

A horizontal bar spanning the width of the slide, divided into two equal halves: the left half is light gray and the right half is dark red.

Redis^e Pack Admin Training

Redis API

Agenda

- **Redis Data Store**
- Strings
- Lists
- Hashes
- Sets
- Sorted Set
- Advanced API introduction
- Redis CLI additional Features
- Lab

Redis Data Store

- A single very large data store
- Key can be any String/Byte Array
- Value can be any of the following types
 - Strings (Byte Arrays)
 - Lists
 - Sets
 - ZSET (Sorted Sets)
 - Hashes
 - Bitmap and HyperLogLogs

```
CREATE TABLE Redis (  
    k VARCHAR(512MB) NOT NULL,  
    v VARCHAR(512MB),  
    PRIMARY KEY (k)  
);
```

Key and Value

- Both key and value can be at max 512 Megabytes in length
 - Within complex data structure every value in the data structure can be at max 512 Megabytes
- Key/Value basic command families
 - Setters – associating a key with a value
 - A Setter command name is dependent on the data structure you use
 - Getters – specifying a key and returning the value
 - A getter command name is dependent on the data structure you use
 - [EXISTS](#) command returns 1 or 0 to signal if a given key exists or not in the database
 - [DEL](#) command deletes a key and associated value

Agenda

- Redis Data Store
- **Strings**
- Lists
- Hashes
- Sets
- Sorted Set
- Advanced API introduction
- Redis CLI additional Features
- Lab

Strings

- Strings are the most basic kind of Redis value
- Redis Strings are binary safe
 - A Redis string can contain any kind of data, for instance a JPEG image or a serialized Ruby object

```
> SET mykey somevalue
OK
> GET mykey
"somevalue"
```

SET and GET

- [SET](#) and the [GET](#) commands are the way we set and retrieve a string value
 - SET performs an assignment
 - SET will replace any existing value already stored into the key
- Values can be strings (including binary data) of every kind, for instance you can store a jpeg image inside a key

More on SET Command

- The [SET](#) command has interesting options, that are provided as additional arguments
- EX seconds
 - Set the specified expire time, in seconds
- PX milliseconds
 - Set the specified expire time, in milliseconds
- NX
 - Only set the key if it does not already exist
- XX
 - Only set the key if it already exist

```
> SET mykey somevalue
OK
> SET mykey newval nx
(nil)
> SET mykey newval xx
OK
```


Altering and querying the key space

- EXISTS command returns 1 or 0 to signal if a given key exists or not in the database
- DEL command deletes a key and associated value, whatever the value is
 - Returns 1 or 0 depending on whether the key was removed
- These commands can be used on all keys regardless of the value data structure

```
> SET mykey hello
OK
> EXISTS mykey
(integer) 1
> DEL mykey
(integer) 1
> EXISTS mykey
(integer) 0
```

String with Integer Values - INCR

- Use Strings as atomic counters using commands in the INCR family: [INCR](#), [DECR](#), [INCRBY](#), [DECRBY](#)
- Even if strings are the basic values of Redis, there are interesting operations you can perform with them
 - For instance, one is atomic increment

```
> SETcounter 100
OK
> INCR counter
(integer) 101
> INCR counter
(integer) 102
> INCRBY counter 50
(integer) 152
```

MSET and MGET

- [MSET](#) and [MGET](#) commands enable to set or retrieve the value of multiple keys in a single command is also useful for reduced latency

```
> MSET a 10 b 20 c 30
OK
> MGET a b c
1) "10"
2) "20"
3) "30"
```

- [GETSET](#) command sets a key to a new value, returning the old value as the result

- Check all the [available string commands](#) for more information, or read the [introduction to Redis data types](#)

Keys Expiration

- Set using seconds or milliseconds precision
- However the expire time resolution is always 1 millisecond
- Information about expires are replicated and persisted on disk, the time virtually passes when your Redis server remains stopped
 - Redis saves the date when a key will expire

```
> SET key some-value
OK
> EXPIRE key 5
(integer) 1
> TTL key
(integer) 4
> GET key (immediately)
"some-value"
> GET key (after some time)
(nil)
```

Agenda

- Redis Data Store
- Strings
- **Lists**
- Hashes
- Sets
- Sorted Set
- Advanced API introduction
- Redis CLI additional Features
- Lab

