Runway Model SDK

Release v0.1.0

Runway AI, Inc.

Jun 12, 2019

CONTENTS

These documents serve as a reference for the Runway Model SDK. With a few lines of code, you can port existing ML models to the Runway platform so they can be used and shared by others.

CHAPTER

ONE

INSTALLING

This SDK supports both Python 2.7 and Python 3, but we recommend using Python 3. You can install the module using either `pip` or `pip3` like so:

```
pip3 install runway-python
```

Published versions of the SDK are hosted on the PyPI project website.

CHAPTER

TWO

RUNWAY MODELS

A Runway model consists of two special files:

- `runway_model.py`: A python script that imports the `runway` module (SDK) and exposes its interface via one or more `runway.command()` functions. This file is used as the **entrypoint** to your model.
- `runway.yml`: A configuration file that describes dependencies and build steps needed to build and run the model.

2.1 Example `runway_model.py`

Runway models expose a standard interface that allows the Runway app to interact with them over HTTP. This is accomplished using three functions: `@runway.setup()`, `@runway.command()`, and `runway.run()`.

Any Python-based model, independent of the ML framework or toolkit, can be converted into a Runway model using this simple interface. For more information about the `runway` module, see the module reference page.

Note: This is example code for demonstration purposes only. It will not run, as the `your_image_generation_model` import is not a real python module.

```
import runway
from runway.data_types import category, vector, image
from your_image_generation_model import big_model, little_model

# The setup() function runs once when the model is initialized, and will run
# again for each well formed HTTP POST request to http://localhost:8000/setup.
@runway.setup(options={'model_size': category(choices=['big', 'little'])})
def setup(opts):
    if opts['model_size'] == 'big':
        return big_model()
    else:
        return little_model()

inputs = { 'noise_vector': vector(length=128) }
outputs = { 'image': image(width=512, height=512) }

# The @runway.command() decorator is used to create interfaces to call functions
# remotely via an HTTP endpoint. This lets you send data to, or get data from,
# your model. Each command creates an HTTP route that the Runway app will use
# to communicate with your model (e.g. POST /generate). Multiple commands
# can be defined for the same model.
@runway.command('generate', inputs=inputs, outputs=outputs)
```

(continues on next page)

(continued from previous page)

```
def generate(model, input_args):
    # Functions wrapped by @runway.command() receive two arguments:
    # 1. Whatever is returned by a function wrapped by @runway.setup(),
    #    usually a model.
    # 2. The input arguments sent by the remote caller via HTTP. These values
    #    match the schema defined by inputs.
    img = input_args['image']
    return model.generate(img)

# The runway.run() function triggers a call to the function wrapped by
# @runway.setup() passing model_options as its single argument. It also
# creates an HTTP server that listens for and fulfills remote requests that
# trigger commands.
if __name__ == '__main__':
    runway.run(host='0.0.0.0', port=8000, model_options={ 'model_size': 'big' })
```

If you are looking to port your own model, we recommend starting from our Model Template repository hosted on GitHub. This repository contains a basic model that you can use as boilerplate instead of having to start from scratch.

2.2 Example `runway.yml`

Each Runway model must have a `runway.yml` configuration file in its root directory. This file defines the steps needed to build and run your model for use with the Runway app. This file is written in YAML, a human-readable superset of JSON. Below is an example `runway.yml` file. This example file illustrates how you can provision your model's environment.

```
version: 0.1
python: 3.6
entrypoint: python runway_model.py
cuda: 9.2
framework: tensorflow
files:
    ignore:
        - image_dataset/*
build_steps:
    - pip install runway-python
    - pip install -r requirements.txt
```

Continue on to the Runway YAML reference page to learn more about the possible configuration values supported by the `runway.yml` file.

2.2.1 Runway YAML File

A `runway.yml` file is required to be in the root file directory of each Runway model. This file provides instructions for defining an environment, installing dependencies, and running your model in a standard and reproducible manner. These instructions are used by the Runway build pipeline to create a Docker image that can be run in a platform independent way on a local machine or a remote GPU cloud instance (although this process is abstracted away from the model builder). The `runway.yml` config file is written in YAML and has a simple structure. You are free to copy and paste the example config file below, changing or removing the values that you need.

Note: The Runway configuration file must be named `runway.yml` and exist in the root (top-level) directory of your

model. Also, make sure you use the `.yml` file extension as the alternative `.yaml` extension is not supported.

Example

```yaml
# Specify the version of the runway.yml spec.
version: 0.1
# Supported python versions are 2.7 and 3.6
python: 3.6
# The command to run your model. This value is used as the CMD value in
# the generated Docker image.
entrypoint: python runway_model.py
# Which NVIDIA CUDA version to use. Supported versions include 10, 9.2, and 9.
cuda: 9.2
# Which ML framework would you like to pre-install? The appropriate GPU/CPU
# versions of these libraries are selected automatically. Accepts values
# "tensorflow" and "pytorch", installinv Tensorflow v1.12 and Pytorch v1.0
# respectively.
framework: tensorflow
# Builds are created for CPU and GPU environments by default. You can use the
# spec object to limit your builds to one environment if you'd like, for
# instance if your model doesn't use CUDA or run on a GPU you can set
# gpu: False.
spec:
    cpu: True
    gpu: True
files:
    # All files in the root project directory will be copied to the Docker image
    # automatically. Builds that require excessive storage can fail or take a
    # very long time to install on another user's machine. You can use the
    # files.ignore array to exclude files from your build.
    ignore:
        - my_dataset/*
        - secrets.txt
# The build_steps array allows you to run shell commands at build time. Each
# Each build step is executed in the order it appears in the array.
build_steps:
    # We recommend pinning to a specific version of the Runway Model SDK until
    # the first major release, as breaking changes may be introduced to the SDK
    - pip install runway-python==0.0.74
    - pip install -r requirements.txt
    # The if_gpu and if_cpu directives can be used to run build steps
    # conditionally depending on the build environment.
    - if_gpu: echo "Building in a GPU environment..."
    - if_cpu: echo "Building in a CPU only environment..."
```

