Configure Apache2 by editing the `nano phppgadmin.conf` file:
   ```
   cd /etc/apache2/conf-available/
   nano phppgadmin.conf
   ```
   Delete the following line: `Require local`. Add the following line to the file:
   ```
   Require all granted
   ```
4. Configure the PHPPgadmin by editing the `config.inc.php` file:
   ```
   cd /etc/phppgadmin/
   nano config.inc.php
   ```
   Find the following line in the file:
   ```
   $conf['extra_login_security'] = true
   ```
   Change `true` to `false`.
5. Restart PostreSQL and Apache2 by using the following commands:
   ```
   systemctl restart postgresql
   systemctl restart apache2
   ```
6. Verify that you can access the user interface on the stand-alone edge by accessing the following URL:
   `http://`*`standalone_edge_IP`*`/phppgadmin`
   where *standalone_edge_IP* is the IP address of the stand-alone edge.
7. Create the database schema in PostgreSQL.
   a) Run the following command on the SQL console on PHPPgadmin:
   ```
   create database edge with owner postgres encoding='UTF-8'
   lc_collate='en_US.utf8' lc_ctype='en_US.utf8' template template0;
   ```
   b) In the database, create the following tables:
   ```
   CREATE TABLE vi_tenant_inspectionresult(id text, info jsonb);
   CREATE TABLE vi_tenant_notification(id text, info jsonb);
   CREATE TABLE vi_tenant_defectsummary(id text, info jsonb);
   CREATE TABLE vi_tenant_uploaddataset(id text, info jsonb);
   CREATE TABLE vi_tenant_syncprocess(id text, info jsonb);
   CREATE TABLE vi_tenant_model(id text, info jsonb);
   CREATE TABLE vi_tenant_datagroup(id text, info jsonb);
   ```
   where *tenant* is the tenant for the operation user in the Maximo PQI On-Premises Visual Insights center. Get the tenant value from the user profile in the user interface for the center application.

Installing PostgreSQL for Linux on Power Systems Servers

Use this task to install PostgreSQL for Linux on Power Systems Servers. The training server and stand-alone edge use a PostgreSQL database.

Procedure

1. su to root by using the following command:
   `sudo su`
2. Download the source for PostgreSQL:
   `wget https://ftp.postgresql.org/pub/source/v9.5.13/postgresql-9.5.13.tar.gz`
3. Install PostgreSQL by using the following commands:
   ```
   tar -zxvf postgresql-9.5.13.tar.gz
   cd postgresql-9.5.13/
   yum -y install readline-devel
   ./configure --prefix=/usr/local/postgresql
   make
   make install
   ```
4. Create user postgres and change owner for the `postgres` directory:
   ```
   useradd postgres
   chown -R postgres:postgres /usr/local/postgresql/
   ```
5. Change to user postgres:
   `su postgres`
6. Configure the system path for postgres:
   ```
   vi ~/.bashrc
   PGHOME=/usr/local/postgresql
   export PGHOME
   PGDATA=/usr/local/postgresql/data
   export PGDATA
   PATH=$PATH:$HOME/.local/bin:$HOME/bin:$PGHOME/bin
   export PATH
   ```
7. Source the configuration:
   `source ~/.bashrc`
8. Initialize the PostgreSQL database:
   `initdb`
9. Configure the database. Open `postgresql.conf` in vi:
   `vi /usr/local/postgresql/data/postgresql.conf`
   Change:

   ```
   #listen_address='localhost'
   #port = 5432
   ```

   to:

   ```
   listen_address='*'
   port = 5432
   ```

   Open the `pg_hba.conf` file in vi:
   `vi /usr/local/postgresql/data/pg_hba.conf`
   Add the following line to the file:
   `host all all 0.0.0.0/0 trust`
10. Restart postgresql:
    `pg_ctl -D /usr/local/postgresql/data -l logfile restart`
11. Change the password for user postgres in the PostgreSQL database:
    ```
    psql
    ALTER USER postgres WITH PASSWORD 'password';
    \q
    ```
    If the postgresql service is not started, run the following commands:

```
su postgres
vi ~/.bashrc
```
Add /usr/local/pgsql/bin/ to the file:
```
export PATH=/usr/local/cuda-8.0/bin:$PATH:/usr/local/pgsql/bin/
```
Run the following command:
```
source ~/.bashrc
```

12. Create the database schema in PostgreSQL. Run the following command on the psql console:
```
create database edge with owner postgres encoding='UTF-8'
lc_collate='en_US.utf8' lc_ctype='en_US.utf8' template template0;
```
In the database, create the following tables:

```
CREATE TABLE vi_tenant_inspectionresult(id text, info jsonb);
CREATE TABLE vi_tenant_notification(id text, info jsonb);
CREATE TABLE vi_tenant_defectsummary(id text, info jsonb);
CREATE TABLE vi_tenant_uploaddataset(id text, info jsonb);
CREATE TABLE vi_tenant_syncprocess(id text, info jsonb);
CREATE TABLE vi_tenant_model(id text, info jsonb);
CREATE TABLE vi_tenant_datagroup(id text, info jsonb);
```

where *tenant* is the tenant for the operation user in the Maximo PQI On-Premises Visual Insights center. You can get the tenant value from the user profile in the user interface for the center application.

Installing Python modules
To create a training server or stand-alone edge, you must install the required Python module.

Procedure

Install the Python module by using the following commands:
```
sudo apt-get update
sudo apt -y install postgresql
sudo apt -y install libpq-dev
sudo pip install PyGreSQL
sudo pip install DBUtils
```

Installing the stand-alone edge
For the stand-alone edge, you must install the edge onto the edge machine manually.

Downloading the build file and installation script
You must download the stand-alone edge build file and installation script to install a stand-alone edge.

Procedure

1. In the Maximo PQI On-Premises Visual Insights application, select **Data** > **Edges**. Select the master edge and view the edge details.
2. Select and download the build file for the required operating system. The following operating systems are supported: Ubuntu, Red Hat Enterprise Linux, and IBM Power System.
3. Select and download the installation shell script file.

Installing the build file on the stand-alone edge
To install the edge system on the edge machine, you must use an SSH user name and perform an SSH login to the edge machine.

Procedure

1. After the SSH login, switch the working directory to the deployment path of the edge creation dialog.
2. Add with execution permission to the shell script file using the following command:
```
chmod +x ./edgeDeployed.sh
```
3. Run the shell script file to install the stand-alone edge. Use the following command:
```
./edgeDeployed.sh
```

4. Update the *deploy_path*/vi_edge-bin_vi/vi_edge/postgres/DBConfig.json file with your PostgreSQL information to correct the database connection. Then run the following commands to restart the edge:
   cd *deploy-path*/vi_edge-bin_vi/vi_edge/
   ./restartController.sh

Checking the status of services on the edge systems

After you register the edge to the center application, verify that the correct Python processes are running on the edge systems.

Procedure

1. After the edge is installed on the edge system, perform an SSH login to the edge system.
2. On the master edge system, run the following command to look for the Python process for the master engine:
   ps aux | grep python
   The result should include the following Python process:
   python *deployment_folder*/vi_edge-bin_vi/vi_task_manager/run.py.
   If you do not find the Python process on the master edge system, check the log files at *deployment_folder*/vi_edge-bin_vi/vi_task_manager/master.log.
   Try to start the master engine by running the following command:
   deployment_folder/vi_edge-bin_vi/vi_task_manager/restartMaster.sh *IP_of_master_node*
3. On the master edge system, run the following command to look for the Python process for the edge controller:
   ps aux | grep python
   The result should include the following Python processes:
   python *deployment_folder*/vi_edge-bin_vi/vi_edge/runMonitor.py
   python *deployment_folder*/vi_edge-bin_vi/vi_edge/runService.py.
   If you do not find the runMonitor Python process on the master edge system, check the log files at *deployment_folder*/vi_edge-bin_vi/vi_edge/Event.log.
   If you do not find the runService Python process on the master edge system, check the log files at *deployment_folder*/vi_edge-bin_vi/vi_edge/Service.log.
   Try to start the edge controller by running the following command:
   deployment_folder/vi_edge-bin_vi/vi_edge/restartController.sh
4. Run the following command on the master edge system and the slave edge systems to look for Python processes for the score engine of the classification model:
   ps aux | grep python
   The result should include the following Python processes:
   python *deployment_folder*/vi_edge-bin_vi/vi_score_engine_restful/front_run.py 5005 6005
   python *deployment_folder*/vi_edge-bin_vi/vi_score_engine_restful/back_run.py 6005
   If you do not find the Python process on the edge systems, check the log files:
   deployment_folder/vi_edge-bin_vi/vi_score_engine_restful/front_log.txt
   deployment_folder/vi_edge-bin_vi/vi_score_engine_restful/back_log.txt
   After you resolve any issues, the master engine should start the score engine automatically. To start the score engine manually, run the following commands:
   source ~/.bashrc
   deployment_folder/vi_edge-bin_vi/vi_score_engine_restful/restartEngine.sh
5. Run the following command on the master edge system and the slave edge systems to look for the Python process for the score engine of the object detection model:
   ps aux | grep python
   The result should include the following Python processes:
   python deployment_folder/vi_edge-bin_vi/vi_obj_detection/RESTAPI/model/run.py port gpuid FRCNN

```
python deployment_folder/vi_edge-bin_vi/vi_obj_detection/RESTAPI/model/
run.py port gpuid SSD
```
If you do not find the Python processes on the edge systems, check the following log files:
deployment_folder/vi_edge-bin_vi/vi_obj_detection/RESTAPI/model/FRCNN_*gpuid*.log
deployment_folder/vi_edge-bin_vi/vi_obj_detection/RESTAPI/model/SSD_*gpuid*.log
If you see a "No module named iotmyolo" error message, copy the `iotmyolo.so` or `iotmyolo.py` file from the `/home/user/vi_edge-bin_vi/vi_obj_detection/model_library/yolo` directory to the YOLO_HOME directory.
After you resolve any issues, the master engine should start the score engine automatically.

6. Rebuild the YOLO library with the edge component:
   a. Go to the `/home/user/vi_edge-bin_vi/vi_obj_detection/model_library/yolo` directory.
   b. Add the `detectorobj.c` file to the `darknet/examples` folder.
   c. Edit the `darknet/Makefile` file and indicate that EXECOBJA=`detectorobj.o`. The following code is an example of the code in the Makefile file:

```
EXECOBJA=detectorobj.o captcha.o lsd.o super.o voxel.o art.o tag.o cifar.o go.o rnn.o
rnn_vid.o compare.o segmenter.o regressor.o classifier.o coco.o dice.o yolo.o
detector.o writing.o nightmare.o swag.o darknet.o
```

   d. In the `Makefile` file, add `$(EXECOBJ)` for the `$(SLIB)` and `$(ALIB)` objects. The following code is an example of the code in the `Makefile` file:

```
$(ALIB): $(EXECOBJ) $(OBJS)
     $(AR) $(ARFLAGS) $@ $^
$(SLIB): $(EXECOBJ) $(OBJS)
     $(CC) $(CFLAGS) -shared $^ -o $@ $(LDFLAGS)
```

   e. Run the following command:
   `make`

Upgrading edge systems

Upgrade your edges if they are at an older version. You can download the edge build file and installation script to upgrade the edge.

About this task

If you upgrade the Maximo PQI On-Premises Visual Insights center application but do not upgrade the edge, this causes an inconsistency between the center and edge, and models might fail to deploy. You might see an error message similar to the following: "The edge needs to be upgraded before deploying new model."

Procedure

1. Run the following command to get the ID of the edge:
   `/ibm/iotm/vi/service/edge?user=xxx&solution=vi`
2. For the installation shell script, call the following service:
   `/ibm/iotm/vi/service/edgeFile?edgeId=xxx&version=shell&user=xxx&solution=vi`
   Replace `edgeId=xxx` with the edge ID you got from the previous step. Save the file as `edgeDeployed.sh`. Here is a sample command:
   `curl -k -H "apikey:yourapikey" "https://iotm.predictivesolutionsapps.ibmcloud.com/ibm/iotm/vi/service/edgeFile?edgeId=1508123458000&version=shell&user=youruser&solution=vi" > edgeDeployed.sh`

3. Check the edgeDeployed.sh file. Check the values for username, which is the SSH user that is used to start the edge services, password, which is the password of the SSH user, and basefolder, which is the deployment path of edge services. Input the value for <to edit>.
4. For the edge build file, call the following service:
   /ibm/iotm/vi/service/edgeFile?edgeId=*xxx*&version=*xxx*&user=*xxx*
   &solution=vi
   where version is ubuntu, redhat, or power depending on the system. Save the file as vi_edge-bin_vi.zip. Here is a sample command:
   curl -k -H "apikey:*yourapikey*" -o vi_edge-bin_vi.zip
   "https://iotm.predictivesolutionsapps.ibmcloud.com/ibm/iotm/vi/service/edgeFile?edgeId=1508123458000&version=ubuntu&user=*youruser*&solution=vi"
5. Put the edgeDeployed.sh and vi_edge-bin_vi.zip files on the edge system.
6. Log in to the edge as the SSH user that was used when creating the edge in the center application.
7. Put the edgeDeployed.sh and vi_edge-bin_vi.zip files in the deployPath directory.
8. Shut down the current edge service.
9. Run the following commands:
   dos2unix edgeDeployed.sh
   chmod +x edgeDeployed.sh
   ./edgeDeployed.sh
10. Check to see whether the Python service for the edge has started.

Related tasks

Checking the status of services on the edge systems

After you register the edge to the center application, verify that the correct Python processes are running on the edge systems.

Chapter 5. Creating and using models

You create models to collect historical images and defect information. The information is used to train the model. After the model is trained, it must be validated before it is deployed to an edge. Validating the model provides model accuracy information. There can be multiple versions of a model. Models can share defect information but have different image files from different product lines. You can retrain a model to attempt to get a higher model accuracy so that the model version can be replaced with a newer model version. There are two types of models implementations: the classification model and the object detection model.

Structure of compressed image files

Before you add historical images for image groups, you must have files that contain the image files that you need for either the classification model or the object detection model.

Classification model

Add the images into compressed files. One compressed file must contain all of the images that belong to the same image group. You must put all of the images in a flat structure with no subfolders in the compressed file. The following image types are supported: PNG, JPEG, and JPG. Image files must all have uppercase file extensions, for example, PNG, JPEG, JPG, or they all must have lowercase file extensions, for example, png, jpeg, jpg.

Object detection model

The structure for the object detection model must contain two folders in one compressed file. One folder must be named `JPEGImages` and the other must be named `Annotations`. In addition to the two folders, the compressed file must also contain a `labels.txt` file.

Add all of the image files to the `JPEGImages` folder. The following image types are supported: PNG, JPEG, and JPG. Image files must all have uppercase file extensions, for example, PNG, JPEG, JPG, or they all must have lowercase file extensions, for example, png, jpeg, jpg. Add all of the annotations files to the `Annotations` folder. An annotation file must have the same file name as its image file. The files must be in XML format. If you use the image labeling tool in Maximo PQI On-Premises Visual Insights, the output is a compressed image file that contains the images file and the annotation file. If you use an external image labeling tool, you must make sure that the images file and the annotation file are in the expected structure. The following information is an example of an annotation file:

```
<annotation>
  <folder>JPEGImages</folder>
  <filename>000001.jpg</filename>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>864</width>
    <height>1296</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>defect1</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>474</xmin>
      <ymin>368</ymin>
      <xmax>540</xmax>
      <ymax>448</ymax>
    </bndbox>
  </object>
  <object>
```

```
    <name>defect2</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>303</xmin>
      <ymin>387</ymin>
      <xmax>369</xmax>
      <ymax>452</ymax>
    </bndbox>
  </object>
</annotation>
```

The `labels.txt` file contains the names of all of the defects types that are in the annotation folder. Each defect must be on a separate line, as shown in the following example:

```
defect1
defect2
defect3
```

Adding historical images for image groups

The model manager uses historical images to train the model.

About this task

You can upload compressed (.zip) image files to the image group. Prepare the compressed image files with correct file size. If a file is too big, you can split the file into several files. Based on your network upload speed to the application, make sure each compressed image file can be uploaded within 30 minutes. For example, if your network upload speed to the application is 100 KB/s, the maximum size for a compressed image file is 200 MB.

Procedure

1. Select **Data** > **Image groups** > **New Image Group**. Select **New unlabeled image group** if you want to use the image labeling tool in Maximo PQI On-Premises Visual Insights to label the images. Select **New labeled image group** if the images were already labeled by an external tool.
2. Add a unique image group name and description, select the image group type, and select **Next**.

   For the image group type:

   - Single characteristics means that the images in the group belong to one defect type. Single characteristic uses the classification model.
   - Not a defect means single characteristics where the images in the group do not have a defect. Not a defect uses the classification model.
   - Multiple characteristics means that the images each contain one or more defects that can belong to the same or different defect types. Multiple characteristic uses the object detection model.

   Note: After the image set is uploaded, the image group type that you select cannot be changed.
3. In the **Image sets** pane, add images and select **Add Image Group**.

Results

In the **Image groups** pane, you can select an image group and select edit. You can add or delete the image sets, update the image group name or description, or change the image group type.

Note: If the image set is cited in a model instance, the image set cannot be deleted.

Image labeling tool

The image labeling tool can be used to semi-automatically label images. You can upload unlabeled image files, manually label some of the images, and then trigger the automatic labeling to label the unlabeled

images. You can check the result of the automatic labeling. Based on the summary information, you can accept the labeling result. Images are labeled and packaged to corresponding image zip files and automatically attached to image groups so that they can be used in the model training process.

Creating an unlabeled image group

You can upload compressed (.zip) image files that contain unlabeled images to create a new unlabeled image group.

About this task

When creating an unlabeled image group, you can use a compressed file that contains no labeled images or a compressed file that contains both labeled and unlabeled images. For single characteristics files, any labeled images need to be contained in a subfolder where the name of the folder is the defect name. For multiple characteristics files, the labeled image file must have a corresponding annotation XML file in the same folder.

Prepare the compressed image files with correct file size. If a file is too big, you can split the file into several files. Based on your network upload speed to the application, make sure each compressed image file can be uploaded within 30 minutes. For example, if your network upload speed to the application is 100 KB/s, the maximum size for a compressed image file is 200 MB.

Procedure

1. Select **Data** > **New Image Group** > **New Unlabeled Image Group**.
2. In the **New Unlabeled Image Group** dialog box, specify a name for the image group.
3. Specify the image group type:
   - Single characteristics means the images in the group have only one defect type.
   - Multiple characteristics means the images in the group contain one or more defect types.
4. Click the upload link to upload a compressed file that contains unlabeled images.
5. Click **Add Image Group** to create the new unlabeled image group.
6. In the **Sampling rules** dialog box, set the ratio of images to be manually labeled and then click **Confirm**.

Results

After you complete this task, you can begin to manually label the sample images.

Manually labeling sample images

You can manually label images that are contained in an unlabeled image group.

Procedure

1. In the **All Unlabeled Image Groups** window, click **View** next to the group that contains the images that you want to label.
2. Click the sampling rules icon to set the sampling rate, which indicates the number of images that you want to manually label. Click **Confirm**.
3. For each sample image, manually label the image by selecting the corresponding label and then save the image. You can choose existing labels or create new labels. For single characteristics groups, one label corresponds to one data group. For multiple characteristics groups, one data group with same group name is created. All labels used in the images in the unlabeled group are saved as tags in the corresponding data group.
4. After all sample images are labeled, you can click **Run auto-label** to start the automatic labeling process.

