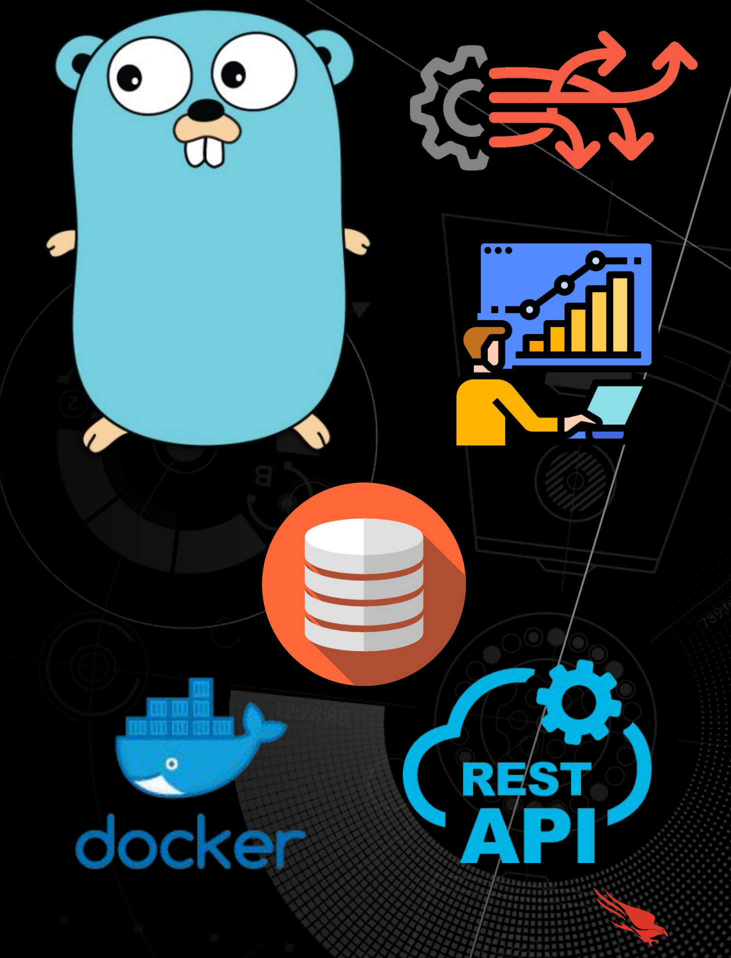
Databases (cont)
Redis

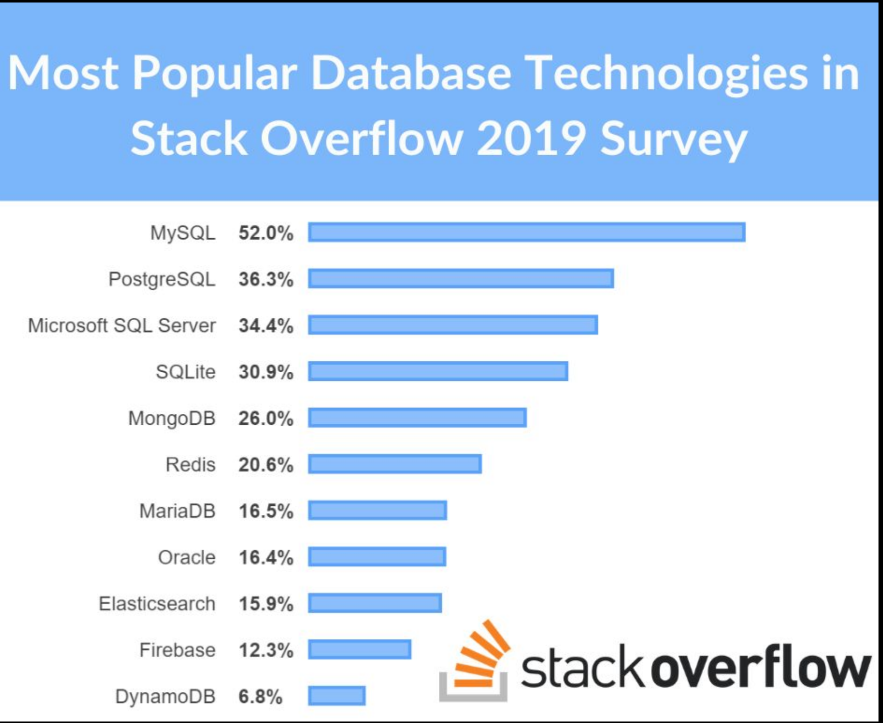
CrowdStrike HEROES - Cloud Workshop

Date	Topic
July 25 - 16:00	Intro to golang
July 26 - 16:00	Intro to golang (continuation)
July 27 - 16:00	Multithreading
July 28 - 16:00	Rest API
July 29 - 16:00	Unit testing, logging and monitoring
August 1 - 16:00	Workshop and Q&A
August 2 - 16:00	Deployments/Docker
August 3 - 16:00	Databases
August 4 - 16:00	**Databases extended**
August 5 - 13:00	Microservices contest (4h with Awards)