Note: If you require an ML framework other than Tensorflow or Pytorch, or a version of these libraries that is different than the versions provided by the `frameworks` object, you can install these dependencies manually in the build steps.

```yaml
build_steps:
    - pip install tensorflow==1.0
    - if_gpu: pip install tensorflow-gpu==1.0
```

Schema Reference

- `version` (int, optional, default = `0.1`): This version specifies the schema of the configuration file not the version of the Runway Model SDK itself.
- `python` (float, **required**): The Python version to use when running the model installing python dependencies. Currently supported values are `2.7` and `3.6`.
- `entrypoint` (string, **required**): The command to run your model. This value is used as the CMD value in the generated Docker image. A standard value for this field might be `entrypoint: python runway_model.py` where `runway_model.py` implements the `@runway.setup()`, `@runway.command()`, and most importantly the `runway.run()` functions.
- `cuda` (float, **required if building for GPU**): The NVIDIA CUDA version to use in the production GPU runtime environment. The currently supported CUDA versions are `10`, `9.2`, and `9`.
- `framework` (string, optional, default = `None`): The machine learning framework to pre-install during the build. Currently we support `"tensorflow"` and `"pytorch"` which will install the appropriate CPU or GPU packages of Tensorflow v1.12.0 and Pytorch v1.0 respectively depending on the build environment. If you require an ML framework other than Tensorflow or Pytorch, or a version of these libraries that is different than the versions provided by the `frameworks` object, you can omit this object and install these dependencies manually in the build steps.
- `spec` (object, optional): A dictionary of boolean values specifying which CPU/GPU environments to build for. Both the `cpu` and `gpu` environments are enabled (`True`) by default.
  - `cpu` (boolean, optional, default = `True`): Create a CPU build.
  - `gpu` (boolean, optional, default = `True`): Create a GPU build.
- `files` (object, optional): A dictionary that defines special behaviors for certain files. All values in this dictionary are specified as paths, with support for the glob character (e.g. `data/*.jpg`).
  - `ignore` (array of strings, optional): A list of file paths to exclude from the build.
- `build_steps` (array of strings or dictionary values containing the `if_cpu` and `if_gpu` keys, optional): A list of shell commands to run at build time. Use this list to define custom build steps. Build steps are run in the order they appear in the array. The `if_gpu` and `if_cpu` directives can be used to run build steps conditionally depending on the build environment.

2.2.2 Runway Module

The Runway module exposes three simple functions that can be combined to expose your models to the Runway app using a simple interface.

- *`@runway.setup()`*: A Python decorator used to initialize and configure your model.
- *`@runway.command()`*: A Python decorator used to define the interface to your model. Each command creates an HTTP route which can process user input and return outputs from the model.
- *`runway.run()`*: The entrypoint function that starts the SDK's HTTP interface. It fires the function decorated by `@runway.setup()` and listens for commands on the network, forwarding them along to the appropriate functions decorated with `@runway.command()`.

Reference

2.2.3 Data Types

The Runway Model SDK provides several data types that can be used to pass values to and from runway models and the applications that control them. The data types currently supported by the SDK are `number`, `text`, `image`, `array`, `vector`, `category`, `file`, and `any`, an extensible data type. These data types are primarily used in two places:

- The `options` parameter in the `@runway.setup()` decorator
- The `input` and `output` parameters in `@runway.command()` decorator

Note: This is example code for demonstration purposes only. It will not run, as the `your_code` import is not a real python module.

```
import runway
from runway.data_types import category, vector, image
from your_code import model

options = {"network_size": category(choices=[64, 128, 256, 512], default=256)}
@runway.setup(options=options)
def setup(opts):
    return model(network_size=opts["network_size"])


sample_inputs= {
    "z": vector(length=512),
    "category": category(choices=["day", "night"])
}

sample_outputs = {
    "image": image(width=1024, height=1024)
}

@runway.command("sample", inputs=sample_inputs, outputs=sample_outputs)
def sample(model, inputs):
    img = model.sample(z=inputs["z"], category=inputs["category"])
    # `img` can be a PIL or numpy image. It will be encoded as a base64 URI
    # string automatically by @runway.command().
    return { "image": img }

if __name__ == "__main__":
    runway.run()
```

Reference

2.2.4 Exceptions

The Runway Model SDK defines several custom exception types. All of these exceptions derive from the base `RunwayError` class, which itself derives from the standard Python `Exception` class. Each `RunwayError` contains error `message` and `code` properties, and a `to_response()` method that converts the exception to a Python dictionary, which can be returned as a JSON HTTP response.

```
try:
    do_something_with_runway()
except RunwayError as e:
    print(e.code, e.message)
    # 500 An unknown error has occurred
    print(e.to_response())
    # { "error": "An unknown error has occurred", "traceback": "..." }
```

Reference