Automatically labeling sample images

Maximo PQI On-Premises Visual Insights uses convolutional neural network-based deep learning techniques to automatically label images.

Procedure

1. In the **All Unlabeled Image Groups** window, click **View** next to the group that contains the images that you want to label.
2. Click **Run auto-label** to start the automatic labeling process. You can click **Running in the background** to run the automatic labeling process in the background.
3. After the automatic labeling process completes, click **View** to review the results.
4. For each automatically labeled image, confirm or modify the label.
5. If necessary, re-run the automatic labeling process.
6. Click **View summary** to view label and model accuracy information.
7. When you are satisfied with the model accuracy, click **Accept**.
   All labeled images are saved as a labeled compressed file. You can find the file in the **Data** pane.

Model creation

To get good model accuracy, you need to set hyper parameters carefully based on your image sets and selected model type.

When you create a model, you can choose the recommended model settings or choose to customize them. The recommended model settings are not appropriate for all scenarios. They are just one reference. You might need to customize the model settings.

When you train a model, you need to know how many images are in your image sets. Some images are used as training data sets and others are used as validation data sets. The ratio depends on the training and validation sampling rule. By default, 80% of images are used for training and 20% of images are used for validation. You also need to know which batch size is used in your model definition. Based on the image number and batch size, you can calculate how many iterations are needed to go through all the images. This is called an epoch. Then you can evaluate the maximum iteration number and, for some models, the validation interval and validation iteration.

If you select the classification model type, the batch size is predefined in the network definition. For GoogLeNet, the train batch size is 32 and the test batch size is 16. For AlexNet, the train batch size is 128 and the test batch size is 32. For LeNet, the train batch size is 64 and the test batch size is 32. You can adjust hyper parameters based on the batch size, total image number, and sampling rule.

Creating models

After the defect types are added, the model manager creates a model. The model can be saved as a draft or trained immediately.

About this task

You view the details of a model on the **All Models** tab. The model details include the versions of the model. Different model versions are built by different image sets.

Model names must be unique.

Procedure

1. Select **Create New Model** > **Create New Model**.
2. On the **General** tab, update the information.

The product type is used to map the model. You choose the product type outside of Maximo PQI On-Premises Visual Insights based on your own company standards. The product type is the link between the model and the production line. The edge controller selects the model based on the product type information that is on the image.

3. On the **Model type** tab, update the model type information. For **Model type**, you can specify classification or multi-object detection, or you can choose to upload an existing trained model. If you select classification or multi-object detection, you can select the recommended model settings or choose to customize them.

   If you choose to customize the model settings, specify the detection method, hyper parameters, and sampling rules. Each hyper parameter has a valid value scope. If you enter a value outside of that scope, an error message displays. Due to resource limitations, there is a maximum value for epoch, iteration and batch size. The maximum value of iteration is related to the payment type of the tenant. For paid users, the maximum value of iteration is 40,000 for FRCNN, 20,000 for YOLO, or 5,000 for SSD. For trial users, the maximum value of iteration is 3,000 for FRCNN, 2,000 for YOLO, or 500 for SSD. The batch size of YOLO and SSD is 32. For CNN, the maximum epoch value is 30,000 for paid users and 5,000 for trial users.

   If you upload an existing trained model compressed file, the model setting values are read from the file and displayed in the dialog box. If the epoch or iteration value and batch size exceed the maximum value, the values are replaced with the maximum values.
4. On the **Images sets** tab, select the image sets that you want to use to train the model.
5. On the **Global Policies** tab, set the retrain policy, sampling policy, and manual inspection policy settings.
6. On the **Summary** tab, select **Save as draft** to save the model, or **Train** to train the model.
7. In the **All models** pane, view the model. If the model version is set to Draft, you can click **View** to view the model instance details. You can edit the image groups before you train the model.

Using the model catalog

The model catalog is a library of shared models that can be reused for industry solutions and shared by tenants.

The model catalog contains the following models.

Domain	Model Name	Description
Electronics - Hardware Manufacturing	Server Hardware Defect Inspection	This object detection model detects missing parts for IBM servers. It supports seven major defects including missingChinLabel, missingPowerLabel, missingIBMLogo, missingDASD, incorrectDASD, missingCover, and missingPowerButton.
Automotive - Car Wearout	Car Wearout Defect Inspection	This object detection model detects car wear out situations. Automobile manufacturers test tens of thousands of open and close activities and detect when wear out is at unacceptable level. This model is focused on detecting paint scratches.

Domain	Model Name	Description
Automotive - Car Seat	Car Seat Defect Inspection	This object detection model detects wrinkle defects in car seats. The inspection result identifies whether there is wrinkle defect on a car seat and localizes the defect on the image.

Testing models in the model catalog

You can test models in the model catalog by uploading your own image and checking the scoring result. You can upload multiple images and check the scoring result one by one.

Procedure

1. Select **Create New Model** > **Browse Model Catalog**.
2. On the **Model Catalog** page, select the shared model that you want to use.
3. Click **Upload more** to upload an image. The model scores the image and displays the result.

Creating models by using the model catalog

You can create new models by using shared models in the model catalog. To use a shared model, import the model definition and use your own data sets to train your model.

Procedure

1. Select **Create New Model** > **Browse Model Catalog**.
2. On the **Shared Models** page, select the shared model that you want to use.
3. Click **Use Model**.
4. On the **General** tab, update the information. The product type is used to map the model.
5. On the **Model type** tab, update the model type and model setting information.
6. On the **Image sets** tab, select the image sets that you want to use to train the model.
7. On the **Global Policies** tab, update the retrain policy and manual inspection policy if needed.
8. On the **Summary** tab, click **Save as draft**. The model status is set to Draft.

Training models

Maximo PQI On-Premises Visual Insights uses convolutional neural network-based deep learning techniques to train models. When you train a model, the model learns the features of labeled images and builds an executable model that embeds this knowledge. A trained model is able to perform classification or object detection on images from the product line.

About this task

You can train a newly created model or a model that was created using an existing model in the model catalog. You can train models in draft, failed, or rejected status. During training, you can inspect the training status and log files. After training is completed, you can check the training curve including the train loss, test loss, and test accuracy values.

Procedure

1. In the **All Models** pane, select a model.
2. Click **View** to view the model instance details.
3. Optional: Change the image groups that are used by the model.
4. Click **Train** to start the training.

The training status changes to Waiting as the data is prepared and the training job is queued on the server. When training begins, the status changes to Training. When training completes, the status changes to Trained.

5. After training is completed, review the training results in the **Train dashboard**. If the model supports snapshots, the snapshots are displayed in the dashboard. You can check the information for each snapshot. You can use a snapshot by selecting the snapshot and then clicking **Use**. You can select an intermediate snapshot to use as the final model file. You can also view and download the training log files.

Trained models

After a model is created or edited, the model manager can train the model. After the model is trained, the model status changes to Trained. The model manager validates the trained model and either accepts or rejects the model. The model manager validates the model by using validation image sets.

If the model manager chooses to upload the trained model when creating a new model, the model status changes to Accepted after the upload process.

Structure of model files

Maximo PQI On-Premises Visual Insights supports the convolutional neural network (CNN) classification and object detection model types.

CNN classification models for Caffe

The CNN classification model must be a single compressed file and contain the correct directory structure and files.

The compressed model file

The compressed model file must contain the following directories and files.

- `model.config` (file, required)
- `sink.config` (file, required)
- `parameter.config` (file, optional)
- `cnet1` (directory). The `cnet1` directory must contain the following files:
  - `labels.txt` (file, required)
  - `deploy.prototxt` (file, required)
  - `mean.binaryproto` (file, required)
  - `info.json` (file, optional)
  - `snapshot.caffemodel` (file, required)
  - `solver.prototxt` (file, required)
  - `train_val.prototxt` (file, required)

Each file must have the correct structure and keywords. The files are described in the following sections.

model.config

The following text is an example of the contents of the `model.config` file. Keywords are shown in bold text.

```
submodel{
  module {
       type:"ChipROIExtractor"
       ref_file:"parameter.config"
  }
  module {
       type:"ClassificationNet"
       net_name:"cnet1"
```

```
  }
}
sink{
    type:"SinkFilter"
    ref_file:"sink.config"
}
```

This file must have at least one module that has the **ClassificationNet** type in the submodel. The ref_file keyword points to the other configuration files within the CNN classification model compressed file. The net_name keyword refers to the folder name that contains the CNN model. You do not need to change the sink information unless you have a different name for the `sink.config` file.

sink.config

The contents of the `sink.config` file are as follows. You do not need to edit the contents.

```
keyword:"position"
keyword:"probableTypes"
```

labels.txt

The `labels.txt` file contains all the class names with which this classification model is classified. Each class name must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

Other files

All *filename*`.prototxt` files are model definition files that are required when you train a CNN model.

The `snapshot.caffemodel` and `mean.binaryproto` files are output files that are created after the model training is complete.

Object detection model

The object detection model must be a single compressed file and contain the correct directory structure and files. The following object detection algorithms are supported: faster region-based convolutional neural network (Faster R-CNN), You Only Look Once (YOLO) V2, and single shot multibox detector (SSD).

Faster R-CNN

The Faster R-CNN object detection model must be a single compressed file and contain the correct directory structure and files.

Compressed model file

The compressed model file must contain all of the following files:

- `labels.txt`
- `faster_rcnn_final.caffemodel`
- `model.config`
- `stage1_fast_rcnn_solver30k40k.pt`
- `stage1_fast_rcnn_train.pt`
- `stage1_rpn_solver60k80k.pt`
- `stage1_rpn_train.pt`
- `stage2_fast_rcnn_solver30k40k.pt`
- `stage2_fast_rcnn_train.pt`
- `stage2_rpn_solver60k80k.pt`
- `stage2_rpn_train.pt`

- faster_rcnn_test.pt
- rpn_test.pt

Each file must have the correct structure and keywords. The files are described in the following sections.

labels.txt

The labels.txt file contains the names of all of the objects that this object detection model detects. Each object must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

faster_rcnn_final.caffemodel

This file contains the output after the Faster R-CNN model is trained. The file name must match the definition of **model** in model.config. Currently, two networks are supported for Faster R-CNN: ZF and VGG16. For ZF-net, the name should start with ZF, for example ZF_mobile_final.caffemodel. For VGG16-net, the name should start with VGG16, for example, VGG16_mobile_final.caffemodel.

model.config

Keywords in the following example are shown in bold text:

```
{
    "modelType": "FRCNN",
    "model": "faster_rcnn_final.caffemodel",
    "solvers":
"stage1_rpn_solver60k80k.pt,stage1_fast_rcnn_solver30k40k.pt,stage2_rpn_solver60k80k.pt,stage2_f
ast_rcnn_solver30k40k.pt",
    "net_file":
"stage1_rpn_train.pt,stage1_fast_rcnn_train.pt,stage2_rpn_train.pt,stage2_fast_rcnn_train.pt",
    "deploy_net": "faster_rcnn_test.pt",
    "parameters": {
        "iteration": "40000,80000,40000,80000",
        "learningRate": 0.001,
        "stepsize": "10000",
        "gamma": "0.1"
    }
}
```

The value of modelType is always FRCNN. The value of model is the name of the model file. The value of solvers is the list of solver files that are used during model training. The value of net_file is the list of network definition files. The value of deploy_net is the name of the scoring network definition. The values in parameters are all hyper-parameters of the Faster R-CNN model. Iteration is a string value composed of four numbers separated by commas, presenting the iteration number of four phases in the training process.

*.pt files

The files with the .pt extension are model definition files. The pascol_voc model that is provided by Faster R-CNN is supported. The template file can be found in the models/pascal_voc/*netname*/faster_rcnn_alt_opt/ Faster R-CNN installation directory, where *netname* is ZF or VGG16.

YOLO V2

The YOLO V2 object detection model must be a single compressed file and contain the correct directory structure and files.

Compressed model file

The compressed model file must contain the following files:

- labels.txt
- model.config

- `yolo_final.weights`
- `Yolo.cfg`

Each file must have the correct structure and keywords. The files are described in the following sections.

labels.txt

The `labels.txt` file contains the names of all of the objects that this object detection model detects. Each object must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

model.config

Keywords in the following example are shown in bold text:

```
{
  "modelType": "YOLO",
  "modelCfg": "Yolo.cfg",
  "model": "yolo_final.weights",
  "parameters": {
    "iteration": "40000",
    "batchSize": 16,
    "learningRate": 0.001,
    "subBatchSize": 2,
    "steps": "100,15000,25000,35000",
    "scales": "1,10,0.1,0.1"
  }
}
```

The value of modelType is always YOLO. The value of modelCfg is the name of the deep learning network definition file. The value of model is the name of the actual model file. The values in parameters are hyper-parameters of the YOLO V2 model. The value of batchSize in parameters cannot be larger than 32.

yolo_final.weights

This file contains the output after the YOLO model is trained. The file name must match the definition of **model** in `model.config`.

Yolo.cfg

This model definition file includes network definitions, hyper-parameters, and anchor settings. A template of this file can be found in the `darknet/cfg/` YOLO installation directory. This file must match the weights file.

SSD

The SSD (Single Shot MultiBox Detector) object detection model must be a single compressed file and contain the correct directory structure and files.

Compressed model file

The compressed model file must contain all of the following files:

- `labels.txt`
- `solver.prototxt`
- `deploy.prototxt`
- `model.config`
- `SSD.caffemodel`

Each file must have the correct structure and keywords. The files are described in the following sections.

labels.txt

The `labels.txt` file contains the names of all of the objects that this object detection model detects. Each object must be on a separate line, as shown in the following example.

```
defect1
defect2
defect3
```

solver.prototxt

This file contains all hyper-parameters for the SSD model.

deploy.prototxt

This file contains the network definition of the trained model.

model.config

Keywords in the following example are shown in bold text:

```
{"modelType": "SSD",
"parameters": {
    "learningRate": 0.01,
    "iteration": 10000,
    "steps": "6000,8000",
    "batchSize": 16,
    "learningRatePolicy": "multistep",
    "display": 10,
    "snapshot": 1000
}
}
```

The value of modelType is always SSD. The values in parameters are hyper-parameters of the SSD model.

SSD.caffemodel
This file contains the output after the SSD model is trained.

Validated models

After a model is trained, the status of the model changes to Trained. The model manager can validate the model version and either accept or reject the trained model. The model manager validates the model by using validation image sets. After a model is validated, it can be used and deployed.

You can validate a model version that has a status of Trained. A trained or retrained model version can trigger the validation process. When you validate a report, you must manage the image sets. Every image group must have at least one validation image set to validate the model. You must use different image sets and training image sets to validate the model version. To begin the validation process, select **Validate**.

After the validation process is finished, a report is generated that shows the model accuracy. You can create the following types of reports:

Classification model report
In the classification model report, one image has one defect at most. The confusion matrix is used to generate the report where each column represents one real image group type in the validation data sets. Each row represents the predicted image group type. The last row in the chart represents the aggregate results.

Object detection model report
In the object detection model report, one image has multiple defects. In this report, the mean average precision and the recall are calculated to indicate the model accuracy.

To view the report, select **Show Report**. From the **Validation Report** window, you can revalidate, reject, or accept and deploy the model.

If the first model version is rejected, you can change the training parameters and then retrain the model. If a model version that is not the first version is rejected, you can retrain a new model version by using new image sets.

Distributing trained models to edges

After the model manager accepts the trained model version, the model is distributed to edges so that it can be inspected.

Procedure

1. Select a model version with a status of Validated. The first time you deploy a model, click **View Report** to view the validation report.
2. Click **Accept and Publish** on the Validation Report page.
3. A Publish and deploy dialog box displays. Select the edges you want to deploy the model to, and then click **Publish and deploy** to deploy the model.
4. If the model has been deployed on some edges, click **Manage deployment** to open the deployment dialog box. Select the edges that you want to deploy on or undeploy on, and then click **Deploy** or **Undeploy**.

Retraining models

When you retrain a model, a new model version is created. When you create a model request, you define the retrain policy. The retrain policy is the condition that triggers auto-retrain. If there is concern about the current model accuracy, the model manager can manually select the image files to trigger the retrain process.

Procedure

1. Select a model version that has a status of Deployed and select **Retrain**.
2. Manage the image sets. Every image group must have at least one retrain image set to retrain the model.

   Note: To retrain the model version, it is best to use different image sets that contain more images.
3. Select **Retrain** to begin the retrain process.

What to do next

The model manager validates if the retrained model is accepted. The model manager can deploy the new version, and the old model version is no longer deployed.

Using models with a stand-alone edge

You use models with a stand-alone edge to score images in the production line, sync data to the center, and to clean uploaded data. To use models with a stand-alone edge, you must publish the models and deploy the models to the stand-alone edge.

Publishing models

Before you manually deploy a stand-alone edge, you must publish the model on the center application.

About this task

To publish a model, it must already be validated.

Procedure

1. Log in to the user interface for the center application, and select a model instance.
2. Select **View Report** to view the validation report.
3. Select **Accept and publish** to publish the model instance.

Results

After you publish the model, the status of the model will change to Ready for deploy.

Deploying one model instance

Use the API to deploy one model instance on the stand-alone edge.

Procedure

1. Get all published models on the center application by using the following command:
   `GET https://<edge machine host>:8449/api/getAvailableModels`
2. Deploy one model instance on the stand-alone edge by using the following command:
   `POST https://<edge machine host>:8449/api/deployModel`
   Use the body that is returned from the getAvailableModels API.

What to do next

For more information about API details, go to the Stand-alone edge services topics.

Undeploying models

Use the API to undeploy models on the stand-alone edge.

Procedure

Undeploy models by using the following command:
`POST https://<edge machine host>:8449/api/undeployModel`
Use the following body:

```
{"model_id":"model_id","model_instance_id":"model_instance_id"}
```

What to do next

For more information about API details, go to the Stand-alone edge services topics.

Chapter 6. Checking inspection results

After the inspection results are sent to the center application, the inspector and the inspector supervisor can go to the Defect Check tab to view and filter the inspection results and make any necessary changes.

Images

The inspector and the inspector supervisor can view images to see if they are classified as existing defects or not, and to find out if someone else checked the images. Viewing the images determines what the inspector or supervisor must do when they check the defects.

The inspector views unchecked and checked images. Unchecked images mean that the image was only scored by the model and was not checked by an inspector. Checked images mean that the image was scored by the model and was already checked by an inspector.

The inspector views unconfirmed and confirmed images. Unconfirmed images mean that the image was only scored by the model and was not confirmed by an inspector. Confirmed images mean that the image was scored by the model and was confirmed by an inspector.

The inspector supervisor can view objects and unknown objects. Unknown objects mean that the image was marked by an inspector as an unknown defect because the inspector did not classify the image as an existing defect. These images are highlighted on the inspector supervisor's list.

Filtering defects

The inspector and the inspector supervisor can apply filters to the cell overview and the defect list.

Procedure

1. On the **All Workstations** window, select a workstation to view the list of unconfirmed, confirmed, and unknown objects.
2. Select the filter icon.
3. Input a value for a condition to set the filter and then select the add icon. The filter applies to the list immediately.

Checking defects

The inspector and inspector supervisor review the inspection results and make any necessary changes.

About this task

When you select an image, the defect candidate and corresponding confidence display. The first defect is selected by default. The inspector can select unknown for a defect type if the defect type is unknown.