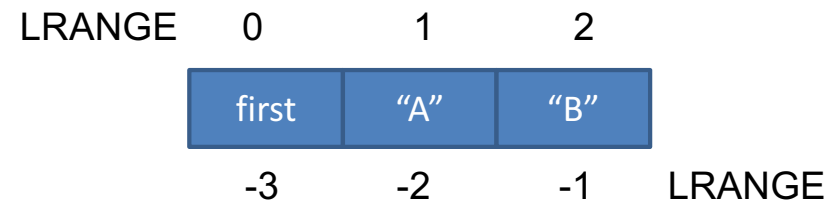
List

- A sequence of ordered elements
 - 10,20,1,2,3 is a list
- Redis lists are implemented via Linked Lists
 - Adding a new element in the head with the [LPUSH](#) command or in the tail of the list is performed *in constant time*
 - Same for a list with 10 or 1M elements
 - Every list value (Element) can be up to 512MB
- Accessing an element *by index* is not so fast in lists implemented by linked lists
- Redis Lists are implemented with linked lists

List – Basic API

- [LPUSH](#) command adds a new element into a list, on the left (at the head)
- [RPUSH](#) command adds a new element into a list, on the right (at the tail)
- [LRANGE](#) command extracts ranges of elements from lists
 - Takes two indexes, the first and the last element of the range to return
 - Both the indexes can be negative
 - -1 is the last element
 - -2 is the penultimate element of the list, and so forth

```
> RPUSH mylist A
(integer) 1
> RPUSH mylist B
(integer) 2
> LPUSH mylist first
(integer) 3
> LRANGE mylist 0 -1
1) "first"
2) "A"
3) "B"
```



List – Basic API continued

- [LPUSH](#) and [RPUSH](#) allow for variable number of parameters commands
 - You can push multiple elements into a list in a single call

```
> RPUSH mylist 1 2 3 4 5 "foo bar"
(integer) 9
> LRANGE mylist 0 -1
1) "first"
2) "A"
3) "B"
4) "1"
5) "2"
6) "3"
7) "4"
8) "5"
9) "foo bar"
```

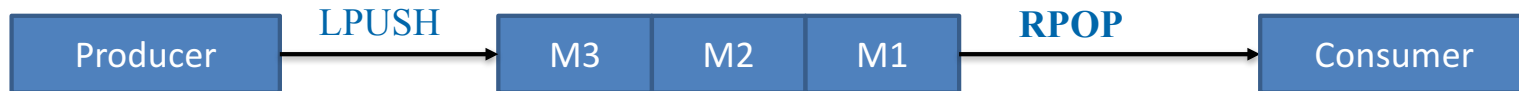

List – RPOP and LPOP

- Popping elements is the operation of both
 - retrieving the element from the list, and
 - eliminating it from the list

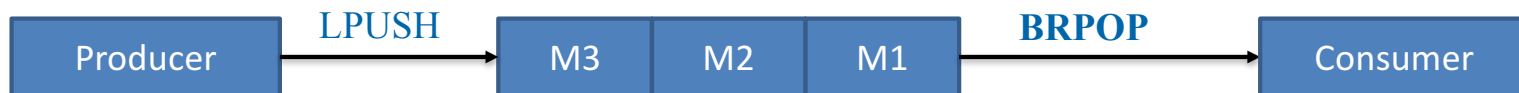
```
> RPUSH mylist a b c
(integer) 3
> RPOP mylist
"c"
> RPOP mylist
"b"
> RPOP mylist
"a"
> RPOP mylist
(nil)
```

List – common usages

- Remember the latest updates posted by users into a social network
- Communication between processes
 - using a consumer-producer pattern where the producer pushes items into a list, and a consumer (usually a *worker*) consumes those items and executed actions



- Redis has special list commands to make this use case both more reliable and efficient
 - [BRPOP](#) and [BLPOP](#) are blocking xPOP operation which will wait for a value to be in the list



Capped Lists using LTRIM

- Use the [LTRIM](#) command to use lists as a capped collection,
 - Only remembering the latest N items and discarding all the oldest items

```
> RPUSH mylist 1 2 3 4 5
(integer) 5
> LTRIM mylist 0 2
OK
> LRANGE mylist 0 -1
1) "1"
2) "2"
3) "3"
```

Agenda

- Redis Data Store
- Strings
- Lists
- **Hashes**
- Sets
- Sorted Set
- Advanced API introduction
- Redis CLI additional Features
- Lab

Hash

- Redis hashes data structure contains field-value pairs

Notice the key "user:1000" this way we can simulate a table name + PK in the single key. This is just like any other key and can be used anywhere in Redis.

