thampi Documentation

theScore Inc.

Nov 06, 2018

Contents:

thampi creates a machine learning prediction server on AWS Lambda.

CHAPTER 1

Overview

1.1 Introduction

Thampi creates a serverless machine learning prediction system using Zappa.

1.2 Benefits

- **No Devops**. With a single command, create a web server that scales, is fault tolerant and zero maintenance(courtesy AWS Lambda).
- **Focus on your model**. Work in Python to train your model. Thampi provides all support for serving the model.
- __Work on any platform*__. Work on Mac(technically possible on other platforms but untested) and still deploy to AWS Lambda(which is Linux).

  Thampi does this by using Docker underneath so that you can work on a Mac(or Windows?). When serving the model, it recreates your machine learning project in the docker image(which is Linux) and compiles all the C libraries(e.g. numpy) for you to Linux binaries, which is then uploaded to AWS Lambda.
- Pip and Conda support(See Limitations below).

1.3 Limitations

- Conda support is only for dependency files crafted by hand
- Max 500MB disk space. Ensure that your project with it's libraries is below this size.
- **Max 900 MB model size**. This number was derived from a few live tests. We circumvent the 500 MB disk space limit by loading the model directly from S3 to memory. `thampi` tries to reduce repeated calls to S3 by using `zappa's` feature of pinging the lambda instance every 4 mins or so(configurable). In that way, the model will stay on the lambda instance(unless it's a first time load or if AWS Lambda does decide to evict the instance for other reasons)

Based on feedback, an option could be added to package the model with the code but you'll then have to have a very small model size. It depends on what framework you use but generally you may have 100-250 MB space for your model as machine learning libraries take up a lot of space. Also look at the section `Design`

1.4 Alternatives

1.4.1 Sagemaker

SageMaker has the following advantages(not exhaustive):

- If you have SLAs, then Sagemaker may be a good choice. As they are on demand instances, you won't have the cold start delays.
- GPU Inference available
- Can serve models bigger than 1 GB(upon filling a form)
- You can choose more powerful machines than what AWS Lambda can offer you.

Sagemaker has the following costs(correct me if I am wrong):

- If you have to deploy a model which is not supported by Sagemaker(e.g. `lightfm`), then you have to:
  - create your own docker image(manage your own OS updates e.g. security)
  - implement a web server which has to implement a few endpoints
  - manage auto scaling,
  - provide handlers to `SIGTERM` and `SIGKILL` events
  - manage some environment variables.

For more details, look here.

`thampi`(via AWS Lambda) abstracts all this away from you. Of course, one can build a library to do the above for Sagemaker for you. Let me know when you make that library!

1.4.2 Algorithmia

Algorithmia is a commercial alternative to Thampi. I'm not sure though how easy it is to deploy models on libraries not supported by Algorithmia(e.g. `lightfm`). Note: I haven't spend too much researching Algorithmia.

CHAPTER 2

Installation

```
pip install thampi
```

2.1 Prerequisites

- Setup AWS Credentials
- Install Docker(e.g for Mac)
- Ensure that the user which calls thampi has *admin* access
- For now, only python 3.6 is supported

2.2 Important

- Do not install `zappa` or `flask` for your environment for serving models. If you have to use `zappa` or `flask`, consider creating another environment(and it's corresponding `requirements.txt` file)
- Do not upgrade `zappa` or `flask` or `pip` if asked to do so. The latest version of `zappa(0.46.1)` does not work for our use case so it's at `0.45.1`. As a `thampi` user, you should not be much affected by that.

2.3 What's Next

Step ahead to the tutorial

CHAPTER 3

Tutorial

3.1 Prerequisites

- First ensure you have gone through all the installation steps
- Your python installation should be 3.6, since AWS Lambda supports 3.6 as of now

Let's create a new project `myproject`. We'll use `scikit-learn` as an example but you could use any framework.

3.2 Dummy Project

3.2.1 Setup(Pip)

```
mkdir myproject && cd myproject
virtualenv -p python3 venv
source ./venv/bin/activate
pip install thampi
pip install scikit-learn
pip install numpy
pip install scipy
pip freeze > requirements.txt
```

3.2.2 Setup(Conda)

Note: This is one way of creating a conda environment. Please use the conventional way if you are comfortable in that style.

```
mkdir myproject && cd myproject
# Create a  conda environment inside the directory myproject
conda create --prefix=venv python=3.6.7
```

(continues on next page)

(continued from previous page)

```
pip install thampi
pip install scikit-learn
pip install numpy
pip install scipy
```

IMPORTANT: `thampi` only supports conda requirements files crafted by hand. So, let's manually create a requirements file with the above dependencies as shown below and save it as `requirements.txt`. The versions will change but you get the idea.

```
name: thampi-tutorial
dependencies:
  - thampi=0.1.0
  - numpy=1.15.*
  - scikit-learn=0.20.0
  - scipy=1.1.0
```

3.2.3 Initialization

- Run `thampi init` and you should see something similar to the terminal output below.
  - For the s3 bucket, you can choose to have one bucket for all your thampi applications. Each project(model) is at a different prefix so as long as the projects have unique names, they won't overwrite each other. If you aren't confident of that, you could just give a different bucket for each thampi project.
  - Choose `pip` or `conda` according to your preference.

```
thampi init
```

```
Welcome to Thampi!
-------------
Enter Model Name. If your model name is 'mymodel', the predict endpoint will be␣
↪myendpoint.com/mymodel/predict
What do you want to call your model: mymodel
-----------------

AWS Lambda and API Gateway are only available in certain regions. Let's check to make␣
↪sure you have a profile set up in one that will work.
We found the following profiles: analytics, and default. Which would you like us to␣
↪use? (default 'default'): default
------------

Your Zappa deployments will need to be uploaded to a private S3 bucket.
If you don't have a bucket yet, we'll create one for you too.
What do you want to call your bucket? (default 'thampi-2i1zp4ura'): thampi-store
-----------------
Enter package manager:['conda', 'pip'](default: pip):pip
A file zappa_settings.json has been created. If you made a mistake, delete it and run␣
↪`thampi init` again
```