Procedure

1. Select an image to view the image details and inspection results.
2. Select **Edit Zoom** to zoom in and out of the image or drag the image to locate a position.
3. Select **Set Zoom** to change back to edit mode. You can add, resize, move, and view the details of a defect box.
4. Select the defect box that you want to confirm and view the details of the defect type and confidence level. You can change the defect type or delete the position.
5. Select **Confirm**.

6. If the image does not belong to any existing defects, the inspector supervisor can create a new image group. The new defect is added into the candidate list for images that are under the same model.

Uploading images by using the simulator

You can use the simulator to manually send images to an edge. This process simulates sending images from the product line to the edge. You can send your own images or use predefined images on the server.

About this task

Procedure

1. Select **Simulator** from the main menu.
2. Specify a product type and cell.
3. To upload your own image, select **Use my own image** and browse to your image. Or, to use images from the server, clear **Use my own image** and specify the total image number and sending frequency.
4. Click **Start**.
5. Click **View analysis result**.

Chapter 7. KPI dashboard

The inspector supervisor uses the KPI dashboard to check the image level defect rate and the location level defect rate. These metrics can help provide information so that you can ask the IT team to retrain the model or adjust the manufacturing procedure.

The KPI dashboard is on the **KPI** tab. You can select all workstations or a specific workstation. This selection impacts the scope that you are working on. You can also switch between real-time and historical views. In the real-time view, KPI data is refreshed every 5 seconds. The KPI values include defect per unit and defect rate. The defect per unit is calculated as a specific defect number divided by the total image number. The defect per unit value represents the occurrence rate of a defect type. The defect rate is calculated as the number of images with one or more defects divided by the total image number. The defect rate represents the product defect rate. Each line in the chart represents the KPI value in the current 5-second interval. The historical view shows historical KPI data. You can edit the start date and end date to determine the time range. KPIs in the historical view include defect per unit and defect rate.

The historical view has three granularities: hourly, daily, and monthly. In the hourly chart, each point represents 1 hour. For example, a KPI value that is 24 points represents 24 hours. In the daily chart, each point represents one day. For example, a KPI value that is 30 points represents 30 days. In the monthly chart, each point represents one month. For example, a KPI value that is 12 points represents 12 months. There are relationships between the selected time range and granularity. If there are too many points in a granularity for a selected time range, then that granularity is disabled until you shorten the time range. If you want to refresh the chart, you can change time range or click **Refresh**. The historical KPI data is calculated periodically by the server, so there is some time delay based on the configuration. The default period to calculate historical KPI is 1 hour.

By default, only the top five defects types are displayed in the Defect Per Unit chart. If you want to check other defect types, you can select one or more defect types under the chart and then click **Show trend**. A new KPI chart displays the selected defect types. If the KPI value of a defect type is 0, it cannot be selected to show a trend.

Chapter 8. Integration with Prescriptive Quality

You can export historical defect rate data from Maximo PQI On-Premises Visual Insights in CSV format and import the data in Prescriptive Quality.

You can export the data from the KPI dashboard when you view historical defect rates. To view this data, select **Historical** on the KPI dashboard and then select **Defect Rate**. When you select **Export**, the data is exported based on the selected cell, time range, and time granularity.

The following columns are contained in the CSV file.

Table 3. Columns in CSV file

Column	**Description**
ATTRIBUTE_NAME	This value is always VI_DEFECT_IMAGE_RATE.
DATE	The date in yyyy-MM-dd format when the interval is daily or monthly, or yyyy-MM-dd HH:mm:ss format when the interval is hourly.
CELL_NAME	The name of the cell where the defect happened. It is one of the dimensions in Prescriptive Quality.
PRODUCED_QTY	The total number of images in the selected cell and time range.
TESTED_QTY	The total number of images that were tested.
FAILED_QTY	The number of images that show defects

Chapter 9. Application programming interface

The application programming interface enables you to perform actions such as managing data groups, data files, models, inspection results, and edges.

API workflows

Many of the Maximo PQI On-Premises Visual Insights API calls have prerequisites, or correlations with other API calls. Also, only certain user roles are given access to each API call. Therefore, it is important to understand the API workflow for each role.

Model manager

The model manager generally uses the API calls in the following sequence:

- Create a data group. Refer to Create a data group in Data group services.
- Upload a data file to a data group. Refer to Upload data files to a data group in Data file services.
- Create a model. Refer to Create a model in Model services.
- Create a model instance. Refer to Create a model instance in Model instance services.
- Train the model. Refer to Train model instance. The training process takes several minutes to hours depending on the number of images and the training parameters. To check the status of the model instance, refer to Get one model instance.
- After the online training is completed, the model manager can validate the model. Refer to Validate model instance.
- Create an edge. Refer to Create edge in Edge services.
- Deploy the model to the edge or reject the model. Refer to Deploy model instance or Reject model instance in Model instance action services.
- Retrain or undeploy the deployed model. Refer to Retrain model instance or Undeploy model instance in Model instance action services.

Inspector

The inspector generally uses the API calls in the following sequence:

- Score an image. Refer to Score an image in Score service.
- Get the inspection result cell overview. Refer to Get the inspection result cell overview in Inspection result services.
- Get the inspection result list. Refer to Get the inspection result list in Inspection result services.
- Confirm inspection results. Refer to Confirm inspection results in Inspection result services.

Supervisor

The supervisor generally uses the API calls in the following sequence:

- Score an image. Refer to Score image in Score service.
- Get the inspection result cell overview. Refer to Get the inspection result cell overview in Inspection result services.
- Get the inspection result list. Refer to Get the inspection result list in Inspection result services.
- Confirm inspection results. Refer to Confirm inspection results in Inspection result services.
- Get defect image rate file. Refer to Get defect image rate file in QEWS integration service.

Stand-alone edge administrator

The stand-alone edge administrator must have credentials for the stand-alone edge machine. The stand-alone edge administrator generally uses the API calls in the following sequence:

- Get available models. Refer to Get available models in Stand-alone edge services.
- Deploy a model. Refer to Deploy a model in Stand-alone edge services.
- Deploy all models. Refer to Deploy all models in Stand-alone edge services.
- Upload and score an image on the edge. Refer to Upload and score image on the edge in Stand-alone edge services. The production line or external services typically use this API.
- Sync the inspection result from the edge to the center application. Refer to Sync inspection result from the edge to the center application in Stand-alone edge services.
- Clean up the inspection result that you have synced to the center application. Refer to Clean up inspection result that you have synced to the center application in Stand-alone edge services.
- Undeploy a model. Refer to Undeploy a model in Stand-alone edge services.

Preparing to use the API calls

Before you use any of the API calls, ensure that you have prepared the user and the solution and obtained an API key.

Perform the following tasks to prepare the user and the solution:

- Ensure that the user that is logged in has access to the wanted API calls.
- For API calls that require a solution ID, specify `vi` as the solution.
- Ensure that the user has the API key that IBM created for the user.

Service responses

After an API call completes, it provides a success or error response.

The API services provide the following responses.

- Success responses:
  - Status code 201 for create services
  - Status code 200 for other services
- Database error responses:
  - Status code 500
  - Error messages, for example, "The parameters are missing or have invalid format when creating a new data group."
- Authorization errors
  - Status code 401
  - Error messages, for example, "Either authorization or APIKEY need to be set in header."

Data group services

The data group services enable you to perform such tasks as getting all data groups, getting one data group, and creating a new data group.

Get all data groups

Gets all existing data groups. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataGroup

Method

The request type GET

URL parameters

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

APIKEY: *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
  {
    "id": "892d05cd-84e5-4bae-b57f-edbbc3f8c598",
    "groupName": "MisT",
    "createdBy": "auto_mm",
    "description": "test_description",
    "dataFormat": "jpg",
    "parameters": {
      "isHybrid": "false",
      "isDefect": "true"
    },
    "updatedTime": "2017-11-29T16:47:54.039+08:00"
  },
  {
    "id": "9bebc890-0237-4a3b-8908-9f5f53fdb3a2",
    "groupName": "NoDefect",
    "createdBy": "auto_mm",
    "description": "test_description",
    "dataFormat": "jpg",
    "parameters": {
      "isHybrid": "false",
      "isDefect": "true"
    },
    "updatedTime": "2017-11-29T16:47:54.463+08:00"
  },
  {
    "id": "b989aae2-f1ae-4a77-93bb-4e9e4f18264d",
    "groupName": "Defect",
    "createdBy": "auto_mm",
    "description": "test_description",
    "dataFormat": "jpg",
    "parameters": {
      "isHybrid": "false",
      "isDefect": "true"
    },
    "updatedTime": "2017-11-29T16:47:53.536+08:00"
```

```
  }
]
```

Response items

`id: String`. The data group ID.

`groupName: String`. The data group name.

`createdBy: String`. The data group owner.

`description: String`. The data group description.

`dataFormat: String`. The data group format.

`isHybrid: Boolean`. True means object detection. False means classification.

`isDefect: Boolean`. True means is a defect. False means is not a defect.

`updatedTime: Time`. The last time that the data group was updated.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aa
b2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/dataGroup?user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get all data groups with data files

Gets all existing data groups that have data files for each group. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataGroup

Method

The request type GET

URL parameters

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`dataFiles: []`. Used to identify and get data files. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
  {
    "id": "3e420dbc-88fb-463a-83ee-a317688b02e0",
    "groupName": "test",
    "createdBy": "auto_mm",
```

```
    "description": "test_description",
    "dataFormat": "jpg",
    "parameters": {
      "isHybrid": "false",
      "isDefect": "true"
    },
    "dataFiles": [
      {
        "id": "3e420dbc-88fb-463a-83ee-a317688b02e0_539bf24a-afc9
-4f43-9ca9-b7ec0648e781",
        "groupId": "3e420dbc-88fb-463a-83ee-a317688b02e0",
        "createdBy": "auto_mm",
        "fileName": "MisT.zip",
        "fileCount": "3",
        "updatedTime": "1511949371332",
        "fileUrl": "/user/AUTO/VI/datagroup/3e420dbc-88fb-463a-83
ee-a317688b02e0/1511949371332.zip"
      },
      {
        "id": "3e420dbc-88fb-463a-83ee-a317688b02e0_f76b31fb-f803-
4d0e-995e-b597bf3c761d",
        "groupId": "3e420dbc-88fb-463a-83ee-a317688b02e0",
        "createdBy": "auto_mm",
        "fileName": "Defect.zip",
        "fileCount": "3",
        "updatedTime": "1511949368998",
        "fileUrl": "/user/AUTO/VI/datagroup/3e420dbc-88fb-463a-83ee
-a317688b02e0/1511949368998.zip"
      }
    ],
    "updatedTime": "2017-11-29T16:59:43.873+08:00"
  }
]
```

Response items

`id:` *`String`*. The data group ID.

`groupName:` *`String`*. The data group name.

`createdBy:` *`String`*. The data group owner.

`description:` *`String`*. The data description.

`dataFormat:` *`String`*. The data group format.

`isHybrid:` *`Boolean`*. True means object detection. False means classification.

`isDefect:` *`Boolean`*. True means is a defect. False means is not a defect.

`dataFiles:` *`JSONObject`*. Contains information about data files.

`id:` *`String`*. The data file ID.

`groupId:` *`String`*. The data group ID.

`createdBy:` *`String`*. The data file owner.

`fileName:` *`String`*. The file name.

`fileCount:` *`String`*. The file count.

`updatedTime:` <Time>. The last time that the data file was updated.

`fileUrl:`. The file path that is stored in the server.

`updatedTime:` *`Time`* The last time the data group was updated.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d
4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112.229.91:
9447/ibm/iotm/service/dataGroup?user=auto_mm&solution=vi &dataFiles=[]"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get one data group

Gets one data group. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataGroup/*groupId*

Method

The request type GET

URL parameters

`groupId: String`. The data group ID.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
  {
    "tags": "",
    "id": "3e420dbc-88fb-463a-83ee-a317688b02e0",
    "groupName": "test",
    "createdBy": "auto_mm",
    "description": "test_description",
    "dataFormat": "jpg",
    "parameters": {
      "isHybrid": "false",
      "isDefect": "true"
    },
    "updatedTime": "2017-11-29T16:59:43.873+08:00"
  }
]
```

Response items

`tags: String`. The data group tag.

`id: String`. The data group ID.

`groupName: String`. The data group name.

`createdBy: String`. The data group owner.

`description: String`. The data group description.

`dataFormat: String`. The data group format.

`isHybrid: Boolean`. True means object detection. False means classification.

`isDefect: Boolean`. True means is a defect. False means is not a defect.

updatedTime: *Time*. The last time that the data group was updated.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77
d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/dataGroup/3e420dbc-88fb-463a-8
3ee-a317688b02e0?user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Create a data group

Creates a data group. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataGroup

Method

The request type POST

URL parameters

user: *String*. Used to identify the user. Mandatory.

solution: *String*. Used to identify the solution. Mandatory.

tenant: *String*. Used to identify the tenant. Optional.

Headers

Content-type: application/json.

APIKEY: *encrypted key*. The API key, which is used for authentication. Mandatory

Data parameters

groupName: *String*. The data group name.

description: *String*. The data group description.

dataFormat: *String*. The data group format.

isHybrid: *Boolean*. True means object detection. False means classification.

isDefect: *Boolean*. True means is a defect. False means is not a defect.

Sample body

```
[
  {
    "groupName": "test",
    "description": "test_description",
    "parameters": {
      "isHybrid": "false",
      "isDefect": "true"
    }
  }
]
```

Success response

```
201
[
  {
    "id": "7dac9493-4d8c-4472-a4ca-ae450ccaea5d",
    "groupName": "test",
    "tags": null,
    "dataFormat": "jpg",
    "createdBy": "auto_mm",
    "updatedTime": "2017-11-30T09:59:09.597+08:00",
```

```
    "parameters": {
      "isHybrid": "false",
      "isDefect": "true"
    },
    "description": "test_description"
  }
]
```

Response items

`id: String`. The data group ID.

`groupName: String`. The data group name.

`tags: String`. The data group tag.

`createdBy: String`. The data group owner.

`description: String`. The data group description.

`dataFormat: String`. The data group format.

`isHybrid: Boolean`. True means object detection. False means classification.

`isDefect: Boolean`. True means is a defect. False means is not a defect.

`updatedTime: Time`. The last time that the data group was updated.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f05
79e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" -H "Content-Type:
application/json;charset=UTF-8" "https://9.112.229.91:9447/ibm/iotm/service/
dataGroup?user=auto_mm&solution=vi"  --data '[{"groupName":"test","dataFormat":
"jpg", "description": "test_description","parameters":{"isHybrid":"false","isDefect"
:"true"}}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Update a data group

Updates a data group. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataGroup/*groupId*

Method

The request type PUT

URL parameters

`groupId: String`. The date group ID. Mandatory

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`Content-type: application/json`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory

Data parameters

`description: String`. The data group description.

`isHybrid: Boolean`. True means object detection. False means classification.

`isDefect: Boolean`. True means is a defect. False means is not a defect.

Sample body

```
200
[
    {

    "description":"test",

    "parameters":{

    "isDefect":"true",

    "isHybrid": "true" }
    }
]
```

Success response

```
[{}]
```

Response items

None

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533d
d34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" -H "Content-Type:
application/json;charset=UTF-8" -X PUT "https://9.112.229.91:9447/ibm/iotm/service
/dataGroup/7dac9493-4d8c-4472-a4ca-ae450ccaea5d?user=auto_mm&solution=vi"  --data
'[{"description":"test","parameters":{"isHybrid":"true","isDefect":"true"}}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Delete data groups

Deletes existing data groups. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataGroup/*groupId*

Method

The request type DELETE

URL parameters

`groupId: String`. The data group ID. Mandatory

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

```
[{}]
```

Success response

```
200
[{}]
```

Response items

None

Sample call

```
curl -k -X DELETE -H "APIKEY:8b796582
25de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f05
79e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/dataGroup/
ed568556-1462-45e9-9319-4bfa8186d2cd?user=auto_mm&solution=
vi" -d '[{}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Data file services

The data file services enable you to perform such tasks as getting all data files belonging to one data group, getting one data file, and downloading a binary file.

Get all data files that belong to a data group

Gets all data files that belong to a data group. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataFile

Method

The request type GET

URL parameters

`groupId:` *`String`*. The data group ID. Mandatory.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
    {
        "id": "06c870e5-1701-419a-8a84-0f40ee82fe33_a7f1772f-07a0-4d33-9efe
-f2e45a2dc904",
        "groupId": "06c870e5-1701-419a-8a84-0f40ee82fe33",
        "createdBy": "modelmanager1",
        "fileName": "NG.zip",
        "updatedTime": "1502185276752",
```

```
        "fileCount": "52",
        "fileUrl": "/user/T1/VIQ/datagroup/06c870e5-1701-419a-8a84
-0f40ee82fe33/1502185276752.zip"
    },
    {
        "id": "06c870e5-1701-419a-8a84-0f40ee82fe33_b3af214c-b603-43f4-944d-
40277b9a6fd9",
        "groupId": "06c870e5-1701-419a-8a84-0f40ee82fe33",
        "createdBy": "modelmanager1",
        "fileName": "IoT4M_March2017_VI_Source.zip",
        "updatedTime": "1502184389077",
        "fileCount": "52",
        "fileUrl": "/user/T1/VIQ/datagroup/06c870e5-1701-419a-8a84-0f40ee82fe
33/1502184389077.zip"
    },
    {
        "id": "06c870e5-1701-419a-8a84-0f40ee82fe33_c901e409-d9db-4f02-8d9c
-fa6002929701",
        "groupId": "06c870e5-1701-419a-8a84-0f40ee82fe33",
        "createdBy": "modelmanager1",
        "fileName": "NG.zip",
        "updatedTime": "1502185248433",
        "fileCount": "52",
        "fileUrl": "/user/T1/VIQ/datagroup/06c870e5-1701-419a-8a84-0f40ee82fe
33/1502185248433.zip"
    }
]
```

Response items

`id:` *`String`*. The data file ID.

`groupId:` *`String`*. The data group ID.

`createdBy:` *`String`*. The data file owner.

`fileName:` *`String`*. The file name.

`fileCount:` *`String`*. The file count.

`updatedTime:` *`Time`*. The last time that the data file was updated.

`fileUrl:`. The file path that is stored in the server.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0
579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/dataFile?user=auto_mm&solution
=vi&groupId=ed568556-1462-45e9-9319-4bfa8186d2cd"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get one data file

Gets one data file. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataFile

Method

The request type GET

URL parameters

`fileId:` *`String`*. The data file ID. Mandatory.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

APIKEY: *encrypted key*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
    {
        "id": "06c870e5-1701-419a-8a84-0f40ee82fe33_c901e409-d9db-4f02-8d9c
-fa6002929701",
        "groupId": "c901e409-d9db-4f02-8d9c-fa6002929701",
        "createdBy": "modelmanager1",
        "fileName": "NG.zip",
        "updatedTime": "1502185248433",
        "fileCount": "52",
        "fileUrl": "/user/T1/VIQ/datagroup/06c870e5-1701-419a-8a84
-0f40ee82fe33/1502185248433.zip"
    }
]
```

Response items

`id: String`. The data file ID.

`groupId: String`. The data group ID.

`createdBy: String`. The data file owner.

`fileName: String`. The file name.

`updatedTime: Time`. The last time that the data file was updated.

`fileUrl:`. The file path that is stored in the server.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f05
79e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112
.229.91:9447/ibm/iotm/service/dataFile/06c870e5-1701-419a-8a84-0f40ee82fe33_
c901e409-d9db-4f02-8d9c-fa6002929701?user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Download the binary content of the data file

Downloads the binary content of a data file. The model manager and the data scientist have authority to use this API.

