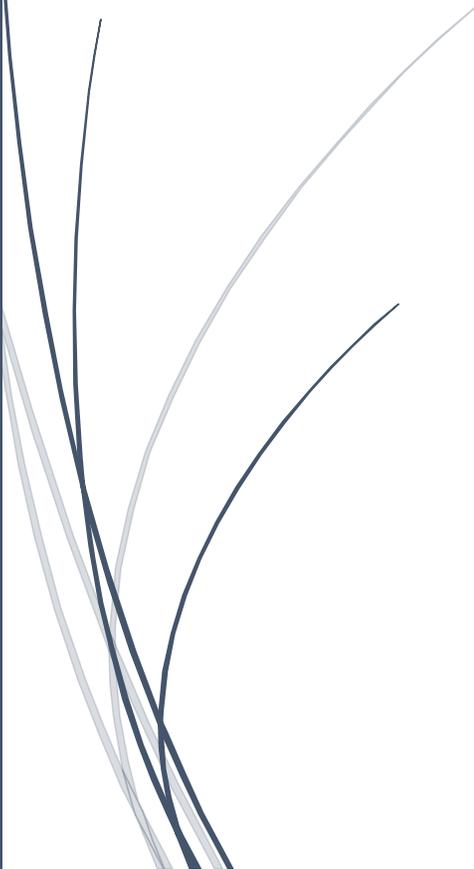




11/10/2020

HUAWEI CLOUD Hands-on lab

Will show how to use MoXing to recognize handwritten digits and images from an MNIST dataset on the Modelarts platform.



HUAWEI CLOUD

Hands-on lab ---Modelarts

Concept Introduction

This section describes how to use MoXing to recognize handwritten digits and images from an MNIST dataset on the ModelArts platform.

MoXing: MoXing is the network model development API provided by the HUAWEI CLOUD deep learning service. Compared with native APIs such as TensorFlow and MXNet, MoXing API simplifies code writing for models. Users only need to care about data input (input_fn) and model build (model_fn) code to implement high-performance running of any model in multiple GPUs and distributed systems.

Overview

The following figure shows the process of identifying handwritten digits and images using MoXing.

1. Region Recommendation: **Singapore**
2. [Preparing Data](#): Obtain the MNIST dataset and upload it to OBS.
3. [Training a Model](#): Use the MoXing framework to compile the model training script and create a training job for model training.
4. [Deploying the Model](#): After obtaining the trained model file, create a prediction job to deploy the model as a real-time prediction service.
5. [Verifying the Model](#): Initiate a prediction request and obtain the prediction result.

Preparing Data

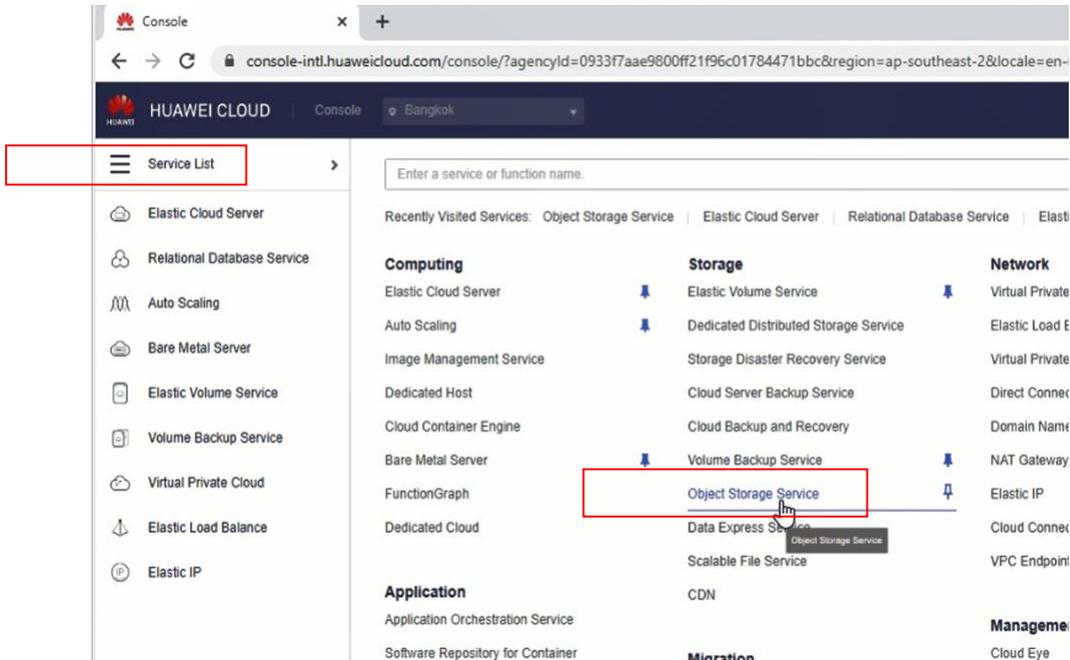
ModelArts provides a sample MNIST dataset named **Mnist-Data-Set**. This example uses this dataset to build a model. Perform the following operations to upload the dataset to the OBS directory **test-modelarts/dataset-mnist** created in preparation.

