redis

In-memory database and message broker

What is it?

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

from redis website

Server

- Ubuntu

```
sudo add-apt-repository ppa:redislabs/redis
sudo apt-get update
sudo apt-get install redis
```

- Docker

  https://hub.docker.com/_/redis/

- Source

Clients

- Supported languages

  ActionScript, ActiveX/COM+, Bash, Boomi, C, C#, C++, Clojure, Common Lisp, Crystal, D, Dart, Delphi, Elixir, emacs lisp, Erlang, Fancy, gawk, GNU Prolog, Go, Haskell, Haxe, Io, Java, Julia, Lasso, Lua, Matlab, mruby, Nim, Node.js, Objective-C, Ocaml, Pascal, Perl, PHP, PL/SQL, Prolog, Pure Data, **Python**, R, Racket, Rebol, Ruby, Rust, Scala, Scheme, Smalltalk, Swift, Tcl, VB, VCL, Xojo, Zig

Setup client

- Create virtual environment

```
virtualenv -p /usr/bin/python3 ./venv
python3 -m venv ./venv
```

- Install client (redis-py)

```
./venv/bin/pip install redis
```

Connect

- Need host, port and DB

```
import redis
r = redis.Redis(host="localhost", port=6379, db=0)
```

- SSL possible
  - no hostname verification (not recommended)
  - explicite certificate file
  - certifi (Mozilla's curated list of Root certs)

Key-value store

- Simplest use case:

  store/retrieve data via keys

- Example

  import redis

  r = redis.Redis(host="localhost", port=6379, db=0)

  r.**set**("foo", "bar")

  print(r.**get**("foo"))

- Uses bytes, so *.decode()* necessary

Message broker

- redis offers pub/sub framework
- Subscribe to one or more channels for listening

```
import redis
r = redis.Redis(host="localhost", port=6379, db=0)  # connect

def msg_handler(message):   # message handling function
    print(message)

p = r.pubsub()
p.psubscribe(**{"my_*": msg_handler})  # p.subscribe(**{"my_messages": msg_handler})
p.run_in_thread(sleep_time=0.001)
```

Message broker (2)

- Broadcast messages

```
import redis
r = redis.Redis(host="localhost", port=6379, db=0)
r.publish("my_channel1", "1st message")
r.publish("my_channel2", "2nd message")
```

Lists

- Operations
  - add/remove from head/tail
  - length of list
  - get slice of list, get item by index
  - set item via index
  - trim list
  - ...

Lists (2)

```
r.rpush("l1", "c", "b", "a")  # add three strings at end ("right")
print(r.llen("l1"))
print([x.decode() for x in r.lrange("l1", 0, -1)])

r.lpush("l1", 1, 2, 3)  # add three numbers (as strings) at head ("left") in reverse order (!)
print(r.llen("l1"))
print([x.decode() for x in r.lrange("l1", 0, -1)])

r.ltrim("l1", 2, 4)  # keep third to fifth index (both incl)
print(r.llen("l1"))
print([x.decode() for x in r.lrange("l1", 0, -1)])

r.lset("l1", 0, 11)  # replace value at index 0 with 11 (string!)
print([x.decode() for x in r.lrange("l1", 0, -1)])
```

Other things

- inc/dec values
- sets
- hash maps
- bit operations
- pipelining (reduce round trip time)
- ...

Application

- Use pub/sub for integrating deep learning model (separate process)
- Model classifies images (flowers)
- Python code broadcasts images
- Model receives images and broadcasts predictions
- Python code responds to predictions

Application (2)

- virtual environment

```
virtualenv -p /usr/bin/python3 ./venv
./venv/bin/pip install wai.tflite_model_maker
```

- Train model

```
./venv/bin/tmm-ic-train --images data/ --num_epochs 5 --output output/
```

- Use model

```
./venv/bin/tmm-ic-predict-redis \
    --model ./output/model.tflite --labels ./output/labels.txt \
    --redis_in images --redis_out predictions --verbose
```

Application (3)

```
import redis
r = redis.Redis(host="localhost", port=6379, db=0)

def msg_handler(message):
    print(message["data"])

p = r.pubsub()
p.psubscribe(**{"predictions": msg_handler})
p.run_in_thread(sleep_time=0.001)

images = [
    "./data/alpine_sea_holly/image_06969.jpg",
    "./data/anthurium/image_01964.jpg",
    "./data/artichoke/image_04081.jpg"]
for image in images:
    with open(image, "rb") as f:
        data = f.read()
        r.publish("images", data)
```