URL

/ibm/iotm/service/dataFileBinary

Method

The request type GET

URL parameters

`fileId: String`. The data file ID. Mandatory.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

The Binary content of the data file.

Response items

None

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f05
79e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.
112.229.91:9447/ibm/iotm/service/ dataFileBinary ?fileId=06c870e5-1701-419a
-8a84-0f40ee82fe33_c901e409-d9db-4f02-8d9c-fa6002929701&user=auto_mm&solution
=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Upload data files to a data group

Uploads one or more data files to a data group. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataFileBinary

Method

The request type POST

URL parameters

`groupId: String`. The data group ID. Mandatory.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`Content-type: multipart/form-data`.

`Content-Disposition: form-data; name="files[]"; filename=your file name`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

The Binary content of the data file.

Success response

```
200
{"result":{"Defect.zip":{"id":"9f7aa9d6-24d6-4611-8d4d-79a52d2bab02_76e8b2cb-057a-
4904-8fad-bbca93abd80b","count":"3","name":"Defect.zip","errorMsg":"","updatedTime"
:"1512021663498","url":"\/user\/AUTO\/VI\/datagroup\/9f7aa9d6-24d6-4611-8d4d-79a52d2
bab02\/1512021663498.zip"}},"error_message":{}}
```

Response items

`id:` *`String`*. The data file ID.

`name:` *`String`*. The file name.

`count:` *`String`*. The image count.

`updatedTime:` *`Time`*. The last time that the data file was updated.

`fileUrl:`. The file path that is stored in the server.

`error_message:`. The error message of an upload response.

Sample call

```
curl -k -X POST -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e53
3dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
-H "Content-Type:multipart/form-data"  -H "Content-Disposition:
form-data; name=\"files[]\"; filename=\"Defect.zip\"" "https://9.112.229.91:9447
/ibm/iotm/service/dataFileBinary?user=auto_mm&solution=vi&groupId=9f7aa9d6-24d6-
4611-8d4d-79a52d2bab02" --connect-timeout 600 -F file=@C:\\code\\Automation_98\\
API\\vi\\VI_API\\file\\Defect.zip
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Delete one data file

Deletes one data file. The model manager has authority to use this API.

URL

/ibm/iotm/service/dataFile/*fileId*

Method

The request type DELETE

URL parameters

`fileId:` *`String`*. The data file ID. Mandatory.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

APIKEY: *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

```
[{}]
```

Success response

```
200
[{}]
```

Response items

None

Sample call

```
curl -k -X DELETE -H "APIKEY:8b79658
225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f057
9e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/dataFile/
9f7aa9d6-24d6-4611-8d4d-79a52d2bab02_e2cc3d4a-bae9-
4600-9dff-d80fa3ee8832?user=auto_mm&solution=vi" -d '[{}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Unlabeled data group services

The unlabeled data group services enable you to perform such tasks as uploading an unlabeled compressed image file, creating an unlabeled data group, and getting one unlabeled data group.

Upload an unlabeled compressed image file

Uploads an unlabeled compressed file to the server. The model manager has authority to use this API.

URL

/ibm/iotm/service/unLabeledImageZipServlet

Method

The request type POST

URL parameters

`groupType:` *`String`*. The data group type. The value must be *classification* or *objectdetection*. Mandatory.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

`Content-type: multipart/form-data`.

`Content-Disposition: form-data; name="files[]"; filename=`*`your file name`*.

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

The Binary content of the data file.

Success response

```
200 {"message":{},"result":{"imageZip":{"imageZipId":"b76b33b0-26d6-4f85-83ea-e520edbe5ff8",
"labeledImageCount":0,"labelCount":0,"imageZipName":"images.zip","count":85,"updatedTime":
```

```
"2019-03-19 00:18:53.304","url":"\/user\/TENANT\/VI\/unlabeledgroup\/b76b33b0-26d6-4f85-
83ea-e520edbe5ff8.zip"}}}
```

Response items

`imageZip: JSON Object`. The information for the uploaded compressed image file. The information includes the `imageZipId`, `LabeledImageCount`, `labelCount`, `imageZipName`, `count`, `updatedTime`, and URL.

`imageZipId: String`. The ID that is assigned for uploaded compressed image file.

`LabeledImageCount: Integer`. The number of images that were pre-labeled in the compressed image file.

`labelCount: Integer`. The number of labels that are assigned to the images in the compressed image file.

`imageZipName: String`. The file name of the compressed image file.

`count: Integer`. The total number of images in the compressed image file.

`updatedTime: Time`. The time that the compressed image file was uploaded.

`Url: String`. The file path that is stored in the server.

`error_message: JSON Object`. The error message for an upload response.

Sample call

```
curl -k -X POST -H"APIKEY:apikey"
  -H "Content-Type:multipart/form-data"  -H "Content-Disposition: form-data;
 name=\"files[]\"; filename=\"image.zip\"" "https://iotm.predictivesolutionsapps
.ibmcloud.com/ibm/iotm/service/unLabeledImageZipServlet?user=auto_mm&solution=vi&
 groupType=classification" --connect-timeout 600
-F file=@C:\\code\\Automation_98\\API\\vi\\VI_API\\file\\image.zip
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Create an unlabeled data group

Creates an unlabeled data group with the uploaded compressed image file. The model manager has authority to use this API.

URL

/ibm/iotm/service/unLabeledGroup

Method

The request type POST

URL parameters

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`Content-type: multipart/form-data`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

`groupName: String`. The data group name.

`groupType: String`. The data group type. The value must be *classification* or *objectdetection*.

imageZip: *JSON Object*. The information for the uploaded compressed image file. You can get the information from the response when you upload an unlabeled compressed image file. The information includes the imageZipId, LabeledImageCount, labelCount, imageZipName, count, updatedTime, and URL.

imageZipId: *String*. The ID that is assigned for uploaded compressed image file.

LabeledImageCount: *Integer*. The number of images that were pre-labeled in the compressed image file.

labelCount: *Integer*. The number of labels that are assigned to the images in the compressed image file.

imageZipName: *String*. The file name of the compressed image file.

count: *Integer*. The total number of images in the compressed image file.

updatedTime: *Time*. The time that the compressed image file was uploaded.

Url: *String*. The file path that is stored in the server.

Sample body

```
[{"groupName":"DI","groupType":"classification","imageZip":{"imageZipId":
"b76b33b0-26d6-4f85-83ea-e520edbe5ff8","labeledImageCount":0,"labelCount"
:0,"imageZipName":"image.zip","count":85,"updatedTime":"2019-03-19
00:18:53.304","url":"/user/TENANT/VI/unlabeledgroup/b76b33b0-26d6-4f85-
83ea-e520edbe5ff8.zip"}}]
```

Success response

```
200
[{"groupName":"DI","id":"04f4eaf3-dc93-4ad8-b2e2-d3fbd202c758",
"groupType":"classification","imageZip":{"imageZipId":"b76b33b0
-26d6-4f85-83ea-e520edbe5ff8","labeledImageCount":0,"labelCount"
:0,"imageZipName":"image.zip","count":85,"updatedTime":
"2019-03-19 00:18:53.304","url":"/user/TENANT/VI/unlabeledgroup
/b76b33b0-26d6-4f85-83ea-e520edbe5ff8.zip"}}]
```

Response items

id: *String*. The data group ID.

groupName: *String*. The data group name.

groupType: *String*. The data group type.

createdBy: *String*. The data group owner.

groupStatus: *String*. The status of the unlabeled data group.

updatedTime: *Time*. The last time that the data group was updated.

imageZip: *JSON Object*. The information that is associated with the uploaded compressed image file. The information includes the imageZipId, LabeledImageCount, labelCount, imageZipName, count, updatedTime, and URL.

imageZipId: *String*. The ID that is assigned for uploaded compressed image file.

LabeledImageCount: *Integer*. The number of images that were pre-labeled in the compressed image file.

labelCount: *Integer*. The number of labels that are assigned to the images in the compressed image file.

imageZipName: *String*. The file name of the compressed image file.

count: *Integer*. The total number of images in the compressed image file.

updatedTime: *Time*. The time that the compressed image file was uploaded.

Url: *String*. The file path that is stored in the server.

Sample call

```
curl –k –X POST -H "APIKEY:apikey" -H
"Content-Type:application/json;charset=UTF-8" "https://iotm.predictivesolutionsapps.
ibmcloud.com/ibm/iotm/service/unLabeledGroup?user=auto_mm&solution=vi"  --data
 '[{"groupName":"DI","groupType":"classification","imageZip":{"imageZipId":"b76b
33b0-26d6-4f85-83ea-e520edbe5ff8","labeledImageCount":0,"labelCount":0,"imageZipName"
:"image.zip","count":85,"updatedTime":"2019-03-19 00:18:53.304","url":"/user/TENANT/
VI/unlabeledgroup/b76b33b0-26d6-4f85-83ea-e520edbe5ff8.zip"}}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get one unlabeled data group

Gets one unlabeled data group. The model manager has authority to use this API.

URL

/ibm/iotm/service/unLabeledGroup/*groupId*

Method

The request type GET

URL parameters

`groupId:` *`String`*. The data group ID.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200[{"groupName":"DI","id":"04f4eaf3-dc93-4ad8-b2e2-d3fbd202c758",
"groupType":"classification","imageZip":{"imageZipId":"b76b33b0-26d6-4f85-83ea
-e520edbe5ff8","labeledImageCount":0,"labelCount":0,"imageZipName":"image.zip"
,"count":85,"updatedTime":"2019-03-19 00:18:53.304","url":"/user/TENANT/VI/
unlabeledgroup/b76b33b0-26d6-4f85-83ea-e520edbe5ff8.zip"}}]
```

Response items

`id:` *`String`*. The data group ID.

`groupName:` *`String`*. The data group name.

`groupType:` *`String`*. The data group type.

`createdBy:` *`String`*. The data group owner.

`groupStatus:` *`String`*. The status of the unlabeled data group.

`updatedTime:` *`Time`*. The last time that the data group was updated.

`imageZip:` *`JSON Object`*. The information that is associated with the uploaded compressed image file. The information includes the `imageZipId`, `LabeledImageCount`, `labelCount`, `imageZipName`, `count`, `updatedTime`, and URL.

`imageZipId:` *`String`*. The ID that is assigned for uploaded compressed image file.

`LabeledImageCount: Integer`. The number of images that were pre-labeled in the compressed image file.

`labelCount: Integer`. The number of labels that are assigned to the images in the compressed image file.

`imageZipName: String`. The file name of the compressed image file.

`count: Integer`. The total number of images in the compressed image file.

`updatedTime: Time`. The time that the compressed image file was uploaded.

`Url: String`. The file path that is stored in the server.

Sample call

```
curl -k -H "APIKEY:apikey"
 "https://iotm.predictivesolutionsapps.ibmcloud.com/ibm/iotm/service
/unLabeledGroup/04f4eaf3-dc93-4ad8-b2e2-d3fbd202c758?user=auto_mm&
solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Model services

The model services enable you to perform such tasks as getting all models, getting one model, and creating a model.

Get all models

Gets all existing models. The model manager has authority to use this API.

URL

/ibm/iotm/service/model

Method

The request type GET

URL parameters

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

APIKEY: *encrypted key*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[{
        "modelId": "a9150695-5dc5-4bef-937d-c71fc186bc14",
        "groupIds": "e83f9faf-2b08-44d1-81b0-7c69e6407329",
        "modelName": "appscan test01",
        "createdBy": "demoadm@cn.ibm.com",
        "modelType": "classification",
        "statusStatistics": {},
```

```
        "dataFormat": "png",
        "retrainPolicy": {
            "scheduler": "Weekly, Sunday",
            "ratio": "",
            "reuseDayNumber": "",
            "maxPieceNumber": "",
            "archiveDataType": "0",
            "imageNumber": 1000,
            "accuracy": "70"
        },
        "parameters": {
            "imageSize": "100*100",
            "confidence": 100,
            "trainParam": {
                "epoc": 50,
                "ratioTrain": 80,
                "stepsize": 30,
                "test_iter": 20,
                "snapshot": 100,
                "momentum": 0.9,
                "ratioVal": 20,
                "learningRate": 0.0001,
                "display": 1,
                "learningRatePolicy": "step",
                "weight_decay": 0.00001,
                "test_epoc": 10,
                "network": "GoogLeNet",
                "algorithm": "CNN",
                "test_interval": 5,
                "gamma": 0.1,
                "solver_type": "SGD",
                "maxIter": 600
            }
        },
        "updatedTime": "2018-11-16T00:16:20.032-06:00",
        "productType": "appscan test01"
    }
…....
]
```

Response items

`modelId: String`. The model ID.

`modelName: String`. The model name.

`createdBy: String`. The model owner.

`updatedTime: Time`. The last time that the model was updated.

`productType: String`. The model product type.

`description: String`. The model description.

`StatusStatistics: String`. The model status.

`groupIds: String`. The model data group ID.

`dataFormat: String`. The model data format.

`retrainPolicy: JSON Object`. The model retrain policy.

`scheduler: Time`. The model retrain date, which in Monday to Sunday.

`imageNumber: Integer`. An image number that is greater than this number can be retrained.

`accuracy: Double`. A model accuracy value that is less than this number can be retrained.

`parameters: JSON Object`. Contains image size and model confidence information and trainParam field.

`imageSize: Double * Double`. The image size.

`confidence: Double`. The model defined confidence value.

`trainParam: JSON Object`. Contains the training parameter information.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e
533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112
.229.91:9447/ibm/iotm/service/model?user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get one model

Gets one existing model. The model manager has authority to use this API.

URL

/ibm/iotm/service/model/<modelID>

Method

The request type GET

URL parameters

`modelId:` *`String`*. The model ID. Mandatory.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response


```
200
[
    {
        "modelId": "f39df888-7559-4283-ae9d-b3504118dddd",
        "groupIds": "9e744bc4-0a22-4bf3-9e63-fa39135842de",
        "modelName": "arthuryolo22000v2",
        "createdBy": "demoadm@cn.ibm.com",
        "modelType": "objectdetection",
        "description": "arthuryolo22000v2",
        "statusStatistics": {
            "trained": "1"
        },
        "dataFormat": "png",
        "retrainPolicy": {
            "scheduler": "Weekly, Sunday",
            "ratio": "",
            "reuseDayNumber": "",
            "maxPieceNumber": "",
            "archiveDataType": "0",
            "imageNumber": 1000,
            "accuracy": "70"
        },
        "trainedBy": "demoadm@cn.ibm.com",
        "parameters": {
            "JobInstanceMap": {
                "4a9605a2-03e3-43ff-9a47-d7638a0145d9": "f39df888-7559-4283-ae9d-
b3504118dddd_1541744213466"
            },
            "odRetrainUrl": "/user/Q3T1/VI/modellist/f39df888-7559-4283-ae9d-
b3504118dddd_1541761094044.zip",
```

```
            "imageSize": "100*100",
            "confidence": 100,
            "trainParam": {
                "ratioTrain": 80,
                "subBatchSize": 2,
                "scales": "1,1",
                "ratioVal": 20,
                "learningRate": 0.0001,
                "batchSize": 16,
                "steps": "6000,10000",
                "recommend": 0,
                "network": "YoloV2",
                "iteration": 10000,
                "algorithm": "YOLO"
            }
        },
        "updatedTime": "2018-11-09T00:24:19.949-06:00",
        "productType": "arthuryolo22000v2"
    }
]
```

Response items

`modelId:` *`String`*. The model ID.

`modelName:` *`String`*. The model name.

`createdBy:` *`String`*. The model owner.

`updatedTime:` *`Time`*. The last time that the model was updated.

`productType:` *`String`*. The model product type.

`description:` *`String`*. The model description.

`StatusStatistics:` *`String`*. The model status.

`groupIds:` *`String`*. The model data group ID.

`dataFormat:` *`String`*. The model data format.

`retrainPolicy:` *`JSON Object`*. The model retrain policy.

`scheduler:` *`Time`*. The model retrain date, which in Monday to Sunday.

`imageNumber:` *`Integer`*. An image number that is greater than this number can be retrained.

`accuracy:` *`Double`*. A model accuracy value that is less than this number can be retrained.

`parameters:` *`JSON Object`*. Contains image size and model confidence information and the odRetrainUrl, JobInstanceMap and trainParam fields.

`imageSize:` *`Double * Double`*. The image size.

`confidence:` *`Double`*. The model defined confidence value.

`trainParam:` *`JSON Object`*. Includes the algorithm and network and hyper parameter information.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533
dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112.229.91:
9447/ibm/iotm/service/model/ab219b13-16c0-4c7b-ae74-721f4719e314?user=auto_mm&so
lution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Create a model

Creates a model. The model manager has authority to use this API.

URL

/ibm/iotm/service/model

Method

The request type POST

URL parameters

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`Content-type: application/json`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

`modelName: String`. The model name.

`modelType: String`. The model type. Options are "classification" and "objectdetection".

`createdBy: String`. The model owner.

`description: String`. The model description.

`groupIds: String`. The model data group ID.

`dataFormat: String`. The model data format.

`retrainPolicy: JSON Object`. The model retrain policy.

`scheduler: Time`. The model retrain date, which in Monday to Sunday.

`imageNumber: Integer`. An image number that is greater than this number can be retrained.

`accuracy: Double`. A model accuracy value that is less than this number can be retrained.

`parameters: JSON Object`. Contains image size, model confidence information, and properties of training parameters including algorithm and network and hyper parameter information.

The training parameters, hyper parameters, and model type values have corresponding relationships. If the model type value is: "classification", the "algorithm" value is "CNN", and the corresponding "network" has 3 options: "GoogleNet", "LeNet", or "AlexNet". The corresponding hyper parameters include: "learningRate", which is a real number between 0 and 1; "maxIter" which is a positive integer; "stepsize" which is a positive integer; "gamma" which is a real number between 0 and 1; "learningRatePolicy" which has a value of "step"; "test_iter" which is a positive integer; "test_interval" which is a positive integer; "snapshot", which is an integer equal to or greater than maxIter/20; "ratioTrain", the ratio of images used for training which are integers between 0 and 100; and "ratioVal", the ratio of images used for validation which is an integer between 0 and 100.

If the model type value is "objectdetection", "algorithm" values include: "FRCNN", "YOLO", "SSD". If "algorithm" value is "FRCNN", the corresponding "network" has two options: "ZfNet" and "VGG16". The corresponding hyper parameters include: "learningRate", which is a real number between 0 and 1; "iteration", which consists of comma-separated positive integers, for example: 10000, 10000, 10000, 10000; "stepsize", which is a positive integer; "gamma", which is a real number between 0 and 1; "ratioTrain", which is an integer between 0 and 100; and "ratioVal", which is an integer between 0 and 100.

If the "algorithm" value is "YOLO", the corresponding "network" value has 3 options: "YoloV2", "TinyYolo", and "YoloV1". The corresponding hyper parameters include: "learningRate", which is a real

number between 0 and 1; "iteration", which is a positive integer; "steps", which is comma-separated positive integers, for example: 100, 1000, 5000, 8000; "batchSize", which is a positive integer; "scales", which is comma-separated real numbers, for example: 0.1, 0.1, 0.1, 0.1; "subBatchSize", which is a positive integer; "ratioTrain", which is an integer between 0 and 100; "ratioVal", which is an integer between 0 and 100.

If the "algorithm" value is "SSD", the corresponding "network" is "SSD300". Corresponding hyper parameters include "learningRate", which is a real number between 0 and 1; "iteration", which is a positive integer; "steps", which is comma-separated positive integers, for example: 100, 1000, 5000, 8000; "batchSize", which value is a positive integer; "learningRatePolicy", which has a value of "multistep"; "snapshot" which is an integer equal or greater than iteration/20; "ratioTrain", which is an integer between 0 and 100; and "ratioVal", which is an integer between 0 and 100.

`confidence:` *`Double`*. The model defined confidence value.

`productType:` *`String`*. The model product type.

Sample body