1. Decompress the **Mnist-Data-Set.zip** file, for example, to the **Mnist-Data-Set** directory on the local PC.

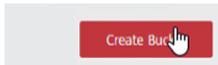
Mnist-Data-Set download link: <https://test-modelarts-hol001.obs.ap-southeast-1.myhuaweicloud.com/Modelars-demo-sample/Mnist-Data-Set.zip>

2. Create the OBS bucket “test-modelarts/dataset-mnist”

In the Console, on the top left of the screen, select Navigation menu > Storage > Object Storage Service



Click on Create Bucket



Create the bucket test-modelars

Create Bucket

Region: AP-Hong-Kong
Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region. Once a bucket is created, the region cannot be changed.

Bucket Name: test-modelarts
Naming conventions:
 - The name must be globally unique in OBS.
 - The name of a bucket or parallel file system can be reused 30 minutes after the bucket or parallel file system is deleted.
 - The name must contain 3 to 63 characters. Only lowercase letters, digits, hyphens (-), and periods (.) are allowed.
 - The name cannot start or end with a period (.) or hyphen (-), and cannot contain two consecutive periods (.) or contain a period (.) and a hyphen (-) adjacent to each other.
 - The name cannot be an IP address.
 - If the name contains any periods, a security certificate verification message may appear when you access the bucket or its objects by entering a domain name.

Storage Class: Standard (Selected), Infrequent Access, Archive
Optimized for frequently accessed (multiple times per month) data such as small and essential files that require low latency. The storage class of a bucket is inherited by objects uploaded to the bucket by default. You can also change the storage class of an object when uploading it to the bucket. [Learn more](#)

Default Encryption: Enable (Selected), Disable
Recommended Key management is offered for free, better securing your core data.

Direct Reading: Direct reading of Archive data is supported in the following region: CN North-Beijing1, CN East-Shanghai1, CN East-Shanghai2, CN North-Beijing4, CN South-Guangzhou.

Enterprise Project: default [Create Enterprise Project](#)

Tags: It is recommended that you use TMS's predefined tag function to add the same tag to different cloud resources.
Tag key: [] Tag value: []
You can add 10 more tags.