What is Redis

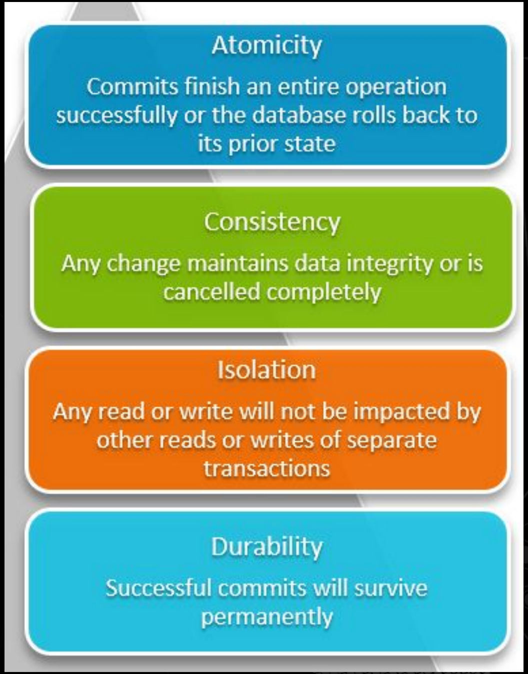
- Redis is open source
- in-memory data structure store
- used as a database, cache, message broker, and streaming engine
- Key-value based



Most Popular Database Technologies in Stack Overflow 2019 Survey

Redis ACID

- Each operation is atomic
- Redis Cluster does not guarantee strong consistency
    - it uses asynchronous replication
- If the transactions impact different keys they are independent
- Redis does offer some forms of persistence but no guarantees



Atomicity

Commits finish an entire operation successfully or the database rolls back to its prior state

Consistency

Any change maintains data integrity or is cancelled completely

Isolation

Any read or write will not be impacted by other reads or writes of separate transactions

Durability

Successful commits will survive permanently

Environment setup

Docker, Golang, Redis

redis-cli

Simple operations

- Adding values

```
127.0.0.1:6379> set a 1
OK
127.0.0.1:6379> set a "ala bala"
OK
```

- Retrieving values

```
127.0.0.1:6379> get a
"ala bala"
```

- Deleting values

```
127.0.0.1:6379> del a
OK
127.0.0.1:6379> get a
(nil)
```

Lists

- Adding values to a list

```
127.0.0.1:6379> lpush list 1
(integer) 1
127.0.0.1:6379> rpush list 2 3
(integer) 3
```

- Retrieving values

```
127.0.0.1:6379> lpop list
"1"
127.0.0.1:6379> rpop list
"3"
```

- Checking values

```
127.0.0.1:6379> lrange list 0 1
1) "1"
2) "2"
127.0.0.1:6379> lrange list 0 -1
1) "1"
2) "2"
3) "3"
127.0.0.1:6379> llen list
(integer) 3
```

Sets

- Adding values to a set

```
127.0.0.1:6379> sadd set1 1 2 3
(integer) 3
127.0.0.1:6379> sadd set1 1
(integer) 0
```

- Retrieving values

```
127.0.0.1:6379> spop set1
"3"
```

- Checking values

```
127.0.0.1:6379> smembers set1
1) "1"
2) "2"
127.0.0.1:6379> scard set1
(integer) 2
127.0.0.1:6379> srandmember set1
"2"
```

Multiple operations

There are two ways for this:

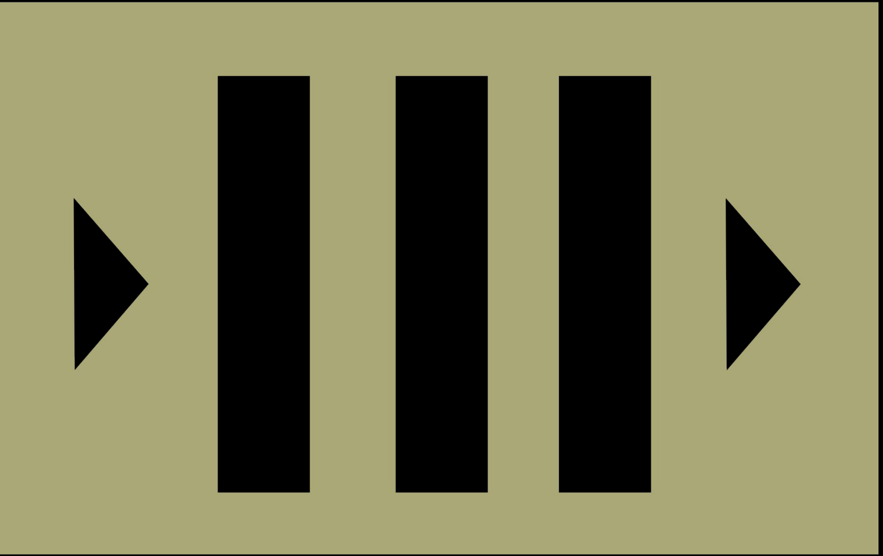
- Atomic transaction blocks

```
> MULTI
… // commands
> EXEC
```

- Lua scripts evaluation

```
> EVAL "return ARGV[1]" 0 hello
"hello"
```

Message queues*



There are dedicated applications for sending messages between services.

Examples:

- Rabbit MQ
- Kafka

These usually implement acknowledgements and retires.

Our way of doing queues

- Since it is easier for us to set up fewer repositories we can use redis as a message queue substitute
- For our queue we can use a list
- Usually features are separated by functionality using a ‘:’
  - lab09:cache:files_stored

Any questions?