```
[{"modelName":"walkermodel1","productType":"walkermodel1","dataFormat":
"png","modelType":"classification","retrainPolicy":{"imageNumber":"1000",
"accuracy":"70","scheduler":"Weekly, Sunday","reuseDayNumber":"",
"maxPieceNumber":"","ratio":""},"description":"this is a test",
"groupIds":"b5164942-370c-418d-9fc2-f6ffd860b79d","parameters":
{"imageSize":"100*100","confidence":100,"trainParam":{"ratioTrain":80,
"ratioVal":20,"gamma":0.1,"stepsize":33,"maxIter":100,"learningRate"
:0.01,"learningRatePolicy":"step","test_iter":1,"test_interval":2,
"snapshot":10,"network":"GoogLeNet","algorithm":"CNN"}}}]
```

Success response

```
201

[{"modelId":"4d0dfe01-7a26-4071-92fb-ec597e7863ce","modelName":
"walkermodel1","description":"this is a test","productType":
"walkermodel1","groupIds":"b5164942-370c-418d-9fc2-f6ffd860b79d",
"dataFormat":"png","createdBy":"automm@cn.ibm.com","updatedTime":
"2018-02-13T00:44:31.288-06:00","parameters":{"imageSize":"100*100",
"confidence":100,"trainParam":{"ratioTrain":80,"stepsize":33,
"test_iter":1,"snapshot":10,"momentum":0.9,"ratioVal":20,
"learningRate":0.01,"display":1,"learningRatePolicy":"step",
"weight_decay":1.0E-5,"network":"GoogLeNet","test_interval":
2,"algorithm":"CNN","gamma":0.1,"solver_type":"SGD","maxIter":100}},
"retrainPolicy":{"ratio":"","scheduler":"Weekly, Sunday",
"maxPieceNumber":"","reuseDayNumber":"","imageNumber":"1000",
"accuracy":"70"}}]
```

Response items

`modelId:` *`String`*. The model ID.

`modelName:` *`String`*. The model name.

`productType:` *`String`*. The product type.

`createdBy:` *`String`*. The model owner.

`updatedTime:` *`Time`*. The last time that the model was updated.

`productType:` *`String`*. The model product type.

`description:` *`String`*. The model description.

`groupIds:` *`String`*. The model data group ID.

`dataFormat:` *`String`*. The model data format.

`retrainPolicy:` *`JSON Object`*. The model retrain policy.

`scheduler:` *`Time`*. The model retrain date, which in Monday to Sunday.

`imageNumber:` *`Integer`*. An image number that is greater than this number can be retrained.

`accuracy: Double`. A model accuracy value that is less than this number can be retrained.

`parameters: JSON Object`. Contains image size and model confidence information.

`confidence: Double`. The model defined confidence value.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c
77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
-H "Content-Type:application/json;charset=UTF-8" "https://9.112.229.91/
ibm/iotm/service/model?&tenant=VITest&solution=vi&user=auto_mm"
--data ' [{"modelName":"testcreatemodel11","imagegroupList":"[{\"id\":
\"b5164942-370c-418d-9fc2-f6ffd860b79d\",\"dataFiles\":[{\"id\":\
"b5164942-370c-418d-9fc2-f6ffd860b79d--b5164942-370c-418d-9fc2-
f6ffd860b79d_74674410-06df-4a9e-a9e9-12f5038efc22\",\"name\":\
"Good_val.zip\",\"count\":5,\"updatedTime\":\"12/27/2017,
10:38:13 AM\"}],\"name\":\"Good\",\"updatedTime\":\"12/27/2017,
10:36:10 AM\"}]","productType":"testcreatemodel11","dataFormat":
"png","modelType":"classification","retrainPolicy":{"imageNumber":
"1000","accuracy":"70","scheduler":"Weekly, Sunday","reuseDayNumber"
:"","maxPieceNumber":"","ratio":""},"description":"this is a test",
"groupIds":"b5164942-370c-418d-9fc2-f6ffd860b79d","parameters":
{"imageSize":"100*100","confidence":100,"trainParam":{"ratioTrain":
80,"ratioVal":20,"gamma":0.1,"stepsize":33,"maxIter":100,"learningRate"
:0.01,"learningRatePolicy":"step","test_iter":1,"test_interval":2,
"snapshot":10,"network":"GoogLeNet","algorithm":"CNN"}}}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Update a model

Updates one existing model. The model manager has authority to use this API.

URL

/ibm/iotm/service/model/*modelId*

Method

The request type PUT

URL parameters

`modelId: String`. The model ID. Mandatory.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`Content-type: application/json`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

`groupIds: String`. The data group ID.

Sample body

```
[{"groupIds":"45805b3c-f234-4f18-a55e-61ce203a7db4,678a3f42-f9d4-48a2-bb
29-c226d61a08ea"}]
```

Success response

```
200
[{}]
```

Response items

None

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533dd
34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" -H "Content-Type:application
/json;charset=UTF-8" -X PUT "https://9.112.229.91:9447/ibm/iotm/service/model/106ef
cb7-1338-4ed2-b8ca-27b45e11e865?user=auto_mm&solution=vi"  --data '[{"groupIds":"9f
7aa9d6-24d6-4611-8d4d-79a52d2bab02"}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Delete one model

Deletes one existing model and all related model instances. The model manager has authority to use this API.

URL

/ibm/iotm/service/model/*modelId*

Method

The request type DELETE

URL parameters

`modelId:` *`String`*. The model ID. Mandatory.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

```
[{}]
```

Success response

```
200
[{}]
```

Response items

None

Sample call

```
curl -k -X DELETE -H "APIKEY:8b79658225de5
3488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e53
3dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/model/
106efcb7-1338-4ed2-b8ca-27b45e11e865?user=auto_mm&solution=
vi" -d '[{}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get all shared models

Gets all shared models. The model manager has authority to use this API.

URL

/ibm/iotm/service/model/

Method

The request type GET

URL parameters

`category:'shared'`. Used to identify the model catalog. Mandatory.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response


```
[{"modelId":"0f9c8d47-5ccc-4e52-8e0d-0596e792a2cd",
"groupIds":"e5e83a2d-5814-45b3-af40-74096198cd39","modelName":
"Car Seat Defect Inspection","createdBy":"vimodelmanager@163.com",
"modelType":"objectdetection","description":"This is the object
detection model we build to detect the wrinkle defect for car seat.
The inspection result will identify whether there is wrinkle defect
on the car and localize the defect on the image.","statusStatistics":
{"deployed":"1"},"dataFormat":"jpg","retrainPolicy":{"scheduler":
"Weekly, Sunday","ratio":"","reuseDayNumber":"","maxPieceNumber":
"","imageNumber":"1000","accuracy":"70"},"trainedBy":
"vimodelmanager@163.com","parameters":{"odRetrainUrl":
"\/user\/VI\/VI\/modellist\/carseatfrcnn_1516957839865.zip",
"lastRetrainData":
"[{\"groupId\":\"e5e83a2d-5814-45b3-af40-74096198cd39\"}]",
"imageSize":"100*100","confidence":"100"},"updatedTime":
"2018-01-26T03:11:05.804-06:00","productType":"carseat"}]
```


Response items

`modelId:` *`String`*. The model ID.

`groupIds:` *`String`*. The model data group ID.

`modelName:` *`String`*. The model name.

`createdBy:` *`String`*. The model owner.

`modelType:` *`String`*. The model type.

`description:` *`String`*. The model description.

`updatedTime:` *`Time`*. The last time that the model was updated.

`productType:` *`String`*. The model product type.

`StatusStatistics:` *`String`*. The model status.

`dataFormat:` *`String`*. The model data format.

`retrainPolicy:` *`JSON Object`*. The model retrain policy.

`scheduler:` *`Time`*. The model retrain date, which in Monday to Sunday.

`imageNumber: Integer`. An image number that is greater than this number can be retrained.

`accuracy: Double`. A model accuracy value that is less than this number can be retrained.

`parameters: JSON Object`. Contains image size and model confidence information.

`imageSize: Double * Double`. The image size.

`confidence: Double`. The model defined confidence value.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb
8c149f6aab2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713
bf305a158fdf485cc6f275f5" " https://9.112.229.91:9447/ibm/
iotm/service/model?category=shared&tenant=Q3T1&solution=
vi&user=testuser"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Model instance services

The model instance services enable you to perform such tasks as creating a model instance, returning a model instance belonging to a model, and returning one model instance.

Create a model instance

Creates a model instance. The model manager has authority to use this API.

URL

/ibm/iotm/service/modelInstance

Method

The request type POST

URL parameters

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`Content-type: application/json`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

`modelID: String`. The model ID.

`trainData: JSON Object`. The model train data.

`groupId: String`. The data group ID.

`fileIds: String`. The data file ID.

Sample body

```
[
  {
    "modelId": "91a21e63-17e6-4e0e-bbe8-5c6b9d599a35",
    "trainData": [
      {
        "groupId": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02",
```

```
        "fileIds": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02_76e8b2cb-057a-4904-8
fad-bbca93abd80b"
      },
      {
        "groupId": "d84c028e-235a-4c13-8a77-894500b8a736",
        "fileIds": "d84c028e-235a-4c13-8a77-894500b8a736_fe201c83-b138-4dca-b
918-ae1b9f7db894"
      },
      {
        "groupId": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb",
        "fileIds": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-44d8-a
2fa-e74c1578eebe"
      }
    ]
  }
]
```

Success response

```
200

[
    {
        "status": "deployed",
        "validateData": [
            {
                "groupId": "1a7d4c81-36ff-4a3e-a614-9e996fb380ba"
            }
        ],
        "instanceId": "4cc40ab4-d3b3-4911-b7a0-90281248c075_1542340917709",
        "validateResult": {
            "matrix": [
                [ "","scratch", "Chris"],
                ["mAP","-","-" ],
                ["recall","-","-"]
            ],
            "type": "objectdetection",
            "accuracy": "-",
            "fileCount": "0"
        },
        "trainedBy": "demoadm@cn.ibm.com",
        "validateTime": "2018-11-15T22:01:57.726-06:00",
        "fileCount": "0",
        "modelId": "4cc40ab4-d3b3-4911-b7a0-90281248c075",
        "createdBy": "demoadm@cn.ibm.com",
        "instanceName": "1",
        "parameters": {
            "edges": [
                {"id": "1542102975873" },
                {"id": "1508476898000"}
            ],
            "modelType": "objectdetection",
            "attached": true,
            "edgesDeploying": []
        },
        "isDeletable": false,
        "modelUrl": "/user/Q3T1/VI/modellist/frcnnmodel_1542340914671.zip",
        "trainData": [
            {"groupId": "1a7d4c81-36ff-4a3e-a614-9e996fb380ba"}
        ],
        "accuracy": "-",
        "updatedTime": "2018-11-16T04:21:34.656-06:00"
    }
]
```

Response items

`instanceId:` *`String`*. The model instance ID.

`modelId:` *`String`*. The model ID.

`instanceName:` *`Integer`*. The model instance name.

`createdBy:` *`String`*. The model instance owner.

`updatedTime:` *`Time`*. The last time that the model instance was updated.

`status:` *`String`*. The model instance status.

`trainData: JSON Object`. The model train data.

`groupId: String`. The data group ID.

`fileIds: String`. The data file ID.

`parameters: JSON Object`. Contain information about the deployed edge.

`edges: JSON Array`. Contain ID information about the deployed edge.

`libModelUrl: String`. modelURL of the model instance in the model library.

`lastAutoTrainTime: String`. The time of the last auto retrain, in yyyy-mm-dd hh:mm:ss.SSS format.

`trainType: String`. The type of training: online_train, retrain_manual, or retrain_auto.

`onDataPreparing: String`. 1 means data is preparing. 0 means data preparation is complete.

`snapshot: String`. The snapshot information used for online training.

`unlabeledGroupId: String`. The unlabeled group ID.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533dd
34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" -H "Content-Type:application/
json;charset=UTF-8" " https://9.112.229.91:9447/ibm/iotm/service/modelInstance?user
=auto_mm&solution=vi"  --data '[  {    "modelId": "91a21e63-17e6-4e0e-bbe8-5c6b9d59
9a35",    "trainData": [      {        "groupId": "9f7aa9d6-24d6-4611-8d4d-79a52d2ba
b02",        "fileIds": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02_76e8b2cb-057a-4904-8fa
d-bbca93abd80b"      },      {        "groupId": "d84c028e-235a-4c13-8a77-894500b8a73
6",        "fileIds": "d84c028e-235a-4c13-8a77-894500b8a736_fe201c83-b138-4dca-b918-a
e1b9f7db894"      },      {        "groupId": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb",
        "fileIds": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-44d8-a2fa-e74c1
578eebe"      }    ]  }]
'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get the model instance that belongs to the model

Gets the model instance that belongs to the model. The model manager and the data scientist have authority to use this API.

URL

/ibm/iotm/service/modelInstance

Method

The request type GET

URL parameters

`modelId: String`. The model ID. Mandatory.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
  {
    "modelId": "91a21e63-17e6-4e0e-bbe8-5c6b9d599a35",
    "createdBy": "auto_mm",
    "status": "draft",
    "instanceName": "1",
    "instanceId": "91a21e63-17e6-4e0e-bbe8-5c6b9d599a35_1512029557869",
    "trainData": [
      {
        "fileIds": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02_76e8b2cb-057a-4904-
8fad-bbca93abd80b",
        "groupId": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02"
      },
      {
        "fileIds": "d84c028e-235a-4c13-8a77-894500b8a736_fe201c83-b138-4dca-
b918-ae1b9f7db894",
        "groupId": "d84c028e-235a-4c13-8a77-894500b8a736"
      },
      {
        "fileIds": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-44d8-
a2fa-e74c1578eebe",
        "groupId": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb"
      }
    ],
    "fileCount": "9",
    "updatedTime": "2017-11-30T16:12:37.869+08:00"
  }
]
```

Response items

`modelId: String`. The model ID.

`createdBy: String`. The model instance owner.

`status: String`. The model instance status.

`instanceId: String`. The model instance ID.

`instanceName: Integer`. The model instance name.

`trainData: JSON Object`. The model train data.

`groupId: String`. The data group ID.

`fileIds: String`. The data file ID.

`fileCount: Integer`. The total number of the model image count.

`updatedTime: Time`. The last time that the model instance was updated.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533d
d34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112.229.91:9447
/ibm/iotm/service/modelInstance?modelId=91a21e63-17e6-4e0e-bbe8-5c6b9d599a35&user=
auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get one model instance

Gets one model instance. The model manager and the data scientist have authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*

Method

The request type GET

URL parameters

`modelInstanceId: String`. The model instance ID. Mandatory.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
  {
    "modelId": "91a21e63-17e6-4e0e-bbe8-5c6b9d599a35",
    "createdBy": "auto_mm",
    "status": "draft",
    "instanceName": "1",
    "instanceId": "91a21e63-17e6-4e0e-bbe8-5c6b9d599a35_1512029557869",
    "trainData": [
      {
        "fileIds": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02_76e8b2cb-057a-
4904-8fad-bbca93abd80b",
        "groupId": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02"
      },
      {
        "fileIds": "d84c028e-235a-4c13-8a77-894500b8a736_fe201c83-b138-
4dca-b918-ae1b9f7db894",
        "groupId": "d84c028e-235a-4c13-8a77-894500b8a736"
      },
      {
        "fileIds": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-
44d8-a2fa-e74c1578eebe",
        "groupId": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb"
      }
    ],
    "updatedTime": "2017-11-30T16:12:37.869+08:00"
  }
]
```

Response items

`instanceId: String`. The model instance ID.

`modelId: String`. The model ID.

`instanceName: Integer`. The model instance name.

`createdBy: String`. The model instance owner.

`updatedTime: Time`. The last time that the model instance was updated.

`status: String`. The model instance status.

`trainData: JSON Object`. The model train data.

`groupId: String`. The data group ID.

`fileIds: String`. The data file ID.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c
77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/modelInstance/91a21e63
-17e6-4e0e-bbe8-5c6b9d599a35_1512029557869?user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get the model instance validation result

Gets the model instance validation result for the report. The model manager and the data scientist have authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*/validateResult

Method

The request type GET

URL parameters

`modelInstanceId: String`. The model instance ID. Mandatory.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

APIKEY: *encrypted key*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
    {
        "instanceName": "FirstBlood_2017-08-08 19:21:53.888",
        "validateResult": {
            "matrix": [
                [
                    "",
                    "type1",
                    "type2",
                    "type3"
                ],
                [
                    "type1",
                    "77,35,45",
                    "0,0,40",
                    "6,3,50"
                ],
                [
                    "type2",
```

```
                    "17,8,45",
                    "100,40,40",
                    "4,2,50"
                ],
                [
                    "type3",
                    "4,2,45",
                    "0,0,40",
                    "90,45,50"
                ],
                [
                    "Total",
                    "45",
                    "40",
                    "50"
                ]
            ],
            "accuracy": "84.56"
        },
        "validateTime": "2017-08-09T22:58:46.416+08:00",
        "fileCount": "200"
    }
]
```


Response items

`instanceName:` *`Integer`*. The model instance name.

`validateResults:` *`JSON Object`* . The model instance validation result.

`validateTime:` *`Time`*. The model instance validation time.

`fileCount:` *`Integer`*. The model instance image number.

`accuracy:` *`Double`*. The model instance accuracy value.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e53
3dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112.229.91:9447/
ibm/iotm/service/modelInstance/91a21e63-17e6-4e0e-bbe8-5c6b9d599a35_1512029557869/
validateResult?user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Inspection result services

The inspection result services enable you to perform such tasks as getting an inspection result list, getting inspection result details, and getting an inspection result cell overview.

Get the inspection result list

Gets the inspection result list. The inspector and the supervisor have authority to use this API.

URL

/ibm/iotm/service/inspectResult

Method

The request type GET

URL parameters

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

`isConfirmed: Integer`. A 0 means that the inspection result is unconfirmed. A 1 means that the inspection result is confirmed. A 2 means that the inspection result is unknown. Optional.

`cell: String`. Used to identify the cell. Optional.

`inspectTime: Time`. Used to identify the time of the inspection.

`reverse: Boolean`. Used to identify the order of the inspection result list.

`pageCount: Integer`. The page count.

`sampleRate: Integer`. The sample check rate.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response


```
200
[{"id":"plant1,line1,cell1_1511768220935_AUTO|auto_1511768206483.jpg",
"isConfirmed":"0","inspectFileUrl":"imageserver\/911222954\/\/AUTO|
auto_plant1_line1_cell1_1511768206483.jpg","isUpdated":"0",
"defectTypeCount":"1","cell":"plant1,line1,cell1","instanceId":"ebf9
4e98-958c-48b0-8377-826241ffccd8_1511765289053","defectCount":"1",
"confidence":"99.87","inspectId":"auto_1511768206483.jpg",
"inspectTime":"1511768220935"},{"id":"plant1,line1,
cell1_1511761780903_AUTO|auto_1511761766470.jpg","isConfirmed":"0","
inspectFileUrl":"imageserver\/911222954\/\/AUTO|
auto_plant1_line1_cell1_1511761766470.jpg","isUpdated":"0",
"defectTypeCount":"1","cell":"plant1,line1,cell1","instanceId":
"9ab990d6-465a-48b0-838c-416c8e4a9879_1511761328538","defectCount":
"1","confidence":"99.79","inspectId":"auto_1511761766470.jpg",
"inspectTime":"1511761780903"},
{"id":"plant1,line1,cell1_1511761559536_AUTO|auto_1511761541254.jpg",
"isConfirmed":"0","i
nspectFileUrl":"imageserver\/911222954\/\/AUTO|
auto_plant1_line1_cell1_1511761541254.jpg","isUpdated":"0",
"defectTypeCount":"1","cell":"plant1,line1,cell1","instanceId":
"9ab990d6-465a-48b0-838c-416c8e4a9879_1511761328538","defectCount":
"1","confidence":"99.87","inspectId":"auto_1511761541254.jpg",
"inspectTime":"1511761559536"},{"id":"plant1,line1,
cell1_1511421414687_AUTO|auto_1511421410914.jpg","isConfirmed":"0",
"inspectFileUrl":"imageserver\/911222954\/\/AUTO|
auto_plant1_line1_cell1_1511421410914.jpg","isUpdated":"0",
"defectTypeCount":"1","cell":"plant1,line1,cell1","instanceId":
"ae509ec0-e1d6-4827-afd1-a8af598eb28c_1511419442550",
"defectCount":"1","confidence":"99.99","inspectId":
"auto_1511421410914.jpg","inspectTime":"1511421414687"}]
```


Response items

`id: String`. The inspection ID.

`isConfirmed: Integer`. A 0 means that the inspection result is unconfirmed. A 1 means that the inspection result is confirmed. A 2 means that the inspection result is unknown.

`cell: String`. The cell information.

`inspectFileUrl: String`. The inspection file URL.

`defectTypeCount: Integer`. The number of defect types.

`defectCount: Integer`. The defect number.

`confidence: Double`. The confidence level of the defect.

`inspectId: String`. The inspection ID.

`inspectTime: Time`. The time of inspection.

Sample call

```
curl -k -H
"APIKEY: 8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579
e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112.22
9.91:9447/ibm/iotm/service/inspectResult?user=auto_inspector&solution=vi&isCon
firmed=0&cell=plant1,line1,cell1&sampleRate=10&pageCount=1&reverse=true"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get the inspection result details

Gets the inspection result details. The inspector and the supervisor have authority to use this API.