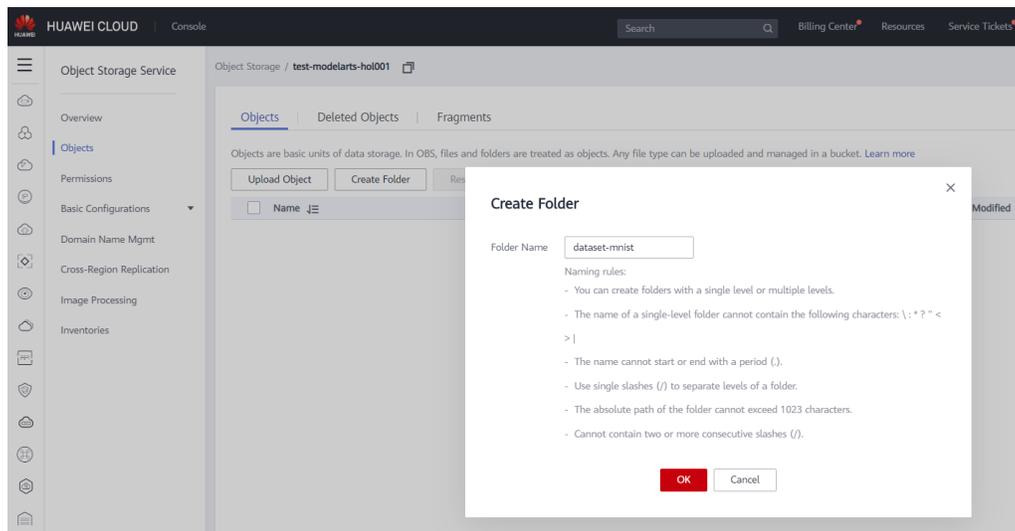
Buckets are billed on a per-use basis.

Bucket creation is free of charge. You will be billed for the service only after using billable items. [Pricing details](#)

[Create Now](#)

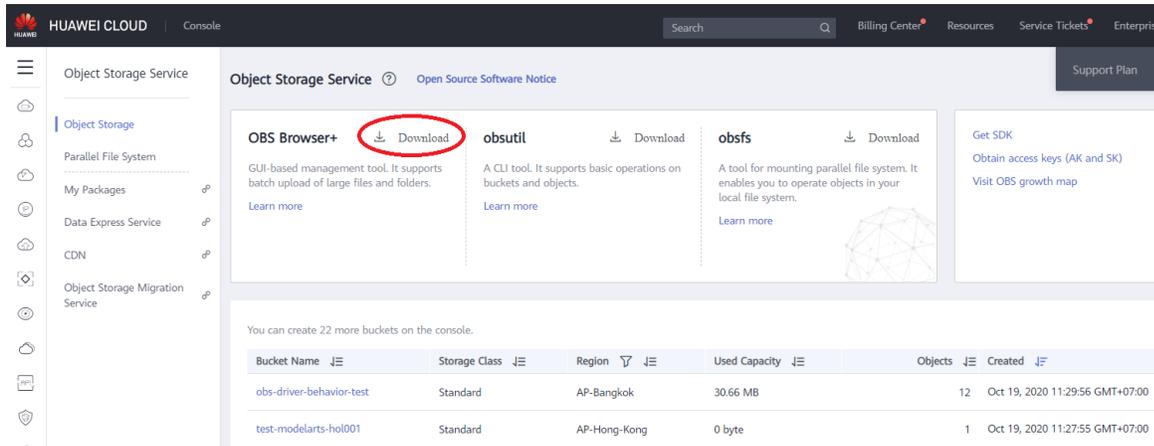
If the console informs that ‘The bucket name already exists or used by other users. Try another one.’ Suggest to using other bucket name such as **test-modelarts-hol001**

Create the folder **dataset-mnist** in the bucket test-modelarts-hol001



3. Upload all files in the **Mnist-Data-Set** folder to the **test-modelarts/dataset-mnist** directory on OBS in batches. For details about how to upload files, see as follows:

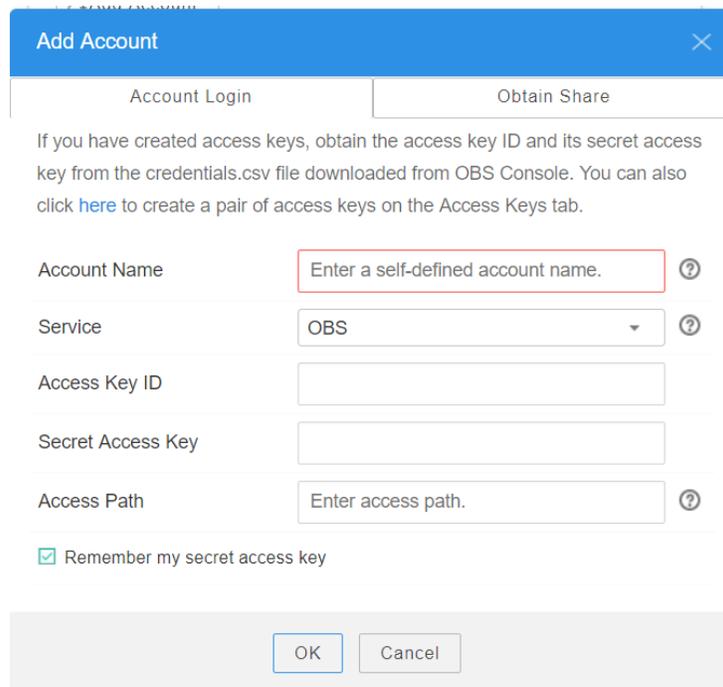
Download the OBS browser



The screenshot shows the Huawei Cloud console interface for the Object Storage Service. The 'Download' button for 'OBS Browser+' is highlighted with a red circle. Below the download buttons, there is a table of buckets.

Bucket Name	Storage Class	Region	Used Capacity	Objects	Created
obs-driver-behavior-test	Standard	AP-Bangkok	30.66 MB	12	Oct 19, 2020 11:29:56 GMT+07:00
test-modelarts-hol001	Standard	AP-Hong-Kong	0 byte	1	Oct 19, 2020 11:27:55 GMT+07:00

Configure the OBS browser, Account Name, Access Key ID (AK), Secret Access Key (SK) is mandatory.



The 'Add Account' dialog box is shown with the 'Account Login' tab selected. The 'Account Name' field is highlighted with a red border.

Account Name: Enter a self-defined account name. ?

Service: OBS ?

Access Key ID: [Empty field]

Secret Access Key: [Empty field]

Access Path: Enter access path. ?

Remember my secret access key

OK Cancel

Upload the file directory to cloud OBS bucket.

Account Name [?](#)

APClouddemoTH

Service [?](#)

HUAWEI CLOUD OBS (default) ▼

Access Key ID

S6HCF4PM24UMJOO6WN6V

Secret Access Key

.....

Access Path [?](#)

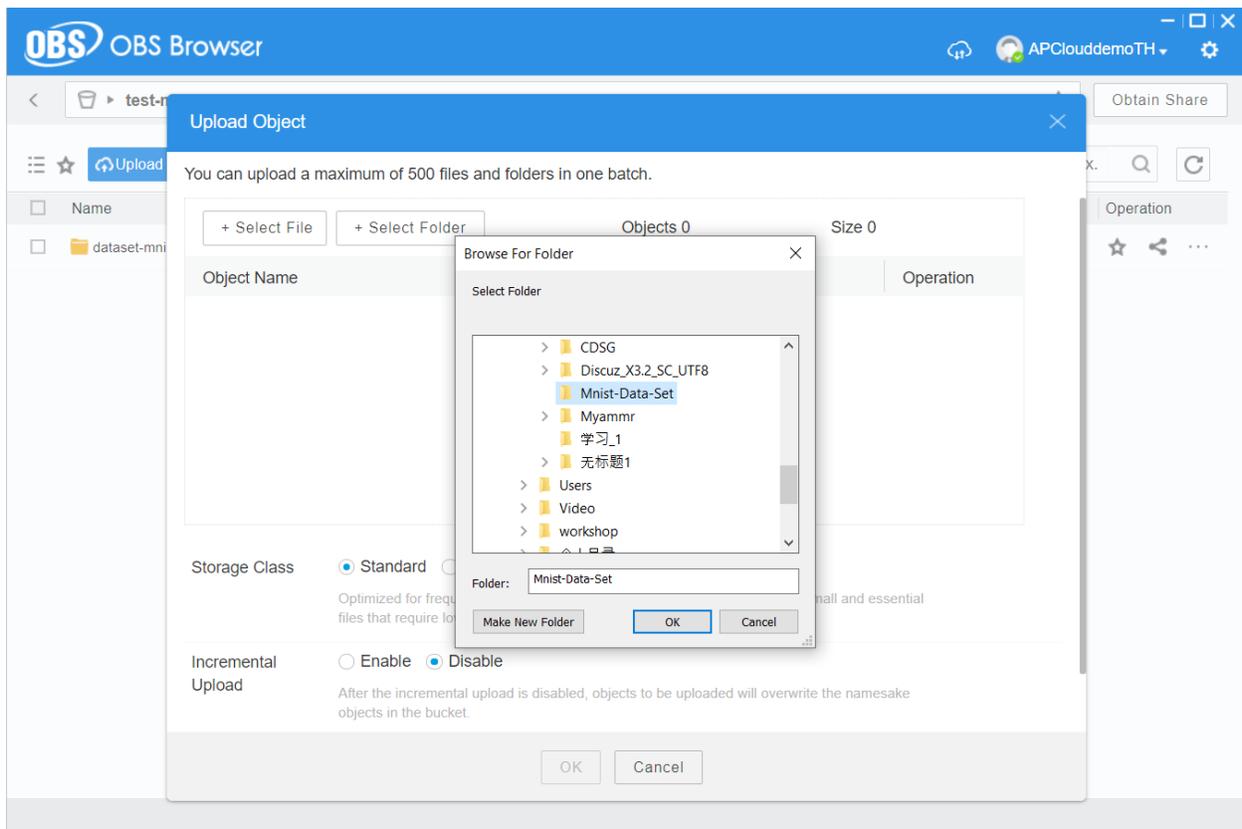
Enter an access path (eg: obs://bucket/folder)

