Wallaroo-101

April 5, 2022

Welcome to the Wallaroo, the fastest, easiest, and most efficient production ready machine learning system.

This tutorial is created to help you get started with Wallaroo right away. We'll start with a brief explanation of how Wallaroo works, then provide the credit card fraud detection model so you can see it working.

This guide assumes that you've installed Wallaroo in your cloud Kubernetes cluster. This can be either:

- Amazon Web Services (AWS)
- Microsoft Azure
- Google Cloud Platform

For instructions on setting up your cloud Kubernetes environment, check out the Wallaroo Environment Setup Guides for your particular cloud provider.

0.1 SDK Introduction

The Wallaroo SDK lets you quickly get your models working with your data and getting results. The typical flow follows these steps:

- **Connect**: Connect to your Wallaroo Instance.
- **Create or Connect to a Workspace**: Create a new workspace that will contain your models and pipelines, or connect to an existing one.
- **Upload or Use Existing Models**: Upload your models to your workspace, or use ones that have already been uploaded.
- **Create or Use Existing Pipelines**: Create or use an existing pipeline. This is where you'll set the **steps** that will ingest your data, submit it through each successive model, then return a result.
- **Deploy Your Pipeline**: Deploying a pipeline allocates resources from your Kubernetes environment for your models.
- **Run an Inference**: This is where it all comes together. Submit data through your pipeline either as a file or to your pipeline's deployment url, and get results.
- **Undeploy Your Pipeline**: This returns the Kubernetes resources your pipeline used back to the Kubernetes environment.

For a more detailed rundown of the Wallaroo SDK, see the Wallaroo SDK Essentials Guide.

0.1.1 Introduction to Workspaces

A Wallaroo **Workspace** allows you to manage a set of models and pipelines. You can assign users to a workspace as either an **owner** or **collaborator**.

When working within the Wallaroo SDK, the first thing you'll do after connecting is either create a workspace or set an existing workspace your **current workspace**. From that point on, all models uploaded and pipelines created or used will be in the context of the current workspace.

0.1.2 Introduction to Models

A Wallaroo **model** is a trained Machine Learning model that is uploaded to your current workspace. These are the engines that take in data, run it through whatever process they have been trained for, and return a result.

Models don't work in a vacuum - they are allocated to a pipeline as detailed in the next step.

0.1.3 Introduction to Pipelines

A Wallaroo **pipeline** is where the real work occurs. A pipeline contains a series of **steps** - sequential sets of models which take in the data from the preceeding step, process it through the model, then return a result. Some models can be simple, such as the `cc_fraud` example listed below where the pipeline has only one step:

- Step 0: Take in data
- Step 1: Submit data to the model `ccfraud-model`.
- Step Final: Return a result

Some models can be more complex with a whole series of models - and those results can be submitted to still other pipeline. You can make pipelines as simple or complex as long as it meets your needs.

Once a step is created you can add additional steps, remove a step, or swap one out until everything is running perfectly.

Note: The Community Edition of Wallaroo limits users to two active pipelines, with a maximum of five steps per pipeline.

With all of that introduction out of the way, let's proceed to our Credit Card Detection Model.

This example will demonstrate how to use Wallaroo to detect credit card fraud through a trained model and sample data. By the end of this example, you'll be able to:

- Start the Wallaroo client.
- Create a workspace.
- Upload the credit card fraud detection model to the workspace.
- Create a new pipeline and set it to our credit card fraud detection model.
- Run a smoke test to verify the pipeline and model is working properly.
- Perform a bulk inference and display the results.
- Undeploy the pipeline to get back the resources from our Kubernetes cluster.

This example and sample data comes from the Machine Learning Group's demonstration on Credit Card Fraud detection.

0.2 Open a Connection to Wallaroo

The first step is to connect to Wallaroo through the Wallaroo client. The Python library is included in the Wallaroo install and available through the Jupyter Hub interface provided with your Wallaroo environment.

This is accomplished using the `wallaroo.Client()` command, which provides a URL to grant the SDK permission to your specific Wallaroo environment. When displayed, enter the URL into a browser and confirm permissions. Store the connection into a variable that can be referenced later.

[1]:
```
import wallaroo
```

[2]:
```
wl = wallaroo.Client()
```

0.3 Create a New Workspace

Next we're going to create a new workspace called `ccfraud_workspace` for our model, then set it as our current workspace context.