URL

/ibm/iotm/service/inspectResult/*inspectResultId*

Method

The request type GET

URL parameters

`inspectResultId: String`. Used to identify the inspection ID.

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
    {
        "isConfirmed": "0",
        "inspectFileUrl": "imageserver/edge1/image1.jpg",
        "isUpdated": "0",
        "cell": "plant1,line1,cell1",
        "instanceId": "6e4bfc6b-0ddb-4c85-81e7-0ea789681ae8_775e39b9-fedd-4418-a5b3
-70de6b86fbec",
        "inspectResult": [
            {
                "position": {
                    "height": 331,
                    "width": 467,
                    "y": 1653,
                    "x": 897
                },
                "probableTypes": [
                    {
                        "confidence": 100,
                        "type": "NoDefect"
                    },
                    {
                        "confidence": 0,
                        "type": "Defect"
                    },
```

```
                    {
                        "confidence": 0,
                        "type": "MisT"
                    }
                ]
            }
        ]
    }
]
```

Response items

`isConfirmed: Integer`. A 0 means that the inspection result is unconfirmed. A 1 means that the inspection result is confirmed. A 2 means that the inspection result is unknown.

`inspectFileUrl: String`. The inspection file URL.

`inspectId: String`. The inspection ID.

`inspectResult: JSON Object`. The inspection result.

`position: JSON Array`. The inspection's height, width, y, and x information.

`probableTypes: JSON Object`. Each defect type and its confidence level.

Sample call

```
curl -k -H
"APIKEY: 8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2
c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/service/inspectResult/plant1,line1,
cell1_1511166988402_AUTO|auto_1511166972283.jpg?user=auto_inspector&
solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get the inspection result cell overview

Gets the inspection result cell overview. The inspector and the supervisor have authority to use this API.

URL

/ibm/iotm/service/inspectResultCell

Method

The request type GET

URL parameters

`user: String`. Used to identify the user. Mandatory.

`solution: String`. Used to identify the solution. Mandatory.

`tenant: String`. Used to identify the tenant. Optional.

Headers

APIKEY: *encrypted key*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

None

Success response

```
200
[
    {
```

```
        "confirmed": 1,
        "cell": "plant1,line1,cell1",
        "unknown": 0,
        "unconfirmed": 10
    }
]
```

Response items

`confirmed:` *`Integer`*. A confirmed inspector number.

`unconfirmed:` *`Integer`*. An unconfirmed inspector number.

`unknown:` *`Integer`*, An unknown inspector.

`cell:` *`Strong`*. The image cell.

Sample call

```
curl -k -H
"APIKEY: 8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c7
7d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" "
https://9.112.229.91:9447/ibm/iotm/service/inspectResultCell?user=auto_
inspector&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Confirm inspection results

Confirms the inspection result. The inspector and the supervisor have authority to use this API.

URL

/ibm/iotm/service/inspectResult/*inspectResultId*

Method

The request type PUT

URL parameters

`inspectResultId:` *`String`*. Used to identify the inspector ID.

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

Headers

`Content-type: application/json`.

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

`position:` *`JSON Array`*. The inspection's height, width, y, and x information.

`probableTypes:` *`JSON Object`*. Each defect type and its confidence level.

`logs:` *`JSON Object`*. Contains the confirm action, the operator, and the update time.

`description:` *`String`*. The confirm inspector description.

Sample body

```
[
    {
        "confirmedResult": [
            {
                "position": {
                    "height": 3231,
```

```
                    "width": 467,
                    "y": 1653,
                    "x": 897
                },
                "probableTypes": [
                    {
                        "confidence": 100,
                        "type": "NoDefect"
                    },
                    {
                        "confidence": 0,
                        "type": "Defect"
                    },
                    {
                        "confidence": 0,
                        "type": "MisT"
                    }
                ],
                "logs":[{"action":"confirm",
                "operator":"inspector1",
                "updateTime":"1503657054321"}]
            }
        ],
        "description":"VI result is OK"
    }
]
```

Success response

```
200
[{}]
```

Response items

None

Sample call

```
curl -k -H"APIKEY: 8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8
c149f6aab2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" -H
"Content-Type:application/json;charset=UTF-8" -X PUT "https://9.112.229.91:9447/ibm
/iotm/service/inspectResult/plant1,line1,cell1_1511507025284_AUTO|auto_1511507012702
.jpg?user=auto_inspector&solution=vi"  --data ' [    {  "confirmedResult": [  {
"position": {  "height": 430,    "width": 552,  "x": 1254,         "y": 2123  },
"probableTypes": [  {  "confidence": 99.87,   "type": "Defect"   },     {
"confidence": 0.1,    "type": "MisT"  },   {   "confidence": 0.03,    "type":"NoDefect"
   }      ],    "logs":[{"action":"confirm",   "operator":"auto_super",
"updateTime":"1503657054321"}]   } ],   "description":"VI result is OK"  }]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Delete inspection results

Deletes the inspection results. The inspector and the supervisor have authority to use this API.

URL

/ibm/iotm/service/inspectResult

Method

The request type DELETE

URL parameters

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`tenant:` *`String`*. Used to identify the tenant. Optional.

`cell:` *`String`*. Used to identify the cell. Mandatory.

`inspectTime: Time`. Used to identify the time of the inspection. The inspection results that occur before this time are deleted. Optional.

`instanceId: String`. Used to identify the instance ID. The inspection results that are produced by this instance ID are deleted. Optional.

Headers

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

None.

Success response

```
200
[{}]
```

Response items

None.

Sample call

```
curl -k -H "APIKEY: 8b79658225de53488321fb
7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533dd34
ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
-H "Content-Type:application/json;charset=UTF-8" -X DELETE
"https:// 9.112.229.91:9447/ibm/iotm/service/inspectResult?instanceId=
04b61494-99ec-49f4-9a0f-0125650c40dd_1520316716077&
tenant=Q3T1&solution=vi&user=demoadm@cn.ibm.com&cell=plant2,line2,cell2&inspectTime=
1520324778324"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Model instance action services

The model instance action services enable you to perform such tasks as submitting a model instance, attaching a model file, and validating a model instance.

Train model instance

Train a model instance by using a deep learning model. The model manager has authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*

Method

The request type PUT

URL parameters

`modelInstanceId:String`. The model instance ID. Mandatory.

`action:String`. The action. Set as `train`. Mandatory.

`user:String`. Identifies the user. Mandatory.

`solution:String`. Used to identify the solution. Mandatory.

`tenant:String`. Used to identify the tenant. Optional.

Headers

`Content-Type: application/json`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

```
 [{}]
```

Success response

```
200

[{"status":"waiting","instanceId":
"1e760614-ab0e-4c2a-be2e-435505087fe4_1518060853002",
"parameters":{"modelType":"classification","trainType":
"online_train"}}]
```

Response items

`status: String`. The status of the retraining of the model instance.

`instanceId: String`. Model instance ID.

`updatedTime: Time`. The last time the model instance was updated.

`parameters: JSON object`. Information about the model type and training type.

Sample call

```
curl -k -X PUT -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5af
b8c149f6aab2c77d4f0579e533dd34ggb959317ac69ff73f886fc37
13bf305a158fdf485cc6f275f5" -H "Content-Type:
application/json;charset=UTF-8"
"https://int_iotm.predictivesolutionsapps.ibmcloud.com/
ibm/iotm/service/modelInstance/1e760614-ab0e-4c2a-be2e-
435505087fe4_1518060853002?action=train&tenant=Q3T1&
solution=vi&user=admin@example.com
" --data ' [{}]'
```

Notes

This service prepares images and performs online training. You can check the status of the model instance by using the "Get one model instance" API call.

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Download a training log file

Download a log file for the training of a model. The model manager has authority to use this API.

URL

/ibm/iotm/vi/service/logFile

Method

The request type GET

URL parameters

`instanceId:String`. The model instance ID. Mandatory.

`user:String`. Identifies the user. Mandatory.

`solution:String`. Used to identify the solution. Mandatory.

`tenant:String`. Used to identify the tenant. Optional.

Headers

APIKEY: *encrypted key*. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

None.

Success response

Binary content of the log file.

Response items

None.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d
4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.91:9447/ibm/iotm/vi/service/logFile?instanceId=06c87
0e5-1701-419a-8a84-0f40ee82fe33_11260029297&user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Validate model instance

Validate a trained model instance. The model manager has authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*

Method

The request type PUT

URL parameters

modelInstanceId:*String*. The model instance ID. Mandatory.

action:*String*. The action. Set as validate. Mandatory.

user:*String*. Identifies the user. Mandatory.

solution:*String*. Used to identify the solution. Mandatory.

tenant:*String*. Used to identify the tenant. Optional.

Headers

Content-type: application/json.

Content-Disposition: form-data; name="files[]"; filename="model.zip".

APIKEY: *encrypted key*. The API key, which is used for authentication. Mandatory.

Data parameters

validateData: *JSON object*. Model validation data.

groupId: *String*. The data group ID.

fileIds: *String*. The data file ID.

Sample body

```
[{"validateData": [    {"fileIds": "9f7aa9d6-24d6-4611-8d4d-
79a52d2bab02_76e8b2cb-057a-4904-8fad-bbca93abd80b",    "groupId":
"9f7aa9d6-24d6-4611-8d4d-79a52d2bab02"  },      { "fileIds":
"d84c028e-235a-4c13-8a77-894500b8a736_fe201c83-b138-4dca-b918-ae1b9f7db894",
```

```
"groupId": "d84c028e-235a-4c13-8a77-894500b8a736" },
{  "fileIds": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-44d8-a2fa-
e74c1578eebe",   "groupId": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb"   }  ]   }]
```

Success response

```
200
[{"validateData":[{"fileIds":"9f7aa9d6-24d6-4611-8d4d-79a52d2bab02_76e8b2cb-
057a-4904-8fad-bbca93abd80b","groupId":"9f7aa9d6-24d6-4611-8d4d-79a52d2bab02"},
{"fileIds":"d84c028e-235a-4c13-8a77-894500b8a736_fe201c83-b138-4dca-b918-
ae1b9f7db894","groupId":"d84c028e-235a-4c13-8a77-894500b8a736"},
{"fileIds":"c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-44d8-a2fa-
e74c1578eebe","groupId":"c5f11ed9-db49-403a-972c-cd6cb1d0f8eb"}],"status":
"validating","instanceId":"ef12294e-2d09-4acd-9946-00a09bdeeba2_1512092239371"}]
```

Response items

`validateData:` *`JSON Object`*. Model validation data.

`groupId:` *`String`*. Data group ID.

`fileIds:` *`String`*. Data file ID.

Sample call

```
curl -k -H "APIKEY:8b79658225de53488321fb7bb657f9f161ac
d2ea830a5afb8c149f6aab2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf30
5a158fdf485cc6f275f5" -H "Content-Type:application/json;charset=UTF-8" -X PUT
"https://9.112.229.91:9447/ibm/iotm/service/modelInstance/ef12294e-2d09-
4acd-9946-00a09bdeeba2_1512092239371?action=validate&user=auto_mm&solution=vi"
--data ' [{"validateData": [    {"fileIds": "9f7aa9d6-24d6-4611-8d4d-
79a52d2bab02_76e8b2cb-057a-4904-8fad-bbca93abd80b",    "groupId":
"9f7aa9d6-24d6-4611-8d4d-79a52d2bab02"  },      { "fileIds":
"d84c028e-235a-4c13-8a77-894500b8a736_fe201c83-b138-4dca-b918-ae1b9f7db894",
"groupId": "d84c028e-235a-4c13-8a77-894500b8a736" },
{  "fileIds": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_
bd0c8246-7208-44d8-a2fa-e74c1578eebe",
"groupId": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb"   }  ]   }]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Reject model instance

Reject a validated model instance. The model manager has authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*

Method

The request type PUT

URL parameters

`modelInstanceId:`*`String`*. The model instance ID. Mandatory.

`action:`*`String`*. The action. Set as `reject`. Mandatory.

`user:`*`String`*. Identifies the user. Mandatory.

`solution:`*`String`*. Used to identify the solution. Mandatory.

`tenant:`*`String`*. Used to identify the tenant. Optional.

Headers

`Content-type: application/json`.

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

```
[{}]
```

Success response

```
200
[
{
"status": "rejected",
"instanceId": "e23b6796-4f4a-4c5d-8fb7-fa3f729f591c_1510812725969"
}
]
```

Response items

`status: String`. Model instance status.

`instanceId: String`. Model instance ID.

Sample call

```
curl -k -H "APIKEY:8b79658225de53488321fb7bb657f9f
161acd2ea830a5afb8c149f6aab2c77d4f0579e533dd34ggb959317ac69ff73f886
fc3713bf305a158fdf485cc6f275f5" -H "Content-Type:application/json;
charset=UTF-8" -X PUT "https://9.112.229.91:9447/ibm/iotm/service/
modelInstance/ef12294e-2d09-4acd-9946-00a09bdeeba2_1512092239371?
action=reject&user=auto_mm&solution=vi"  --data ' [{}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Deploy model instance

Deploy an accepted model instance. The model manager has authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*

Method

The request type PUT

URL parameters

`modelInstanceId:String`. The model instance ID. Mandatory.

`action:String`. The action. Set as `publish`. Mandatory.

`user:String`. Identifies the user. Mandatory.

`solution:String`. Used to identify the solution. Mandatory.

`tenant:String`. Used to identify the tenant. Optional.

`Parameters:JSON Object`. The ID of the edges the model will be deployed on.

Headers

`Content-type: application/json`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

```
[{"parameters": {"edges":[{"id":"1508476898000"},{"id":"1542102975873"},...]}}]
```

Success response

```
[{"status":"deployed","instanceId":"96b651a9-2ba0-4392-
aca4-1463320a19c2_1542354262146"}]
```

Response items

`status:` *`String`*. Model instance status.

`instanceId:` *`String`*. Model instance ID.

`parameters:` *`JSON Object`*. The name and the IP address of the edge.

`modelType:` *`String`*. The model type.

Sample call

```
curl -k -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab
2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
-H "Content-Type:application/json;charset=UTF-8" -X PUT
"https://9.112.229.91:9447/ibm/iotm/service/modelInstance/ef12294e-
2d09-4acd-9946-00a09bdeeba2_1512092239371?action=publish&user=
auto_mm&solution=vi" --data '[{“parameters": {"edges":[{"id":
"1508476898000"},{"id":"1542102975873"},...]}}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Retrain model instance

Retrain a deployed model instance. The model manager has authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*

Method

The request type PUT

URL parameters

`modelInstanceId:`*`String`*. The model instance ID. Mandatory.

`action:`*`String`*. The action. Set as `retrain`. Mandatory.

`user:`*`String`*. Identifies the user. Mandatory.

`solution:`*`String`*. Used to identify the solution. Mandatory.

`tenant:`*`String`*. Used to identify the tenant. Optional.

Headers

`Content-Type: application/json`.

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

`retrainData:`*`JSON_object`*. The retraining data for the model.

`groupId:`*`String`*. The data group ID.

`fileIds:`*`String`*. The data file ID.

Sample body

```
 [{"trainData": [    {"fileIds": "9f7aa9d6-24d6-4611-
8d4d-79a52d2bab02_76e8b2cb-057a-4904-8fad-bbca93abd80b",
"groupId": "9f7aa9d6-24d6-4611-8d4d-79a52d2bab02"  },
{ "fileIds": "d84c028e-235a-4c13-8a77-894500b8a736_
fe201c83-b138-4dca-b918-ae1b9f7db894",
"groupId": "d84c028e-235a-4c13-8a77-894500b8a736" },
{  "fileIds": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_
bd0c8246-7208-44d8-a2fa-e74c1578eebe",
"groupId": "c5f11ed9-db49-403a-972c-cd6cb1d0f8eb"
   }  ]   }]
```

Success response

```
200
[{"modelId":"ef12294e-2d09-4acd-9946-00a09bdeeba2","status":
"waiting","instanceId":"ef12294e-2d09-4acd-9946-00a09bdeeba2_
1512095854903","trainData":[{"fileIds":"9f7aa9d6-24d6-4611-
8d4d-79a52d2bab02_76e8b2cb-057a-4904-8fad-bbca93abd80b",
"groupId":"9f7aa9d6-24d6-4611-8d4d-79a52d2bab02"},
{"fileIds":"d84c028e-235a-4c13-8a77-894500b8a736_
fe201c83-b138-4dca-b918-ae1b9f7db894","groupId":
"d84c028e-235a-4c13-8a77-894500b8a736"},{"fileIds":
"c5f11ed9-db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-
44d8-a2fa-e74c1578eebe","groupId":"c5f11ed9-db49-403a-
972c-cd6cb1d0f8eb"}]}]
```

Response items

`modelId:` *`String`*. Model ID.

`status:` *`String`*. The status of the retraining of the model instance.

`instanceId:` *`String`*. Model instance ID.

`trainData:` *`JSON Object`*. Model training data.

`groupId:` *`String`*. The ID of the data group.

`fileIds:` *`String`*. The ID of the data file.