Hash Fields

Hash Values

- API

- [HMSET](#)
- [HGET](#)
- [HMGET](#)

- Is similar to [HGET](#) but might return an array of values

- Hash value size < 512MB

```
> HMSET user:1000 username antirez birthyear 1977 verified 1
OK
> HGET user:1000 username
"antirez"
> HGET user:1000 birthyear
"1977"
> HGETALL user:1000
1) "username"
2) "antirez"
3) "birthyear"
4) "1977"
5) "verified"
6) "1"
```

Additional Hash API

- There are commands that are able to perform operations on individual fields as well, like [HINCRBY](#)

```
> HINCRBY user:1000 birthyear 10  
(integer) 1987  
> HINCRBY user:1000 birthyear 10  
(integer) 1997
```

- You can find the [full list of hash commands in the documentation](#).

Agenda

- Redis Data Store
- Strings
- Lists
- Hashes
- **Sets**
- Sorted Set
- Advanced API introduction
- Redis CLI additional Features
- Lab

SET Data Structure

- Unordered collections of strings
- SADD adds new elements to a set
 - It's also possible to do a number of other operations against sets like
 - testing if a given element already exists
 - performing the intersection, union or difference between multiple sets, and so forth

```
> SADD myset 1 2 3
(integer) 3
> SMEMBERS myset
1. 3
2. 1
3. 2
> SISMEMBER myset 3
(integer) 1
> SISMEMBER myset 30
(integer) 0
```


Set operations UNION etc...

- SUNION
 - Returns the members of the set resulting from the union of all the given sets
- SINTER
 - Returns the members of the set resulting from the intersection of all the given sets
- SDIFF
 - Returns the members of the set resulting from the difference between the first set and all the successive sets

```
> SADD key1 "a" "b" "c"
(integer) 3
> SADD key2 "c" "d" "e"
(integer) 3
> SUNION key1 key2
1) "b"
2) "a"
3) "c"
4) "e"
5) "d"
>
```

Agenda

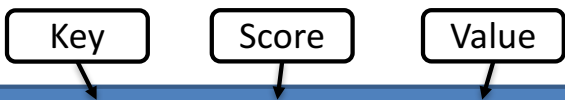
- Redis Data Store
- Strings
- Lists
- Hashes
- Sets
- **Sorted Set**
- Advanced API introduction
- Redis CLI additional Features
- Lab

Sorted Set

- A mix between a Set and a Hash
 - Like sets, sorted sets are composed of unique, non-repeating string elements
 - Every element in a sorted set is associated with a floating point value called *the score*
 - Similar to a hash, since every element is mapped to a value

Sort ordering

- Elements in a sorted sets are taken in order. They are ordered according to the following rule
 - If A and B are two elements with a different score, then $A > B$ if $A.score > B.score$
 - If A and B have exactly the same score, then $A > B$ if the A string is lexicographically greater than the B string. A and B strings can't be equal since sorted sets only have unique elements
- Use [ZADD](#) in order to add values to a sorted set
- Sorted sets support all set operations



```
> ZADD hackers 1912 "Alan Turing"
(integer) 1
> ZADD hackers 1916 "Claude Shannon"
(integer) 1
> ZADD hackers 1969 "Linus Torvalds"
(integer) 1
```

Some sorted set operations

Create a sorted set

```
> ZADD hackers 1940 "Alan Kay"
(integer) 1
> ZADD hackers 1957 "Sophie Wilson"
(integer) 1
> ZADD hackers 1953 "Richard Stallman"
(integer) 1
> ZADD hackers 1949 "Anita Borg"
(integer) 1
> ZADD hackers 1916 "Claude Shannon"
(integer) 1
> ZADD hackers 1969 "Linus Torvalds"
(integer) 1
> ZADD hackers 1912 "Alan Turing"
(integer) 1
```

Return sorted set automatically ordered by score

```
> ZRANGE hackers 0 -1
1) "Alan Turing"
2) "Claude Shannon"
3) "Alan Kay"
4) "Anita Borg"
5) "Richard Stallman"
6) "Sophie Wilson"
7) "Linus Torvalds"
```

Operate on ranges

- ZRANGEBYSCORE
- Example
 - Return all the elements with a score between negative infinity and 1950

```
> ZRANGEBYSCORE hackers -inf 1950  
1) "Alan Turing"  
2) "Claude Shannon"  
3) "Alan Kay"  
4) "Anita Borg"
```

Remove Elements in Range

- [ZREMRANGEBYSCORE](#)
- Remove elements in range by score and returns the number of removed elements

```
> ZREMRANGEBYSCORE hackers 1940 1960  
(integer) 4
```