[3]:
```
new_workspace = wl.create_workspace("ccfraud-workspace")
wl.set_current_workspace(new_workspace)
```

[3]:
```
{'name': 'ccfraud-workspace', 'id': 17, 'archived': False, 'created_by':
'fad23e2b-d326-48da-ba23-18ee30d26030', 'created_at':
'2022-04-05T19:27:31.004672+00:00', 'models': [], 'pipelines': []}
```

Just to make sure, let's list our current workspace. If everything is going right, it will show us we're in the `ccfraud-workspace`.

[4]:
```
wl.get_current_workspace()
```

[4]:
```
{'name': 'ccfraud-workspace', 'id': 17, 'archived': False, 'created_by':
'fad23e2b-d326-48da-ba23-18ee30d26030', 'created_at':
'2022-04-05T19:27:31.004672+00:00', 'models': [], 'pipelines': []}
```

0.4 Upload a model

Our workspace is created. Let's upload our credit card fraud model to it. This is the file name `ccfraud.onnx`, and we'll upload it as `ccfraud-model`. The credit card fraud model is trained to detect credit card fraud based on a 0 to 1 model: The closer to 0 the less likely the transactions indicate fraud, while the closer to 1 the more likely the transactions indicate fraud.

Since we're already in our default workspace `ccfraud-workspace`, it'll be uploaded right to there. Once that's done uploading, we'll list out all of the models currently deployed so we can see it included.

[5]:
```
ccfraud_model = wl.upload_model("ccfraud-model", "./ccfraud.onnx").configure()
```

We can verify that our model was uploaded by listing the models uploaded to our Wallaroo instance with the `list_models()` command. NOte that since we uploaded this model before, we now have different versions of it we can use for our testing.

```
[6]: wl.list_models()
```

```
[6]: [{'name': 'ccfraud-model', 'version': 'd145cd72-8c0c-433c-9966-5572a589f743',
     'file_name': 'ccfraud.onnx', 'last_update_time': datetime.datetime(2022, 4, 5,
     19, 27, 31, 134114, tzinfo=tzutc())},
      {'name': 'ccfraud-model', 'version': '3378c4cd-4238-4be4-ab62-82227ac65748',
     'file_name': 'ccfraud.onnx', 'last_update_time': datetime.datetime(2022, 4, 5,
     19, 22, 52, 614059, tzinfo=tzutc())}]
```

0.5 Create a Pipeline

With our model uploaded, time to create our pipeline and deploy it so it can accept data and process it through our `ccfraud-model`. We'll call our pipeline `ccfraud-pipeline`.

```
[7]: ccfraud_pipeline = wl.build_pipeline('ccfraud-pipeline')
```

Now our pipeline is created. Let's add a single **step** to it - in this case, our `ccfraud-model` that we uploaded to our workspace.

```
[8]: ccfraud_pipeline.add_model_step(ccfraud_model)
```

```
[8]: {'name': 'ccfraud-pipeline', 'create_time': datetime.datetime(2022, 4, 5, 19,
     27, 31, 934255, tzinfo=tzutc()), 'definition': "[{'ModelInference': {'models':
     [{'name': 'ccfraud-model', 'version': 'd145cd72-8c0c-433c-9966-5572a589f743',
     'sha': 'bc85ce596945f876256f41515c7501c399fd97ebcb9ab3dd41bf03f8937b4507'}]}}]"}
```

And now we can deploy our pipeline and assign resources to it. This typically takes about 45 seconds once the command is issued.

```
[9]: ccfraud_pipeline.deploy()
```

```
Waiting for deployment - this will take up to 45s … ok
```

```
[9]: {'name': 'ccfraud-pipeline', 'create_time': datetime.datetime(2022, 4, 5, 19,
     27, 31, 934255, tzinfo=tzutc()), 'definition': "[{'ModelInference': {'models':
     [{'name': 'ccfraud-model', 'version': 'd145cd72-8c0c-433c-9966-5572a589f743',
     'sha': 'bc85ce596945f876256f41515c7501c399fd97ebcb9ab3dd41bf03f8937b4507'}]}}]"}
```

We can see our new pipeline with the `status()` command.

```
[10]: ccfraud_pipeline.status()
```

```
[10]: {'status': 'Running',
       'details': None,
       'engines': [{'ip': '10.52.0.250',
         'name': 'engine-6685bc89dc-gx8rp',
         'status': 'Running',
         'reason': None,
         'pipeline_statuses': {'pipelines': [{'id': 'ccfraud-pipeline',
            'status': 'Running'}]},
         'model_statuses': {'models': [{'name': 'ccfraud-model',
            'version': 'd145cd72-8c0c-433c-9966-5572a589f743',
            'sha': 'bc85ce596945f876256f41515c7501c399fd97ebcb9ab3dd41bf03f8937b4507',
            'status': 'Running'}]}}],
       'engine_lbs': [{'ip': '10.52.0.251',
         'name': 'engine-lb-85846c64f8-ptzq4',
         'status': 'Running',
         'reason': None}]}
```

With our pipeline deployed, let's run a smoke test to make sure it's working right. We'll run an inference through our pipeline from the file `smoke_test.json` and see the results.

[11]:
```
ccfraud_pipeline.infer_from_file('./smoke_test.json')
```

```
Waiting for inference response - this will take up to 45s … ok
```

[11]:
```
[InferenceResult({'check_failures': [],
  'elapsed': 175233,
  'model_name': 'ccfraud-model',
  'model_version': 'd145cd72-8c0c-433c-9966-5572a589f743',
  'original_data': {'tensor': [[1.0678324729342086,
                                0.21778102664937624,
                                -1.7115145261843976,
                                0.6822857209662413,
                                1.0138553066742804,
                                -0.43350000129006655,
                                0.7395859436561657,
                                -0.28828395953577357,
                                -0.44726268795990787,
                                0.5146124987725894,
                                0.3791316964287545,
                                0.5190619748123175,
                                -0.4904593221655364,
                                1.1656456468728569,
                                -0.9776307444180006,
                                -0.6322198962519854,
                                -0.6891477694494687,
                                0.17833178574255615,
                                0.1397992467197424,
                                -0.35542206494183326,
                                0.4394217876939808,
                                1.4588397511627804,
                                -0.3886829614721505,
                                0.4353492889350186,
                                 1.7420053483337177,
                                 -0.4434654615252943,
                                 -0.15157478906219238,
                                 -0.26684517248765616,
                                 -1.454961775612449]]},
  'outputs': [{'Float': {'data': [0.001497417688369751],
                        'dim': [1, 1],
                        'v': 1}}],
  'pipeline_name': 'ccfraud-pipeline',
  'time': 1649186867217})]
```

Looks good! Time to run the real test on some real data. Run another inference this time from the file `high_fraud.json` and let's see the results:

```
[12]: ccfraud_pipeline.infer_from_file('./high_fraud.json')
```

```
[12]: [InferenceResult({'check_failures': [],
  'elapsed': 132117,
  'model_name': 'ccfraud-model',
  'model_version': 'd145cd72-8c0c-433c-9966-5572a589f743',
  'original_data': {'tensor': [[1.0678324729342086,
                                 18.155556397512136,
                                 -1.658955105843852,
                                 5.2111788045436445,
                                 2.345247064454334,
                                 10.467083577773014,
                                 5.0925820522419745,
                                 12.82951536371218,
                                 4.953677046849403,
                                 2.3934736228338225,
                                 23.912131817957253,
                                 1.7599568310350209,
                                 0.8561037518143335,
                                 1.1656456468728569,
                                 0.5395988813934498,
                                 0.7784221343010385,
                                 6.75806107274245,
                                 3.927411847659908,
                                 12.462178276650056,
                                 12.307538216518656,
                                 13.787951906620115,
                                 1.4588397511627804,
                                 3.681834686805714,
                                 1.753914366037974,
                                 8.484355003656184,
                                 14.6454097666836,
                                 26.852377436250144,
                                    2.716529237720336,
                                    3.061195706890285]]},
  'outputs': [{'Float': {'data': [0.9811990261077881], 'dim': [1, 1], 'v': 1}}],
  'pipeline_name': 'ccfraud-pipeline',
  'time': 1649186867233})]
```

With our work in the pipeline done, we'll undeploy it to get back our resources from the Kubernetes cluster. If we keep the same settings we can redeploy the pipeline with the same configuration in the future.

[13]:
```
ccfraud_pipeline.undeploy()
```

[13]:
```
{'name': 'ccfraud-pipeline', 'create_time': datetime.datetime(2022, 4, 5, 19,
27, 31, 934255, tzinfo=tzutc()), 'definition': "[{'ModelInference': {'models':
[{'name': 'ccfraud-model', 'version': 'd145cd72-8c0c-433c-9966-5572a589f743',
'sha': 'bc85ce596945f876256f41515c7501c399fd97ebcb9ab3dd41bf03f8937b4507'}]}}]"}
```

And there we have it! Feel free to use this as a template for other models, inferences and pipelines that you want to deploy with Wallaroo!