Sample call

```
curl -k -H "APIKEY:8b79658225de53488321fb
7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533dd
34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" -H
"Content-Type:application/json;charset=UTF-8" -X PUT
"https://9.112.229.91:9447/ibm/iotm/service/modelInstance/
ef12294e-2d09-4acd-9946-00a09bdeeba2_1512092239371?
action=retrain&user=auto_mm&solution=vi"  --data
'[{"trainData": [    {"fileIds": "9f7aa9d6-24d6-
4611-8d4d-79a52d2bab02_76e8b2cb-057a-4904-8fad-
bbca93abd80b",    "groupId": "9f7aa9d6-24d6-4611-
8d4d-79a52d2bab02"  },      { "fileIds": "d84c028e-
235a-4c13-8a77-894500b8a736_fe201c83-b138-4dca-b918-
ae1b9f7db894",        "groupId": "d84c028e-235a-4c13-
8a77-894500b8a736" },      {  "fileIds": "c5f11ed9-
db49-403a-972c-cd6cb1d0f8eb_bd0c8246-7208-44d8-a2fa-
e74c1578eebe",   "groupId": "c5f11ed9-db49-403a-972c-
cd6cb1d0f8eb"   }  ]   }]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Undeploy model instance

Undeploy a deployed model instance. The model manager has authority to use this API.

URL

/ibm/iotm/service/modelInstance/*modelInstanceId*

Method

The request type PUT

URL parameters

`modelInstanceId:`*`String`*. The model instance ID. Mandatory.

`action:`*`String`*. The action. Set as `undeploy`. Mandatory.

`user:`*`String`*. Identifies the user. Mandatory.

`solution:`*`String`*. Used to identify the solution. Mandatory.

`tenant:`*`String`*. Used to identify the tenant. Optional.

`Parameters:`*`JSON Object`*. The ID of the edge from which the model instance will be undeployed.

Headers

`Content-Type: application/json`.

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

```
[{"parameters": {"edges":[{"id":"1508476898000"}]}}]
```

Success response

```
200
[{"status":"undeployed","instanceId":"ef12294e-2d09-4acd-9946-
00a09bdeeba2_1512092239371"}]
```

Response items

`status:` *`String`*. The status of the retraining of the model instance.

`instanceId:` *`String`*. Model instance ID.

Sample call

```
curl -k -H "APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830
a5afb8c149f6aab2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6
f275f5" -H "Content-Type:application/json;charset=UTF-8"
-X PUT "https://9.112.229.91:9447/ibm/iotm/service/modelInstance/ef12294e-2d09-
4acd-9946-00a09bdeeba2_1512092239371?action=undeploy&user=auto_mm&solution=vi"
--data '[{“parameters": {"edges":[{"id":"1508476898000"}]}}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Edge services

The edge services enable you to perform such tasks as creating an edge, getting an edge, and updating an edge.

Create edge

Create an edge. The model manager has authority to use this API.

URL

/ibm/iotm/vi/service/edge

Method

The request type POST

URL parameters

`user:String`. Identifies the user. Mandatory.

`solution:String`. Used to identify the solution. Mandatory.

`tenant:String`. Used to identify the tenant. Optional.

Headers

`Content-Type: application/json`.

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

`name:String`. The name of the edge. Mandatory.

`ip:IP`. The IP address of the edge. Mandatory.

`port:integer`. The port of the edge. Mandatory.

`userName:String`. The user name for the edge virtual machine. Mandatory.

`passWord:String`. The password for the edge virtual machine. Mandatory.

`clusterType:String`. The cluster type of the edge. Specify `slave` or `master`. Mandatory.

`online:String`. Indicates whether the edge is online. Specify `true` if the edge is online or `false` if the edge is offline. Mandatory.

Sample body

```
[{"name":"edge_auto","ip":"9.112.229.54",
"port":"22","userName":"root","passWord":"U2FtcGxlQDY2NiE=",
"clusterType":"mater","online":"true"}]
```

Success response

```
 [{"port":"22","clusterType":"master","deployPath":"\/home\/pmqopsadmin","hierarchyList":
["plant
1,line1,cell1","plant2,line2,cell2"],"servicePass":"passw0rd@","deletePolicy":
{"schedule":false,"rate
":0.2,"onDay":7,"afterDay":"1","frequency":"weekly"},"version":"2018Q4.01","ip":"viedge1","o
nline
":"true","id":"1508476898000","upgradeFlag":"false","createdBy":"demoadm@cn.ibm.com","servic
eUser":
"admin","name":"master","sshUser":"pmqopsadmin","updatedTime":"2018-11-16T03:54:13.382-06:00
"}]
```

Response items

`id:String`. The ID of the edge.

`deletePolicy:JSON Object`. The policy to delete the inspection result on the edge.

`version:String`. The edge version information.

`upgradeFlag:String`. Indicates whether the edge needs an upgrade.

Sample call

```
curl -k -H "APIKEY:8b79658225d
e53488321fb7bb657f9f161acd2ea830a5afb8c149
f6aab2c77d4f0579e533dd34ggb959317ac69ff
73f886fc3713bf305a158fdf485cc6f275f5" -H
"Content-Type:application/json;charset=UTF-8"
"https://9.112.229.91:9447/ibm/iotm/vi/service/edge?
user=auto_mm&solution=vi "  --data '[{"name":
"edge_auto","ip":"9.112.229.54","port":"22",
```

```
"userName":"root","passWord":"U2FtcGxlQDY2NiE=",
"clusterType":"master","online":"true"}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Get edge

Get an edge. The model manager has authority to use this API.

URL

/ibm/iotm/vi/service/edge

Method

The request type GET

URL parameters

`user:`*`String`*. Identifies the user. Mandatory.

`solution:`*`String`*. Used to identify the solution. Mandatory.

`tenant:`*`String`*. Used to identify the tenant. Optional.

`hierarchyList:`. The cell hierarchy information, for example `plant1, line1, cell1`. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

None.

Success response

```
200
[{"port":"22","id":"1512098793912","clusterType":"slave","hierarchyList":
["plant1, line1, cell1"],"createdBy":"auto_mm","description":"test edge",
"name":"edge_auto","updatedTime":"2017-12-01T10:58:26.559+08:00","online":
"false","ip":"9.112.229.54"}]
```

Response items

`name:`*`String`*. The name of the edge.

`ip:`*`IP`*. The IP address of the edge.

`port:`*`Integer`*. The port of the edge.

`clusterType:`*`String`*. The cluster type of the edge. Specify `slave` or `master`.

`online:`*`String`*. The online status of the edge: true means the edge is online, and false means the edge is offline.

`hierarchyList:`*`String`*. The hierarchy of the edge.

`description:`*`String`*. A description of the edge.

`createdBy:`*`String`*. The owner of the edge.

`updatedTime:`*`Time`*. The last time that the edge was updated.

Sample call

```
curl -k -H "APIKEY:8b79658225de53488321fb7bb657
f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533dd34ggb959317ac69ff
```

```
73f886fc3713bf305a158fdf485cc6f275f5" "https://9.112.229.91:9447/ibm/iotm/vi/
service/edge?user=auto_mm&solution=vi"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Delete edge

Delete an edge. The model manager has authority to use this API.

URL

/ibm/iotm/vi/service/edge/*edgeId*

Method

The request type DELETE

URL parameters

`edgeId:`. The edge ID. Mandatory.

`user:`*`String`*. Identifies the user. Mandatory.

`solution:`*`String`*. Used to identify the solution. Mandatory.

`tenant:`*`String`*. Used to identify the tenant. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

`[{}]`

Success response

```
200
[{}]
```

Response items

None.

Sample call

```
curl -k -X DELETE -H "APIKEY:8b7965
8225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab
2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a
158fdf485cc6f275f5" "https://9.112.229.91:9447/ibm/iotm/
vi/service/edge/1512098793912?user=auto_mm&solution=
vi" -d '[{}]'
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Upgrade edge

Upgrade an edge. Users with the authority to create an edge have authority to use this API.

To upgrade an edge, you must use an API call get the needed edge files, including `edgeDeployed.sh` and `edge-vi.zip`.

To upgrade an edge on the edge side, the SSH user specified when creating the edge must execute the shell script on the edge system.

URL

/ibm/iotm/vi/service/edgeFile

Method

The request type GET

URL parameters

`edgeID:` The ID of the edge. Mandatory.

`user:`*`String`*. Identifies the user. Mandatory.

`solution:`*`String`*. Used to identify the solution. Mandatory.

`tenant:`*`String`*. Used to identify the tenant. Optional.

`Version:`*`String`*. The value can be `shell` for a shell script download, or `ubuntu`, `redhat`, or `power` for the type of Linux on the edge system.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Success response

```
200
```

The `edgeDeployed.sh` file or the `vi_edge-bin_vi.zip` file.

Response items

None

Sample call

```
curl -k -X GET -H "APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb
8c149f6aab2c77d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://int_iotm.predictivesolutionsapps.ibmcloud.com/ibm/iotm/vi/service/edgeFile?tenant=
Q3T1&solution=vi&user=modelmanager1&version=shell&edgeId=1508476898000”
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Score service

The score service enables you to score an image.

Score image

Score an image. The inspector and supervisor have authority to use this API.

URL

/ibm/iotm/vi/service/uploadScoreImage

Method

The request type POST

URL parameters

`productType:`*`String`*. The product type of the model. Mandatory.

`cell:`*`String`*. The cell of the image. Mandatory.

`user:`*`String`*. Identifies the user. Mandatory.

`solution:String`. Used to identify the solution. Mandatory.

`tenant:String`. Used to identify the tenant. Optional.

Headers

`Content-Type: application/binary`

`APIKEY: encrypted key`. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

Binary content of the image file.

Success response

```
200
//for object detection example:
{
  "detections": [
    {
      "position": {
        "height": 60,
        "width": 253,
        "x": 1594,
        "y": 773
      },
      "probableTypes": [
        {
          "confidence": 99.798703193664551,
          "type": "scratch"
        }
      ],
      "properties": []
    }
  ],
  "timestamp": "2017-09-28 10:32:13.528415"
}

//for classification example:
{
  "Timestamp": "2017-09-12 18:43:52",
  "detections": [
    {
      "position": {
        "height": 436,
        "width": 537,
        "x": 1574,
        "y": 1588
      },
      "probableTypes": [
        {
          "confidence": 100.0,
          "type": "Defect"
        },
        {
          "confidence": 0.0,
          "type": "MisT"
        },
        {
          "confidence": 0.0,
          "type": "NoDefect"
        }
      ]
    }
  ]
}
```

Response items

`position:JSON Array`. The inspection height, width, x, and y information.

`probableTypes:JSON Object`. The type and confidence level of each defect.

`Timestamp:Time`. The time that the image is scored.

Sample call

```
curl -k -X POST -H "APIKEY:8b79658225de53488321
fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533dd34ggb9593
17ac69ff73f886fc3713bf305a158fdf485cc6f275f5"  -H
"Content-Type: application/binary" "https://9.112.229.91:9447/
ibm/iotm/vi/service/uploadScoreImage?user=auto_super&productType=
auto&cell=plant1_line1_cell1&solution=vi"  --connect-timeout 600
--data-binary @C:\\code\\Automation_98\\API\\vi\\VI_API\\file\\test.JPG
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

QEWS integration service

The QEWS integration service enables you to get a defect image rate file.

Get defect image rate file

Get a file that contains data about the defect image rate. The supervisor has authority to use this API.

URL

/ibm/iotm/service/defectImageRateFile

Method

The request type GET

URL parameters

`user:`*`String`*. Identifies the user. Mandatory.

`solution:`*`String`*. Used to identify the solution. Mandatory.

`tenant:`*`String`*. Used to identify the tenant. Optional.

`date from`. The start date in yyyy-MM-dd or yyyy-MM-dd HH:mm:ss format. Optional.

`date to`. The end date in yyyy-MM-dd or yyyy-MM-dd HH:mm:ss format. Optional.

`cell name`. The name of the cell. Optional.

`interval`. The interval of the defect image rate. Specify `daily`, `hourly`, or `monthly`. Optional.

Headers

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None.

Sample body

None.

Success response

200. The binary content of the data file. The data file is a comma-separated value (CSV) file.

Response items

`ATTRIBUTE_NAME`: `VI_DEFECT_IMAGE_RATE`, a fixed value.

`DATE`: The date yyyy-MM-dd format when the interval is daily and monthly, and yyyy-MM-dd HH:mm:ss format when the interval is hourly.

`CELL_NAME`: The name of the cell. In this response item, the comma character (,) is replaced with the underscore character (_) because commas are not supported.

Sample calls

```
curl -k -H "APIKEY:8b79658225de53488
321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533
dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.89:9447/ibm/iotm/service/
defectImageRateFile?user=supervisor1&solution=
vi&tenant=T1" --output "C:\Users\IBM_ADMIN\Downloads\a.csv"
```

```
curl -k -H "APIKEY:8b79658225de53488
321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d4f0579e533
dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"
"https://9.112.229.89:9447/ibm/iotm/service/
defectImageRateFile?user=supervisor1&solution=
vi&tenant=T1&dateFrom=2017-10-23%2011:00:00&dateTo=
2017-11-14%2018:40:00&cellName=plant1,line1,cell1&
interval=hourly" --output "C:\Users\IBM_ADMIN\
Downloads\cell1_hourly.csv"
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Composite service

The composite service enables you to get a compressed file that contains a trained model and register the model to Maximo PQI On-Premises Visual Insights.

Register model

Gets a compressed file that contains a trained model and registers the model to Maximo PQI On-Premises Visual Insights. The model manager has authority to use this API.

URL

/ibm/iotm/vi/service/registerModel

Method

The request type POST

URL parameters

`user:` *`String`*. Used to identify the user. Mandatory.

`solution:` *`String`*. Used to identify the solution. Mandatory.

`name:` *`String`*. The model name.

`productType:` *`String`*. The product type that is associated with the model.

Headers

`Content-Type: multipart/form-data`.

`Content-Disposition: form-data; name="files[]";filename="model.zip"`.

`APIKEY:` *`encrypted key`*. The API key, which is used for authentication. Mandatory.

Data parameters

None

Sample body

Binary content of the model file.

Success response

```
200
```

```
{"errorMessage":"","modelId":"3914ad2c-3b7b-47ad-974c-e5b6bc2075ef",
"createdBy":"modelmanager1","status":"validated","instanceId":
"3914ad2c-3b7b-47ad-974c-e5b6bc2075ef_1517796906688",
"updatedTime":"2018-02-05 10:15:06.608"}
```

Response items

`modelId:` *`String`*. The model ID.

`createdBy:` *`String`*. The model owner.

`status:` *`String`*. The status of the model instance.

`instanceId:` *`String`*. The ID of the model instance.

`updatedTime:` *`Time`*. The last time the model instance was updated.

Sample call

```
curl -k -X POST -H
"APIKEY:8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77
d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5" -H
"Content-Type:multipart/form-data" -H "Content-Disposition: form-data;
name="files[]"; filename="walkermodel1.zip""
"https://int_iotm.predictivesolutionsapps.ibmcloud.com/ibm/iotm/vi/
service/registerModel?user=demoadm@cn.ibm.com&solution=vi&name=
samplename&productType=sampletype" --connect-timeout 6000 -F
file=@walkermodel.zip
```

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Stand-alone edge services

The stand-alone edge services enable you to perform such tasks as getting available models, deploying models, uploading image scores, syncing inspection results, and cleaning inspection results.

The stand-alone edge services run on the stand-alone edge machine and not the Maximo PQI On-Premises Visual Insights center. You must use different hosts, ports, and user credentials.

Get available models

Gets all available models. The stand-alone edge administrator has authority to use this API.

URL

https://<edge machine host>:8449/api/getAvailableModels

Method

The request type GET

URL parameters

None.

Headers

`Authorization:` *`user_name/password`*

Data parameters

None.

Sample body

None.

Success response

```
{"model_list": [
        {
            "coviacenter": {
                "apikey": "8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77d
```

```
4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5",
                "credential": "ZGVtb2FkbUBjbi5pYm0uY29tOlEzVDE=",
                "url": "https://int_iotm.predictivesolutionsapps.ibmcloud.com/ibm/iotm/
service/apiWrapper?apiName=addInspectResult",
                "username": "demoadm@cn.ibm.com"
            },
            "groupIds": "0c4f11d7-b255-4c05-9c87-694e9f912fa9",
            "model_id": "049dabe5-d865-4f09-862f-fb7430afef6d",
            "model_instance_id": "049dabe5-d865-4f09-862f-fb7430afef6d_1518401501376",
            "model_type": "classification",
            "model_url": "049dabe5-d865-4f09-862f-fb7430afef6d_1518401599389.zip",
            "product_type": "Q3T1|jianyuan45",
            "status": "deployed"
        },
        …...
]}
```

Response items

`model_list`: *String*. The list of published models.

`model_id`: *String*. The model ID of the published model.

`model_instance_id`: *String*. The model instance ID of the published model.

`model_type`: *String*. The model type of the published model.

`model_url`: *String*. The model URL of the published model.

`product_type`: *String*. The product type of the published model.

`coviacenter`: *JSONObject*. The required center information for the edge.

Sample call

```
curl -k -H 'Authorization: Basic YWRtaW46cGFzc3cwcmRA' -X GET https://{localhost}
:8449/api/getAvailableModel
```

Deploy a model

Deploys one model. The stand-alone edge administrator has authority to use this API.

URL

https://<edge machine host>:8449/api/deployModel

Method

The request type POST

URL parameters

None.

Headers

`Authorization: user_name/password`

Data parameters

`model_list`: *String*. The list of published models.

`model_id`: *String*. The model ID of the published model.

`model_instance_id`: *String*. The model instance ID of the published model.

`model_type`: *String*. The model type of the published model.

`model_url`: *String*. The model URL of the published model.

`product_type`: *String*. The product type of the published model.

`coviacenter`: *JSONObject*. The required center information for the edge.

`edgeId`: *String*. The ID of the current edge.

Sample body

```
{"model_id": "56608a3d-df64-4ef0-85a6-f88778cf583b","model_url": "56608a3d
-df64-4ef0-85a6-f88778cf583b_1540279487176.zip","edgeId": "1540275842203",
"coviacenter": {"url": "https://int_iotm.predictivesolutionsapps.ibmcloud.com/
ibm/iotm/service/apiWrapper?apiName=addInspectResult","credential":
"c3VwZXJ2aXNvcjE6UTNUMQ==","username": "supervisor1","apikey":
"8b79658225de53488321fb7bb657f9f161acd2ea830a5afb8c149f6aab2c77
d4f0579e533dd34ggb959317ac69ff73f886fc3713bf305a158fdf485cc6f275f5"},"groupIds":
"b18a8d56-7a4c-4266-9280-49258c3fd798","model_type": "classification",
"product_type": "Q3T1|qwtest12","model_instance_id": "56608a3d-df64
-4ef0-85a6-f88778cf583b_1540279416752","model_name": "qwtest12"}
```

Success response

```
{ “status”:“success”}
```

Response items

`Status`:*String*. Success or failed.