Remember my access keys. [?](#)

Log In

[Obtain Access Keys](#)

[Login Help](#) | [More](#) ▼



The following provides content of the **Mnist-Data-Set** dataset. **.gz** is the compressed package.

- **t10k-images-idx3-ubyte**: validation set, which contains 10,000 samples
- **t10k-images-idx3-ubyte.gz**: compressed package file of the validation set.
- **t10k-labels-idx1-ubyte**: labels of the validation set, which contains the labels of the 10,000 samples
- **t10k-labels-idx1-ubyte.gz**: compressed package file of the validation set label.
- **train-images-idx3-ubyte**: training set, which contains 60,000 samples
- **train-images-idx3-ubyte.gz**: compressed package file of the training set.
- **train-labels-idx1-ubyte**: labels of the training set, which contains the labels of the 60,000 samples
- **train-labels-idx1-ubyte.gz**: compressed package file of the training set label.

Training a Model

After the data preparation is completed, use the MoXing API to compile the training script code. ModelArts provides a code sample, **train_mnist.py**. The following uses this sample to train the model.

1. Upload the **train_mnist.py** file to OBS, for example, **test-modelarts-hol001/mnist-MoXing-code**.

Down load link: https://test-modelarts-hol001.obs.ap-southeast-1.myhuaweicloud.com/Modelarts-demo-sample/train_mnist.py

2. On the ModelArts management console, choose **Training Management > Training Jobs**, and click **Create** in the upper left corner.
3. On the **Modelarts Console** page, click **Training Management->Training Jobs** and Click **Create**.

Data Source: Select **Data path**, and then select the OBS path for saving the dataset.

Basic information for creating a training job

The screenshot displays the Huawei Cloud ModelArts console interface for creating a training job. The top navigation bar includes 'HUAWEI CLOUD', 'Console', 'Hong-Kong', and a search bar. The main configuration area is divided into several sections:

- Billing Mode:** Set to 'Pay-per-use'.
- Name:** 'trainjob-mnist2' (with a green checkmark).
- Version:** 'V0001 (System-defined version number)'. A 'Description' field is present with a character count of '0/256'.
- One-Click Configuration:** Includes an 'Import parameters' link.
- Algorithm Source:** Tabs for 'Algorithm Management', 'Built-in', 'Frequently-used' (selected), and 'Custom'. Below the tabs, it states 'Frequently-used AI engines used to create training jobs.' and lists:
 - AI Engine:** TensorFlow and TF-1.8.0-python2.7.
 - Code Directory:** '/test-modelarts-hol001/mnist-MoXing-code/' (with a 'Select' button).
 - Boot File:** '/test-modelarts-hol001/mnist-MoXing-code/train_mnist.' (with a 'Select' button).
- Data Source:** Tabs for 'Dataset' and 'Data path' (selected). Below, the **Data Path** is '/test-modelarts-hol001/dataset-mnist/' (with a 'Select' button and a trash icon).

Parameters for creating a training job

* Training Output Path
We recommend you select an empty directory as the output path.

Running Parameter
 =
 =

Job Log Path
By default, logs are stored in the service and will be deleted irregularly. Select a path for storing logs.

* Resource Pool Public resource pools Dedicated resource pools

* Type CPU GPU

* Specifications

* Compute Nodes

Notification

Save Training Parameters

Price **\$6.30 USD**/hour

4. On the **Confirm** tab page, check the parameters of the training job and click **Submit**.
5. On the **Training Jobs** page, when the training job status changes to **Running Success**, the model training is completed. If any exception occurs, click the job name to go to the job details page and view the training job logs.

NOTE:

The training job may take more than 10 minutes to complete. If the training time exceeds a certain period (for example, one hour), manually stop it to release resources. Otherwise, the account balance may be insufficient, especially for the models that are trained using GPUs.

6. (Optional) During or after model training, you can create a visualization job to view parameter statistics.

In **Training Output Path**, select the value of **Training Output Path** specified for the training job. Complete visualization job creation as prompted.

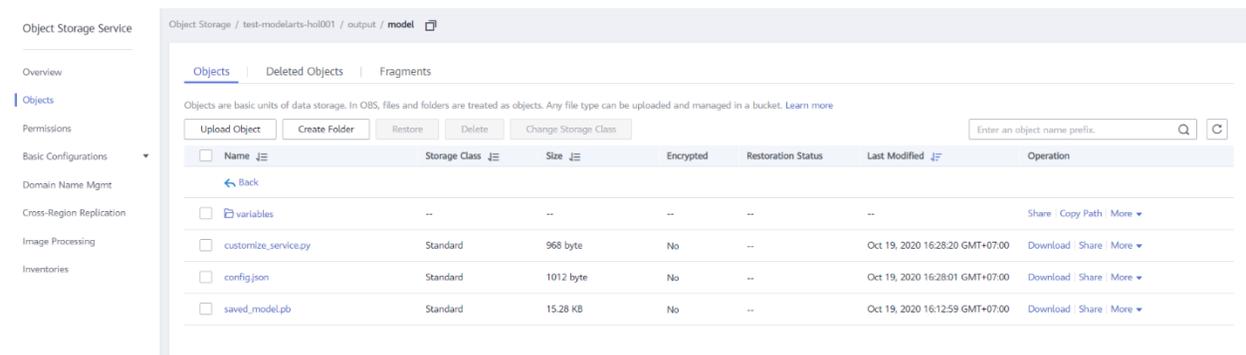
Deploying the Model

After the model training is completed, deploy the model as a real-time prediction service. ModelArts provides the compiled inference code **customize_service.py** and configuration file **config.json**.

Customize_service.py download link: https://test-modelarts-hol001.obs.ap-southeast-1.myhuaweicloud.com/Modelars-demo-sample/customize_service.py

Config.json download link: <https://test-modelarts-hol001.obs.ap-southeast-1.myhuaweicloud.com/Modelars-demo-sample/config.json>

1. Upload the **customize_service.py** and **config.json** files to OBS. The files must be stored in the path for saving the model generated for the training job, for example, **test-modelarts-hol001/output/model**.



NOTE:

- The training job creates a **model** folder in the path specified by **Training Output Path** to store the generated model.
 - The inference code and configuration file must be uploaded to the **model** folder.
2. On the ModelArts management console, choose **Model Management > Models** in the left navigation pane. The **Models** page is displayed. Click **Import** in the upper left corner.
 3. On the **Import Model** page, set required parameters as shown and click **Next**. In **Meta Model Source**, select **OBS**. Set **Meta Model** to the path specified by **Training Output Path** in the training job but not the **model** folder under the path. Otherwise, the system cannot find the model and related files automatically.

Storage Path



Select a folder.

Name	Last Modified	Type	Size
dataset-mnist	--	Folder	--
mnist-MoXing-code	--	Folder	--
mnist-model	--	Folder	--
<input checked="" type="radio"/> output	--	Folder	--

The OBS will be billed based on usage. See [OBS pricing](#)

Import Model

* Name

* Version

Label

Description

0/100

* Meta Model Source

Import one of the following models from OBS: TensorFlow, MXNet, Caffe, Spark_MLlib, Scikit_Learn, XGBoost, PyTorch, Image. To import a model image, you are advised to select Container image. Ensure that the model file is stored in the model directory and specify the parent directory of the model directory as the path. If the model requires inference code, ensure that the code is stored in the model directory. The file name must be "customize_service.py". Ensure that the model meets the [model package specifications](#).

* Meta Model

AI Engine

* Deployment Type Real-time services Batch services

Configuration File

Inference Code

Parameter Configuration

Installation Method	Name	Version	Constraint
pip	numpy	1.15.0	Later version
pip	h5py		--
pip	tensorflow	1.8.0	Later version
pip	Pillow	5.2.0	Later version

Min. Inference Specs

Model Description

Price **Free**

- On the **Models** page, if the model status changes to **Normal**, the model has been imported successfully. Click the triangle next to a model name to expand all versions of the model. In the row of a version, choose **Deploy > Real-Time Services** in the **Operation** column to deploy the model as a real-time service.
- On the **Deploy** page, set parameters by referring to [Figure 4](#) and click **Next**.

Figure 4 Deploy

The screenshot shows the 'Configure' step of the SageMaker console. At the top, there are three steps: 1. Configure, 2. Confirm, and 3. (unlabeled). The 'Configure' step is active. The interface is divided into two main sections. The top section contains: 'Billing Mode' set to 'Pay-per-use'; 'Name' set to 'service-e3b9'; 'Auto Stop' toggle is turned on, with a note: 'If this function is enabled, the real-time service will automatically stop at the specified time, and the service charging will also stop.' Below this, there are radio buttons for '1 hour later', '2 hours later', '4 hours later', '6 hours later', and 'Custom'. A 'Description' text area is empty, with a character count of '0/100'. The bottom section is titled 'Model and Configuration'. It has two tabs: 'My Models' (selected) and 'AI Market Subscriptions'. Under 'My Models', 'Model' is set to 'model-8d32' and 'Specifications' is 'CPU: 2 vCPUs | 8 GiB'. To the right, 'Traffic Ratio (%)' is set to 100 and 'Compute Nodes' is set to 1. Below these, it says 'Application scenario: Standard CPU specifications, meeting the running and prediction requirements of most models'. At the bottom of this section, there is an 'Environment Variable' field and a '+ Add Environment Variable' button. At the very bottom of the console, the price is shown as '\$0.15 USD/hour' and a red 'Next' button is on the right.

6. On the **Confirm** tab page, check the configurations and click **Submit** to create a real-time service.
7. After the real-time service is created, the **Service Deployment > Real-Time Services** page is displayed. The service deployment takes some time. When the service status changes to **Running**, the service is successfully deployed.

Verifying the Model

After the real-time service is deployed, access the service to send a prediction request for test.

1. On the **Real-Time Services** page, click the name of the real-time service. The real-time service details page is displayed.
2. On the real-time service details page, click the **Prediction** tab.
3. Click **Upload** next to **Image File** to upload an image with a white handwritten digit on a black background and click **Predict**.

