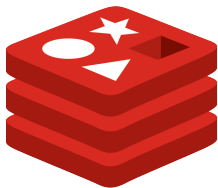


# COSC 416 - Redis

Paul Moore, Stephen Smithbower, William Lee

March 11, 2013



redis

# What is Redis?

- Redis is an **in-memory** key-value data store.
- Can be a middle-ware solution between your expensive persistent data-store (Oracle), and your application.
- Provides PubSub, scripting, persistence models, transactions, etc.
- Supports additional data types beyond Strings.

# Why Redis?

- In-memory, so very fast.
- Built-in data types suit using Redis for intra-process communication.
  - ① Cache
  - ② Queue
  - ③ Message-Passing
- Supports journaling and snapshotting - more than just volatile cache (i.e. memcache).
- Clients communicate via String commands over sockets ∴ client agnostic.

# Redis data types

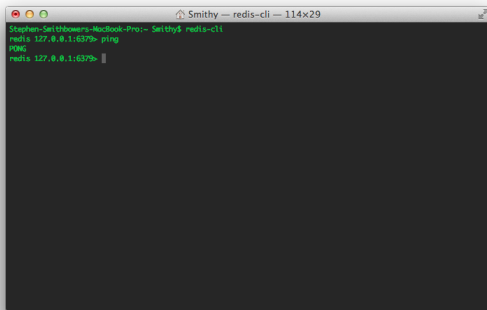
Redis supports several data types internally:

- 1 String
- 2 List
- 3 Set
- 4 Sorted Set
- 5 Hash

Because of this, Redis is often referred to as a **data structure server**

**Note:** Redis returns all values as Strings, in addition to the return type. Clients are responsible for type-conversion (some client libraries handle this for you).

# Redis CLI



```
Stephen-Smithbowers-MacBook-Pro:~ Smithy$ redis-cli
redis: 127.0.0.1:6379> ping
pong
redis: 127.0.0.1:6379> ping
```

Redis has a standard interactive CLI tool.

Start: **redis-cli**

To check if the server is alive, use **ping**. You should get a **pong** response.

# Example Redis Commands

**LPUSH mylist a** The list now contains "a"

**LPUSH mylist b** The list now contains "b", "a"

**RPUSH mylist c** The list now contains "b", "a", "c"

**Note:** If *mylist* doesn't exist, Redis will create it and infer its type.

See: <http://redis.io/commands>

# Example Redis Commands

- LPUSH list1 a b c** The list now contains "a", "b", "c"
- RPOP list1** Returns "c". The list now contains "a", "b"
- LPOP list1** Returns "a". The list now contains "b"
- LREM list1 1 "b"** Returns "1" (one element removed). The list is now empty.
- BRPOP list1** Blocks the connection until list1 contains a value to pop. This is a blocking queue.

See: <http://redis.io/commands>

# Try Redis: Interactive Tutorial



Redis provides an online, interactive tutorial to practise using Redis commands.

You can find the tutorial at: <http://try.redis.io>



# Transactions

- Redis supports transactions (a series of commands will be executed serially and cannot be interrupted by other clients), with the **MULTI** and **EXEC**.
- A transaction is **atomic** (all commands will be executed, or none will), but Redis does not support rollback, so it is **not durable**.
- Redis supports optimistic Check-and-Set atomic operations with the **WATCH** command. If a watched variable is modified during a transaction, the transaction will fail. *It is up to the client to attempt the transaction again.*
- See: <http://redis.io/topics/transactions>

- Redis supports on-disk **snapshotting** and **logging**.
- **RDB Snapshotting:** Using the **SAVE** command, Redis will perform a stop-the-world clone of the entire in-memory dataset to-disk. Using **BGSAVE** will instead perform an asynchronous clone, but is not an atomic operation, and does not guarantee a consistent snapshot.
- **AOF Logging:** All writes to Redis are logged to disk (using the same command format that the CLI uses) in an append-only format. When Redis is restarted, this log is replayed.

- Redis contains a Publish/Subscribe messaging implementation!
- Clients may **SUBSCRIBE** to a channel. Messages can be written to the channel using **PUBLISH**. Messages written to a channel are broadcast to all subscribers.
- **Note:** There is no history kept for a channel. Clients will only receive new messages published to a given channel.
- **Note:** Once in PubSub mode (i.e. once a client is subscribed to at least one channel), no further Redis commands may be issued. Clients must **UNSUBSCRIBE** from all channels before issues new Redis commands.

- Redis supports it's full API through a Lua interface (Lua is an embedded scripting language).
- You can write scripts in Lua, and evaluate them with the **EVAL** command.
- The **EVALSHA** command will try to find a preloaded script with a matching SHA1 hash and execute that instead.

```
> EVAL "return KEYS[1] .. ARGV[1]" 1 key1 value1  
1) "key1value1"
```

- **URL:**

<https://docs.google.com/document/d/1tGcdzhb4uQzLByKcbXSyuFsst9dacUSHPY6wGCoDy8/edit>

- **Part I** Some sample Redis commands.
- **Part II** A short scripting question.
- **Part III** A simple chat client.

# Further Resources

- [redis.io](https://redis.io)
- [try.redis.io](https://try.redis.io)
- [lua.org](https://lua.org)