Sample call

```
curl -k -H 'Authorization: Basic YWRtaW46cGFzc3cwcmRA'
-H "Content-Type:application/json;charset=UTF-8" -X POST --data
'{"model_id": "56608a3d-df64-4ef0-85a6-f88778cf583b", "model_url":
"56608a3d-df64-4ef0-85a6-f88778cf583b_1540279487176.zip", "edgeId":
"1540275842203", "coviacenter": { "url": "https://int_iotm.
predictivesolutionsapps.ibmcloud.com/ibm/iotm/service/apiWrapper?
apiName=addInspectResult", "credential": "c3VwZXJ2aXNvcjE6UTNUMQ==",
"username": "supervisor1", "apikey": "8b79658225de53488321fb7bb657f9f161
acd2ea830a5afb8c149f6aab2c77d4f0579e533dd34ggb959317ac
69ff73f886fc3713bf305a158fdf485cc6f275f5" }, "groupIds": "b18a8d56-7a4c-
4266-9280-49258c3fd798", "model_type": "classification", "product_type":
"Q3T1|qwtest12", "model_instance_id": "56608a3d-df64-4ef0-85a6-f88778cf5
83b_1540279416752", "model_name": "qwtest12"} '   https://{localhost}:
8449/api/deployModel
```

Undeploy model

Undeploys a model. The stand-alone edge administrator has authority to use this API.

URL

https://<edge machine host>:8449/api/undeployModel

Method

The request type POST

URL parameters

None.

Headers

`Authorization: user_name/password`

Data parameters

`model_id`: *String*. The model ID of the undeployed model.

`model_instance_id`: *String*. The model instance ID of the undeployed model.

Sample body

```
{"model_id":"5868dcd7-7032-4101-9ba8-797dea9005aa","model_instance_id":"5868dcd7-
7032-4101-9ba8-797dea9005aa_1523253620387"}
```

Success response

```
{“status”:“success”}
```

Response items

`Status:`*`String`*. Success or failed.

Sample call

```
curl -k -H 'Authorization: Basic YWRtaW46cGFzc3cwcmRA'
 -H "Content-Type:application/json;charset=UTF-8" -X POST --data
 '{"model_id": "8e474fb5-97dd-4f5d-8415-14c72855c6a5","model_instance_id":
 "8e474fb5-97dd-4f5d-8415-14c72855c6a5_1532404443311"}'  https://{local
host}:8449/api/undeployModel
```

Upload and score image on the edge

Uploads and scores an image to the edge. The production line or external services typically use this API.

URL

/api/uploadScoreImage

Method

The request type POST

URL parameters

`productType:` *`String`*. The product type of the model. Mandatory.

`cell:` *`String`*. The cell of the image. Mandatory.

Headers

`Content-Type: application/binary`.

`Authorization:`. The base authorization code for the edge service. Mandatory.

Data parameters

None.

Sample body

Binary content of the image file.

Success response

Object detection example:

```
200
{
  "detections": [
    {
      "position": {
        "height": 60,
        "width": 253,
        "x": 1594,
        "y": 773
      },
      "probableTypes": [
        {
          "confidence": 99.798703193664551,
          "type": "scratch"
        }
      ],
      "properties": []
    }
  ],
  "timestamp": "2017-09-28 10:32:13.528415"
}
```

Classification example:

```
200
{
  "Timestamp": "2017-09-12 18:43:52",
  "detections": [
```

```
    {
      "position": {
        "height": 436,
        "width": 537,
        "x": 1574,
        "y": 1588
      },
      "probableTypes": [
        {
          "confidence": 100.0,
          "type": "Defect"
        },
        {
          "confidence": 0.0,
          "type": "MisT"
        },
        {
          "confidence": 0.0,
          "type": "NoDefect"
        }
      ]
    }
  ]
}
```

Sample call

```
curl -k --cert /home/pmqopsadmin/vi_edge-bin_vi/vi_edge/https/cert.pem --key
 /home/pmqopsadmin/vi_edge-bin_vi/vi_edge/https/key.pem   -H 'Authorization:
 Basic auth' -T 'filePath' -X POST https://localhost
:8449/api/uploadScoreImage?productType=productType\&cell=cell
```

The *auth* value is the base authorization to access the edge service. The *filePath* value is the path of the file that you want to upload. The *localhost* value is the IP address of the system where the edge is deployed. The *productType* value is the product type of the model that you used to score the image that you uploaded. The *cell* value is the cell you want the score result sent to.

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Sync inspection result from the edge to the center application

Syncs the inspection result from the edge to the center application. The stand-alone edge administrator has authority to use this API.

URL

/api/syncInspectResult

Method

The request type POST

URL parameters

None

Headers

`Content-Type: application/json`.

`Authorization:`. The base authorization code for the edge service. Mandatory.

Data parameters

None.

Sample body

```
{
“percentage”:100
}
```

Success response

```
200
{
“message”: “success”
}
```

Sample call

```
curl -k --cert /home/pmqopsadmin/vi_edge-bin_vi/vi_edge/https/cert.pem --key
/home/pmqopsadmin/vi_edge-bin_vi/vi_edge/https/key.pem -H "Content-Type: application/json"
-H 'Authorization: Basic YWRtaW46cGFzc3cwcmRA' -X POST --data '{"percentage":percentage}'
https://localhost:8449/api/syncInspectResult
```

The *percentage* value is the percentage of inspection results that you want to send. The *localhost* value is the IP address to which the edge is deployed.

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Clean up inspection result that you have synced to the center application

Cleans up an inspection result that you have synced to the center application. The stand-alone edge administrator has authority to use this API.

URL

/api/cleanInspectResult

Method

The request type POST

URL parameters

None

Headers

`Content-Type: application/json`.

`Authorization:`. The base authorization code for the edge service. Mandatory.

Data parameters

None.

Sample body

```
{
“fromDayBefore”:1
}
```

Success response

```
{
“message”: “success”
}
```

Sample call

```
curl -k --cert /home/pmqopsadmin/vi_edge-bin_vi/vi_edge/https/cert.pem
--key /home/pmqopsadmin/vi_edge-bin_vi/vi_edge/https/key.pem -H "Content-Type:
application/json"  -H 'Authorization: Basic YWRtaW46cGFzc3cwcmRA' -X POST --data
'{"fromDayBefore":dayNumber}' https://localhost:8449/api/cleanInspectResult
```

The *dayNumber* value is the number of days of data that you want to keep. For example:, if you set *dayNumber* to 0 all data before today is removed. If you set *dayNumber* to 1, all data before yesterday is removed. The *localhost* value is the IP address to which the edge is deployed.

Notes

The system uses the APIKEY to do authentication. If you do not provide an APIKEY, the system refuses your request.

Chapter 10. Registering, deploying, and testing a model by using the API

The API Guidance feature is a web interface that guides you through one common API use case: the process of registering, deploying, and testing a trained model. The API Guidance feature automates many steps of the process to make it easier to complete the tasks.

About this task

The first step is registering a model. You can import a compressed file that contains a trained model and register it to the product.

The second step is deploying the model. You can deploy a model to edge systems so the model can be used to inspect images.

The third step is testing the model. You can use your own images to test the deployed model and review the scoring result.

Procedure

1. Select **API Guidance** from the pull-down menu.
2. Click **Try Register Model** to enter the **End to end API guidance** page.
3. Enter the model name and product type.
4. Click **Select** to select a compressed file that contains a trained model.
5. Click **Register** to register the model to the product.
   The registration result is displayed in the **Result** box.
6. Click **Next**.
7. If necessary, enter the instance ID.
8. Click **Deploy** to deploy the model.
   The deployment result is displayed in the **Result** box.
9. Click **Next**.
10. Enter the product type and cell information, select an image, and then click **Score** to start the scoring process.
    The scoring result is displayed in the **Result** box.

Chapter 11. Troubleshooting

Review log files and error messages to assist in troubleshooting the product.

Log files

Refer to the product log files for more information about error conditions.

Log file	Description
/*Liberty_path*/usr/servers/VICenterServer/logs/messages.log	WebSphere Application Server Liberty messages
/home/*user_name*/edgeDeploy_*tenant name*.log	Edge messages
/home/*user_name*/vi_edge-bin_vi/vi_task_manager/master.log	Task manager messages
/home/*user_name*/vi_edge-bin_vi/vi_edge/Service.log	Service messages

Messages

Edge messages

The following messages appear during error conditions for edge systems.

Table 4. Error messages for edge systems

Message	Action
The master edge already exists, only one master edge is supported.	Specify an edge name that is not used.
Required fields must not be empty.	Include a value in all required fields.
Cannot connect to the machine. Use the correct IP address, SSH user name, and password.	Verify that the IP address, SSH user name, and password are valid values.
To manage the edge, you must have the correct permissions.	Ensure that you are logged in as a user with model manager permissions. If you cannot resolve the issue, contact a system administrator.
The IP address is invalid. Input the correct IP address, for example, 10.172.0.23.	Verify the value of the IP address.
The edge was not created due to a database error.	Check the Apache HBase service from the Ambari console.
An edge with the same IP address exists.	You cannot create two edge systems on the same computer. Use the IP address of another computer to create a new edge.
Failed to upload scripts to the edge machine. Check the existence and the permissions of the deployment path.	Verify that the deployment path exists and has correct permissions.
Failed to start services on edge machine.	Make sure that the service is running on the master edge. Check the log file on the edge system.

Table 4. Error messages for edge systems (continued)	
Message	**Action**
Failed to execute scripts on the edge machine.	Check the log file for the edge system.
The edge name is a duplicate. Use a different edge name.	Use a different name to create a new edge.
The slave edge cannot be created if a master edge does not exist.	You must create a master edge before you create a slave edge.
Failed to delete the edge because it is used by model *model_name*.	The edge cannot be deleted if it is used by a model version. Stop all the model versions that are running on the edge before you delete the edge.
The edge was not deleted due to a database error.	Check the Apache HBase service from the Ambari console.
You cannot delete the master edge because there are existing slave edges.	The edge cannot be deleted if it still has slave edges. Delete all slave edges before you delete the master edge.
Failed to delete the edge from master repository: *repository_name*.	Make sure that the service is running on the master edge. Check the service log files.

Compressed image file messages

The following messages appear during error conditions for compressed image files.

Table 5. Error messages for compressed image files	
Message	**Action**
The compressed file that you uploaded does not contain an image at the expected location.	The uploaded compressed image file must contain images. All of the images in the compressed image file must be in a flat structure with no subdirectories.
The file is invalid. You must upload .zip files only.	The only supported file format is .zip.
The annotation XML files are missing in the uploaded compressed file.	Add the annotation XML files to the compressed image file and then upload it again.
The labels.txt is missing in the uploaded compressed file.	Add the `labels.txt` file to the compressed image file and then upload it again.
Input is invalid. You must specify the ID.	No file ID was provided in the API request. Provide the file ID.
Input is invalid. A maximum number of 100 entities can be deleted.	You cannot delete more than 100 compressed files at one time.
Failed to delete the data file.	The data file could not be deleted.
The specified data file was cited in the model *model_name*. It cannot be deleted.	The compressed image file is referenced by a model. It cannot be deleted.

Image group messages

The following messages appear during error conditions for image groups.

Table 6. Error messages for image groups	
Message	**Action**
The group name cannot be empty when you create a new data group.	Specify a name for the new group.
Data files exist for the selected data group. The update failed. Ensure that the selected data group contains no data files.	If the image group is created with data files, the image group type cannot be updated. If the image group is not created with data files, this error might be caused by an Apache HBase operation error.
Failed to create a data group because parameters are missing or have an invalid format.	Ensure that parameters are specified for the new data group and that the parameters are in a valid format.
The group name is a duplicate. Use a different group name.	Specify a unique name for the new group.
Error when creating data group.	An Apache HBase error occurred while the data group was being created. Check the Apache HBase service from the Ambari console.
The data group does not exist.	The group ID in the request is missing or not valid.
Error when deleting data files in a specified data group.	A Hadoop Distributed File System (HDFS) error occurred while data files were being deleted.
The specified data group was cited in a model. It cannot be deleted.	If a group is used in a model, the group cannot be deleted.
Error when deleting data group.	An Apache HBase error occurred while the data group was being deleted. Check the Apache HBase service from the Ambari console.

Model messages

The following messages appear during error conditions for models.

Table 7. Error messages for models	
Message	**Action**
The model name cannot be empty when create a new model.	Specify a name for the model.
The model name must be fewer than 128 characters and only contain letters, numbers, spaces, and underscores.	Specify a valid model name.
The product type cannot be empty when create a new model.	Specify the product type.
The product type must be fewer than 128 characters and only contain letters, numbers, and spaces.	Specify a valid product type.
The data group IDs cannot be empty when create a new model.	Specify a value for `groupIds` in the API call.
The data format cannot be empty when you create a new model.	Specify a value for `dataFormat` in the API call.
The retrain policy must have a valid format when you create a new model.	Specify a valid value for `retrainPolicy` in the API call.

Table 7. Error messages for models (continued)	
Message	**Action**
The model type cannot be empty when create a new model.	Specify a value for `modelType` in the API call.
The model type must be classification or objectdetection.	Specify a value of `classification` or `objectdetection` for `modelType` in the API call.
The parameters must have a valid format when you create a new model.	Specify a valid value for `parameters` in the API call.
Failed to create the model because the trainParam value in the parameters has an invalid value.	Specify a valid value for `trainParam` in the API call.
The IDs of data group are invalid.	Specify a valid value for `groupIds` in the API call.
Error occurred because the data groups must be all multiple characteristic types or all single characteristic types.	In the API call, the groups in `groupIds` must have same group type.
Failed to create model.	An Apache HBase error occurred during the creation of the model. Check the Apache HBase service from the Ambari console.
Input is invalid. A maximum number of 100 entities can be deleted.	Specify fewer than 100 models to be deleted.
Input is invalid. You must specify the ID.	Include the model ID in the request.
The model cannot be updated because the model ID is empty. Enter a model ID.	Include the model ID in the request.
Failed to update model groups because the model version is not in draft status.	Ensure that the models to be updated are in draft status.
Failed to update model groups because the isHybrid value of the data groups is different.	You cannot change the model type after the model is created.

Model instance messages

The following messages appear during error conditions for model instances.

Table 8. Error messages for model instances	
Message	**Action**
The model ID cannot be empty when create a new model instance.	Create a model instance after its corresponding model is created.
The train data cannot be empty when create a new model instance.	Specify a valid value for `trainData` in the API call.
Failed to register the model to the training server.	Check the service that is running on the training server.
Failed to create the model because the trainParam value in the parameters has an invalid value.	Specify a valid value for `trainParam` in the API call.
Training data is invalid. At least one data group must exist.	Ensure that each data group that belongs to the instance has valid data files.
Training data is invalid. Check the train and validation ratio to ensure that at least one image exists in the train and test set for each data group.	Ensure that at least one image exists in the train and test sets for each data group.

Table 8. Error messages for model instances (continued)

Message	Action
You have *n* models in the training queue. You cannot submit a new train request until they are completed or canceled.	Cancel jobs or wait until the jobs that are currently running are finished.
Failed to cancel the training.	Check the Apache HBase service from the Ambari console.
The input body is invalid.	Check your request input and then retry the operation.
The *model_name* model had a status roll back due to a validation timeout.	Check the log on the validation server. For the classification model, the log is `/home/pmqopsadmin/vi_edge-bin_vi/vi_score_engine_restful/back.log`. For the object detection model, the log is `/home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/frcnn_log.txt` or `ssd_log.txt`. Resolve any issues and then retry the operation.
Failed to validate the model version.	Check the log on the validation server. For the classification model, the log file is `/home/pmqopsadmin/vi_edge-bin_vi/vi_score_engine_restful/back.log`. For the object detection model, the log file is `/home/pmqopsadmin/vi_obj_detection_retrain/RESTAPI/model/frcnn_log.txt` or `ssd_log.txt`. Resolve any issues and then retry the operation.
The model does not exist.	No model with this model ID was found. Check the model ID.
Failed to reject the model version.	Check the Apache HBase service from the Ambari console.
Failed to deploy the model version because the model file is empty.	An error occurred when you were creating the model. Create a new model.
No master edge to deploy the model.	Create a master edge before you deploy the model.
Invalid edge IP *IP_address*, only the master edge can be used to deploy the model.	Use the correct IP address for the master edge to deploy a model.
Cannot deploy the model to the edge because the *model_name* model is deployed with the same product type. You cannot deploy a model to an edge when another model with the same product type is running on the edge. You must undeploy the other model first.	You cannot deploy a model to an edge when another model with the same product type is running on the edge.
Database error when creating the data file.	Check the Apache HBase service from the Ambari console.
The *file_name* file did not upload to the storage.	Check Hadoop Distributed File System.
Failed to update data group due to a database error.	Check the Apache HBase service from the Ambari console.

Table 8. Error messages for model instances (continued)	
Message	**Action**
The status of the model instance does not support this action.	Ensure that the model instance is in the correct status and then retry the action.
Training data is invalid. At least one data file must exist in each data group.	Ensure that each data group that belongs to the model instance has valid data files.
Training data is invalid. Check the train and validation ratio to ensure that at least one image exists in the train and test set for each data group.	Make sure that at least one image exists in the train and test sets for each data group.
Failed to retrain the model due to a database error.	Check the Apache HBase service from the Ambari console.
Failed to deploy the model version to edge *edge_name*.	Ensure that the service is running on the master edge. Check the `Service.log` file on the master edge.
Failed to undeploy the model version from edge *edge_name*.	Ensure that the service is running on the master edge. Check the `Service.log` file on the master edge.
Failed to undeploy the model version due to a database error.	Check the Apache HBase service from the Ambari console.
Failed to cancel the training.	Check the Apache HBase service from the Ambari console.
The snapshot value is invalid. Cannot use the snapshot.	The snapshot value does not exist or is not valid.
The snapshot file does not exist on the training server.	Skip the snapshot or retrain the model.
The model instance cannot be deleted because the instance ID is empty. Enter an instance ID.	Enter a valid instance ID.
Failed to delete the model instance due to a database error.	Check the Apache HBase service from the Ambari console.
The model instance with the status of *status* cannot be deleted.	A model instance with the specified status cannot be deleted.

API guide messages

The following messages appear during error conditions for the API guide.

Table 9. Error messages for the API guide	
Message	**Action**
No model found for the specified product type.	If this error occurs as a result of an API call, confirm that you sent the correct product type. If this error occurs in the user interface, try another product type.
No edge information where the model was deployed.	Verify that the corresponding model is deployed.
Failed to transfer image to edge *edge_name*.	Make sure that the service is running on the master edge. Check the `Service.log` file on the master edge.

Table 9. Error messages for the API guide (continued)	
Message	**Action**
Failed to get the image scoring result from edge *edge_name*.	Make sure that the service is running on the master edge. Check the `Service.log` file on the master edge.
A timeout issue occurred. Could not get the image scoring result from edge *edge_name*.	Make sure that the service is running on the master edge. Check the `Service.log` file on the master edge.
The model name cannot be empty when create a new model.	Specify a name for the model.
The model name must be fewer than 128 characters and only contain letters, numbers, spaces, and underscores.	Specify a valid model name.
The product type cannot be empty when create a new model.	Specify the product type.
The product type must be fewer than 128 characters and only contain letters, numbers, and spaces.	Specify a valid product type.
The model name is a duplicate. Use a different model name.	Specify a unique name for the model.
The *file_name* file did not upload to the storage.	Check Hadoop Distributed File System.
Failed to upload model file.	Check your network connection and then retry the upload.

Simulator messages

The following messages appear during error conditions for the simulator.

Table 10. Error messages for the simulator	
Message	**Action**
Failed to send the simulated images.	Make sure that the service is running on the master edge. Check `Service.log` file.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user’s name, user name, password, or other personally identifiable information for purposes of session management, authentication, single sign-on configuration or other usage tracking or functional purposes. These cookies can be disabled, but disabling them will also likely eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies” and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Part Number:

(1P) P/N: