Running Jupyter Notebooks on bacalhau

Open in Colab launch binder Stars 553

Introduction

Jupyter Notebooks have become an essential tool for data scientists, researchers, and developers for interactive computing and the development of data-driven projects. They provide an efficient way to share code, equations, visualizations, and narrative text with support for multiple programming languages. In this tutorial, we will introduce you to running Jupyter Notebooks on Bacalhau, a powerful and flexible container orchestration platform. By leveraging Bacalhau, you can execute Jupyter Notebooks in a scalable and efficient manner using Docker containers, without the need for manual setup or configuration.

In the following sections, we will explore two examples of executing Jupyter Notebooks on Bacalhau:

1. Executing a Simple Hello World Notebook: We will begin with a basic example to familiarize you with the process of running a Jupyter Notebook on Bacalhau. We will execute a simple "Hello, World!" notebook to demonstrate the steps required for running a notebook in a containerized environment.

2. Notebook to Train an MNIST Model: In this section, we will dive into a more advanced example. We will execute a Jupyter Notebook that trains a machine-learning model on the popular MNIST dataset. This will showcase the potential of Bacalhau to handle more complex tasks while providing you with insights into utilizing containerized environments for your data science projects.

1. Executing a Simple Hello World Notebook

There are no external dependencies that we need to install all dependencies are already there in the container

Prerequisite

To get started, you need to install the Bacalhau client, see more information here

```
%%bash --out job_id
bacalhau docker run \
--wait \
--id-only \
--timeout 3600 \
--wait-timeout-secs 3600 \
-w /inputs \
-i https://raw.githubusercontent.com/js-ts/hello-notebook/main/hello.ipynb \
jsacex/jupyter \
-- jupyter nbconvert --execute --to notebook --output
/outputs/hello_output.ipynb hello.ipynb
```

- `-i`: This flag stands for "input" and is used to provide the URL of the input Jupyter Notebook you want to execute. In this case, we are using a public gist containing the 'hello.ipynb' notebook.
- `https://gist.githubusercontent.com/js-ts/54d1015af4cb2ec5882ada2180ec042c/raw/a4c07357db014572da2ff27628a3669bfb99ba4d/hello.ipynb`: This is the URL of the input Jupyter Notebook.
- `jupyter/base-notebook`: This is the name of the Docker image used for running the Jupyter Notebook. It is a minimal Jupyter Notebook stack based on the official Jupyter Docker Stacks.
- `--`: This double dash is used to separate the Bacalhau command options from the command that will be executed inside the Docker container.
- `jupyter nbconvert`: This is the primary command used to convert and execute Jupyter Notebooks. It allows for the conversion of notebooks to various formats, including execution.
- `--execute`: This flag tells `nbconvert` to execute the notebook and store the results in the output file.
- `--to notebook`: This option specifies the output format. In this case, we want to keep the output as a Jupyter Notebook.
- `--output /outputs/hello_output.ipynb`: This option specifies the path and filename for the output Jupyter Notebook, which will contain the results of the executed input notebook.
- `/inputs/hello.ipynb`: This is the path of the input Jupyter Notebook inside the Docker container.

Checking the State of your Jobs

- **Job status**: You can check the status of the job using `bacalhau list`.

```
%%bash
bacalhau list --id-filter=${JOB_ID} --no-style
```

When it says `Published` or `Completed`, that means the job is done, and we can get the results.

- **Job information**: You can find out more information about your job by using `bacalhau describe`.

```
%%bash
bacalhau describe ${JOB_ID}
```

- **Job download**: You can download your job results directly by using `bacalhau get`. Alternatively, you can choose to create a directory to store your results. In the command below, we created a directory and downloaded our job output to be stored in that directory.

```
%%bash
rm -rf results && mkdir results # Temporary directory to store the results
bacalhau get ${JOB_ID} --output-dir results # Download the results
```

After the download has finished you should see the following contents in the results directory.

```
%%bash
ls results/outputs
```

2. Running Notebook to Train an MNIST Model

Building the container (optional)

