Matt Williamson
CS410
October 25, 2020

MeTAPyquod

Containerized metapy Appliances

The object of this project is to simplify the use of MeTA and the metapy bindings by creating pre-packaged docker containers ready to use with all dependencies for different use cases. The result will be at least the following two reusable containers:

`metapyquod-dev`

This container will contain the metapy libraries and a functional development environment including a known good version of Python, metapy dependencies and useful related libraries (e.g. scipy, numpy), Git version control binaries, and a Linux shell, suitable for experimentation and basic development (i.e., all the CS410 programming assignments should be able to be completed with only the tools in the container). This container (or a derivative) *may* also include a GUI IDE such as Spyder and its dependencies to better facilitate development work. A particular goal of this container is to simplify use of the toolchain on Windows with Docker (and/or Windows Subsystem for Linux), and the hope is that with Docker's new "multi-arch" features that this and running on ARM devices (e.g. Raspberry Pi) will be possible. If this goal is achieved, such users will be able to instantiate a full working environment for metapy development with a single `"docker run ..."` command.

`metapyquod-server`

This container will provide a simple search engine core as a microservice behind a REST API. The goal is for this container to be capable enough to serve in basic production use as a simple search appliance, for instance in an intranet search scenario. It will also facilitate experimentation with different metapy components in a real world use case. The search service will be intended for use with English language corpora.

This container will expose a Volume where it expects to find a web mirror filesystem structure according to the conventions produced by the wget spider process. This will allow the user to use the basic but flexible wget tool (likely in another container) to crawl and download web content for ingest by the `metapyquod-server` appliance. Since wget supports timestamping of downloaded files (and requery culling based on `Last-Modified` headers, the appliance will monitor the Volume for newly modified files, gaining some efficiencies.

In addition to providing sane defaults, this container will also expose a volume with configuration control data (e.g. a `config.toml` or other relevant files) to facilitate customization of or experimentation with the search appliance.

The search service itself will be written in Python (probably with an existing HTTP framework like Django) and will expose at least the following three capabilities as HTTP REST services:

1. A search service, with a text string parameter and pagination parameters. This service will return JSON-formatted search results including the URL, and will include a query identifier for potential use with a feedback mechanism.
2. A feedback service for registering click-throughs for implicit feedback. The viewed URL and the query identifier from the search service will be received as inputs. *This service will likely have no effect in the initial iteration, but is established for future implementation of feedback incorporation.*
3. A telemetry service providing basic information about the search index for diagnostic or engagement purposes (e.g. last 10 URLs indexed, total documents/terms in index, etc.)

A Swagger/OpenAPI specification will be provided such that a compliant client should be able to interact with the service—it should be possible to test the service using a generic Swagger/OpenAPI client. If time permits, a simple example web frontend demonstrator for the service may be provided.

Estimates

metapyquod-dev: Design and build container specification with dependencies, deploy and test on all target environments, create documentation: 4-6 hours.

metapyquod-server: Develop and implement REST services: 10-15 hours. Package and test containerization, create documentation: 4-8 hours.