- It has created a file called `zappa_settings.json`. This file is used by the Zappa framework. You'll note that some defaults have been filled up which are suitable for machine learning projects. A notable setting is `keep_warm` which prevents AWS Lambda from evicting the instance due to lack of use, by pinging the lambda(e.g. every 4 minutes). This is useful in the case when you have very large models. However, you could take it out if you feel that your model is small enough. For more details on how you can customize `zappa_settings.json`, check out zappa docs

- Within `zappa_settings.json`, thampi adds a key `thampi`. All thampi specific settings will go here. Note: `zappa` has no idea of `thampi`. It's just a convenient place to store the `thampi` relevant configuration.

3.3 Training

Inside `myproject`, copy the following code into the file `train.py`

```
import numpy as np
from sklearn import datasets
from typing import Dict
import thampi
from sklearn.neighbors import KNeighborsClassifier

class ThampiWrapper(thampi.Model):
    def __init__(self, sklearn_model):
        self.sklearn_model = sklearn_model
        super().__init__()


    def predict(self, args: Dict, context) -> Dict:
        original_input = [args.get('input')]
        result = self.sklearn_model.predict(np.array(original_input))
        return dict(result=int(list(result)[0]))

def train_model():
    iris = datasets.load_iris()
    iris_X = iris.data
    iris_y = iris.target
    np.random.seed(0)
    indices = np.random.permutation(len(iris_X))
    iris_X_train = iris_X[indices[:-10]]
    iris_y_train = iris_y[indices[:-10]]

    knn = KNeighborsClassifier()
    knn.fit(iris_X_train, iris_y_train)
    return ThampiWrapper(knn)


if __name__ == '__main__':
    model = train_model()
    thampi.save(model, 'iris-sklearn', './models')
```

- The above code first trains the `sklearn` model as `knn`. To make the `thampi` web framework send the request data to the model, we wrap `knn` in `ThampiWrapper`, a class which implements the `thampi.Model` interface. The data sent to the serving endpoint will be passed by `thampi` to the `predict` method as `args`. Likewise, one can wrap models of other libraries as well. Ignore the `context` argument in the `predict` method for now. The `context` object sends in the `Flask` application object(and others in the future) which is probably not required for most of the use cases for now.

And then at the terminal run

```
python train.py
```

This will create the model and save it locally using `thampi.save`. In `thampi`, like `mlflow`, the model artifacts

are stored in a directory(i.e. `iris-sklearn`). Storing it in the `models` directory is just arbitrary convention.

3.4 Serving the model

Now it's time to upload the model to AWS Lambda. All you have to provide is the `requirements.txt` file along with the above trained `./models/iris-sklearn` directory.

```
thampi serve staging --model_dir=./models/iris-sklearn --dependency_file=./
↪requirements.txt
```

The `serve` command will use `zappa` internally to create or update a server endpoint. To see the endpoint, do

```
thampi info staging
```

You'll see something similar to:

```
{'url': 'https://8i7a6qtlri.execute-api.us-east-1.amazonaws.com/staging/mymodel/
↪predict'}
```

Let's hit the endpoint in the next section.

3.5 Predict

You can do a curl like below where you replace `a_url` with the `url` that you receive from `thampi info staging`

```
curl -d '{"data": {"input": [5.9, 3.2, 4.8, 1.8]}}' -H "Content-Type: application/json
↪" -X POST a_url
```

`data` is a keyword here. Anything passed to `data` will be send along to your model. The dictionary with the key `input` depends on your application. It could have been something else like `features` for e.g. If you remember from the `ThampiWrapper` code above, since we use `input`, our code reads the data as `args.get('input')`

Output:

```
{
  "properties": {
    "instance_id": "9dbc56dd-936d-4dff-953c-8c22267ebe84",
    "served_time_utc": "2018-09-06T22:03:09.247038",
    "thampi_data_version": "0.1",
    "trained_time_utc": "2018-09-06T22:03:04.886644"
  },
  "result": {
    "result": 2
  }
}
```

For convenience, you can also do:

```
thampi predict staging --data='{"input": [5.9, 3.2, 4.8, 1.8]}'
```

where `data` is of `json` format.

The `properties` dictionary is meta-data associated with the model. Most of them are populated using the `save` command. If you want to add custom data (e.g `name` for your model and `version`, you can add it within `tags`)

3.6 Undeploy

After you are done with your project, the `zappa` command will bring down the server endpoint permanently. Note, we are using a `zappa` command. Zappa offers other relevant commands as well. Refer to the zappa docs. `thampi` offers a `cleanup` command as well which cleans up `thampi` relevant files locally and remotely where possible.

```
zappa undeploy staging
thampi clean --scope=project  # or --scope=all
```

NOTE: You may still have to clean up/delete the S3 bucket mentioned in `zappa_settings.json` for now.

CHAPTER 4

Thampi Model API

CHAPTER 5

Thampi Local API

CHAPTER 6

Thampi Model API

CHAPTER 7

Design

`thampi` being on AWS Lambda comes with constraints

7.1 Model Size

If your model is too big and serving it takes a lot of time, consider

- Pushing the model weights to a database. The author of lightfm has had success in that direction.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

CHAPTER 9

Credits

Thanks to theScore Inc. for supporting this project.