Prerequisite

- Install Docker on your local machine.
- Sign up for a DockerHub account if you don't already have one. Steps

Step 1: Create a Dockerfile Create a new file named Dockerfile in your project directory with the following content:

```
# Use the official Python image as the base image
FROM tensorflow/tensorflow:nightly-gpu

# Set the working directory in the container
WORKDIR /

RUN apt-get update -y

COPY mnist.ipynb .
# Install the Python packages
COPY requirements.txt .

RUN python3 -m pip install --upgrade pip

# Install the Python packages
RUN pip install --no-cache-dir -r requirements.txt

RUN pip install -U scikit-learn
```

This Dockerfile creates a Docker image based on the official TensorFlow GPU-enabled image, sets the working directory to the root, updates the package list, and copies an IPython notebook (mnist.ipynb) and a requirements.txt file. It then upgrades pip and installs Python packages from the requirements.txt file, along with scikit-learn. The resulting image provides an environment ready for running the mnist.ipynb notebook with TensorFlow and scikit-learn, as well as other specified dependencies.

Step 2: Build the Docker Image In your terminal, navigate to the directory containing the Dockerfile and run the following command to build the Docker image:

```
docker build -t your-dockerhub-username/jupyter-mnist-tensorflow:latest .
```

Replace "your-dockerhub-username" with your actual DockerHub username. This command will build the Docker image and tag it with your DockerHub username and the name "your-dockerhub-username/jupyter-mnist-tensorflow".

Step 3: Push the Docker Image to DockerHub Once the build process is complete, Next, push the Docker image to DockerHub using the following command:

```
docker push your-dockerhub-username/jupyter-mnist-tensorflow
```

Again, replace "your-dockerhub-username" with your actual DockerHub username. This command will push the Docker image to your DockerHub repository.

Running the job on Bacalhau

Prerequisite

To get started, you need to install the Bacalhau client, see more information here

```
%%bash --out job_id
bacalhau docker run \
--wait \
--id-only \
--timeout 3600 \
--wait-timeout-secs 3600 \
 --gpu 1 \
-i gitlfs://huggingface.co/datasets/VedantPadwal/mnist.git \
jsacex/jupyter-tensorflow-mnist:v02 \
-- jupyter nbconvert --execute --to notebook --output
/outputs/mnist_output.ipynb mnist.ipynb
```

Structure of the command

- `--gpu 1`: Flag to specify the number of GPUs to use for the execution. In this case, 1 GPU will be used.
- `-i gitlfs://huggingface.co/datasets/VedantPadwal/mnist.git`: The `-i` flag is used to clone the MNIST dataset from Hugging Face's repository using Git LFS. The files will be mounted inside the container.
- `jsacex/jupyter-tensorflow-mnist:v02`: The name and the tag of the Docker image.

- `--`: Double hyphen is used to separate the Docker command-line options from the command that will be executed inside the container.

- `jupyter nbconvert --execute --to notebook --output /outputs/mnist_output.ipynb mnist.ipynb`: The command to be executed inside the container. In this case, it runs the `jupyter nbconvert` command to execute the `mnist.ipynb` notebook and save the output as `mnist_output.ipynb` in the `/outputs` directory.

Checking the State of your Jobs

- **Job status**: You can check the status of the job using `bacalhau list`.

```
%%bash
bacalhau list --id-filter=${JOB_ID} --no-style
```

When it says `Published` or `Completed`, that means the job is done, and we can get the results.

- **Job information**: You can find out more information about your job by using `bacalhau describe`.

```
%%bash
bacalhau describe ${JOB_ID}
```

- **Job download**: You can download your job results directly by using `bacalhau get`. Alternatively, you can choose to create a directory to store your results. In the command below, we created a directory and downloaded our job output to be stored in that directory.

```
%%bash
rm -rf results && mkdir results # Temporary directory to store the results
bacalhau get ${JOB_ID} --output-dir results # Download the results
```

After the download has finished you should see the following contents in the results directory.

```
%%bash
ls results/outputs
```

The outputs include our trained model and the jupyter notebook with the output cells

Edit this page

*Last updated on **Jan 11, 2024***