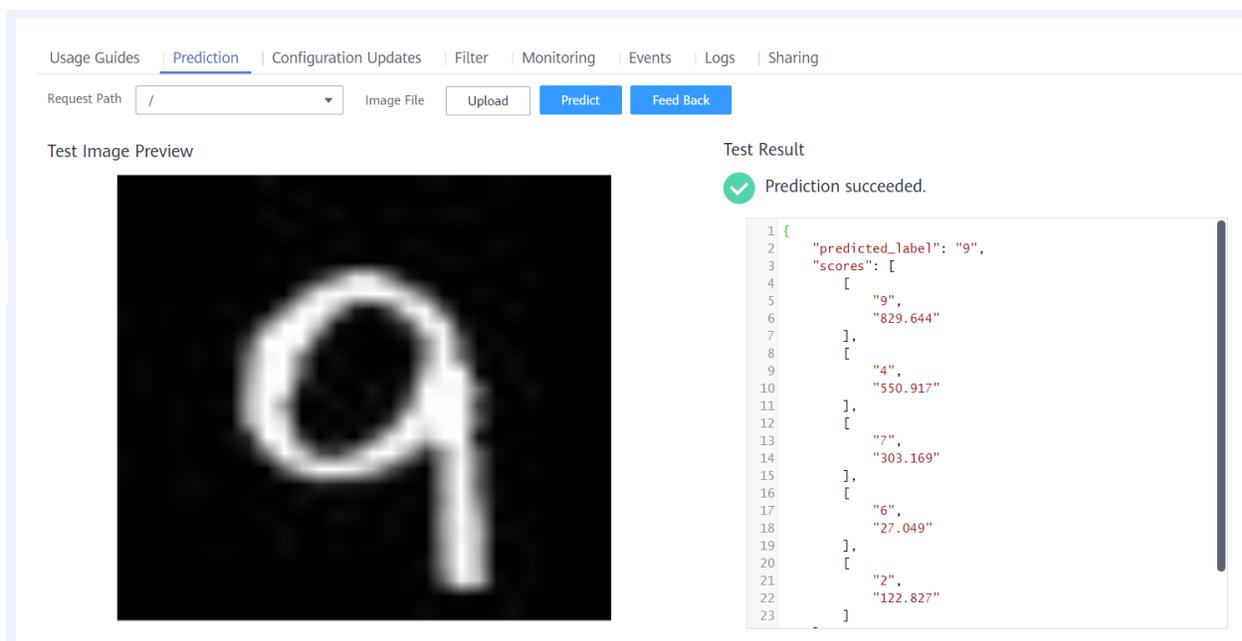
Testing data download link: <https://test-modelarts-hol001.obs.ap-southeast-1.myhuaweicloud.com/Modelars-demo-sample/test%20data.zip>

After the prediction is completed, the prediction result is displayed in the **Test Result** pane. According to the prediction result, the digit on the image is **4**.

NOTE:

- As specified in the inference code and configuration files, the size of the image used for prediction must be 28 x 28 pixels, and the image must be in the JPG format and must contain white handwritten digits on a black background.
- You are advised not to use the images provided by the dataset. You can use the drawing tool provided by the Windows operating system to draw an image for prediction.
- If a single-channel image that is not in the required format is used, the prediction result may be inaccurate.

Figure 5 Prediction result of the real-time service



The screenshot displays a web interface for a real-time prediction service. At the top, there are navigation tabs: Usage Guides, Prediction (selected), Configuration Updates, Filter, Monitoring, Events, Logs, and Sharing. Below the tabs, there is a 'Request Path' dropdown menu set to '/', an 'Image File' input field, and three buttons: 'Upload', 'Predict', and 'Feed Back'. The 'Predict' button is highlighted in blue.

On the left side, under the heading 'Test Image Preview', there is a square image showing a white handwritten digit '9' on a black background. On the right side, under the heading 'Test Result', there is a green checkmark icon followed by the text 'Prediction succeeded.' Below this, a code block displays the JSON response:

```
1 {
2   "predicted_label": "9",
3   "scores": [
4     [
5       "9",
6       "829.644"
7     ],
8     [
9       "4",
10      "550.917"
11    ],
12    [
13      "7",
14      "303.169"
15    ],
16    [
17      "6",
18      "27.049"
19    ],
20    [
21      "2",
22      "122.827"
23    ]
24  ]
25 }
```