- For more information on sorted sets
<http://redis.io/topics/data-types-intro#sorted-sets>

Scan

- Getting a list of keys
- The [SCAN](#) command and the closely related commands [SSCAN](#), [HSCAN](#) and [ZSCAN](#) are used in order to incrementally iterate over a collection of elements.

```
$ redis-cli --SCAN | head -10  
key-419  
key-71  
key-236  
key-50  
key-38
```

```
$ redis-cli --SCAN --pattern '*-11*'  
key-114  
key-117  
key-118  
key-113  
key-115
```

- You can use as a cursor for iterating key value pairs

Agenda

- Redis Data Store
- Strings
- Lists
- Hashes
- Sets
- Sorted Set
- **Advanced API introduction**
- Redis CLI additional Features
- Lab

Info

- The [INFO](#) command returns information and statistics about the server in a format that is simple to parse by computers and easy to read by humans

```
Redis > INFO
# Server
redis_version:999.999.999 redis_git_sha1:ceaf58df
redis_git_dirty:1 redis_build_id:a5eeeb464ee54856
redis_mode:standalone
os:Linux 4.1.5-x86_64-linode61 x86_64 arch_bits:32
multiplexing_api:epoll gcc_version:4.4.1
process_id:21798
run_id:2569bb7433bfe013c2627edf62d9bf21eaf8a010
...
```

Client List

- The [CLIENT LIST](#) command returns information and statistics about the client connections server in a mostly human readable format

```
Redis 7> CLIENT LIST
id=19 addr=10.0.3.57:41162 fd=64 name= age=59 idle=0 flags=N db=0 sub=0
psub=0 multi=-1 qbuf=0 qbuf-free=2147483647 obl=0 oll=0 omem=0 events=r
cmd=client
id=20 addr=10.0.3.1:33128 fd=66 name= age=9 idle=7 flags=N db=0 sub=0 psub=0
multi=-1 qbuf=0 qbuf-free=2147483647 obl=0 oll=0 omem=0 events=r cmd=hgetall
```

Advanced API introduction

- Quick review on additional API
 - Bitmaps
 - Bit Fields
 - HyperLogLogs
 - Geospatial Indexes
 - Transactions
 - Pub Sub
 - Lua Scripts
 - Modules

Bitmaps

- Bitmaps are a set of bit-oriented operations defined on the String type
 - Are not an actual data type
- Since strings are binary safe blobs and their maximum length is 512 MB
- Bits are set and retrieved using
 - [SETBIT](#) and [GETBIT](#) commands

```
> SETBIT key 10 1
(integer) 1
> GETBIT key 10
(integer) 1
> GETBIT key 11
(integer) 0
```

Bitmaps BITOP command

- The [BITOP](#) command supports four bitwise operations: **AND**, **OR**, **XOR** and **NOT**, thus the valid forms to call the command are:

```
>BITOP AND destkey srckey1 srckey2 srckey3 ... srckeyN  
>BITOP OR destkey srckey1 srckey2 srckey3 ... srckeyN  
>BITOP XOR destkey srckey1 srckey2 srckey3 ... srckeyN  
>BITOP NOT destkey srckey
```

BITFIELD

- BITFIELD treats a Redis string as a array of bits
 - Is capable of addressing specific integer fields of varying bit widths and arbitrary non (necessary) aligned offset
 - Are not an actual data type.
- Similarly the command handles increments and decrements of the specified integers

```
> BITFIELD mykey incrby i5 100 1 get u4 0  
1) (integer) 1  
2) (integer) 0
```

- More on BITFIELD Operations <http://redis.io/commands/BITFIELD>

HyperLogLog – Estimating the cardinality of a set

- A probabilistic data structure used in order to count unique things
 - Use an algorithm that trade memory for precision
 - Not actually store values
- HLLs in Redis is encoded as a Redis `string`
- Commands
 - [PFADD](#) : add an element to the count with [PFADD](#)
 - [PFCOUNT](#) : Retrieve the current approximation of the unique elements added with [PFADD](#) so far

```
> PFADD hll a b c d
(integer) 1
> PFCOUNT hll
(integer) 4
```


Geospatial Data and Indexes

- Redis data structures include geo specific commands allowing for extremely simple, low latency and efficient processing of location data in-memory
- Related commands
 - [GEOADD](#)
 - [GEODIST](#)
 - [GEOHASH](#)
 - [GEOPOS](#)
 - [GEORADIUS](#)
 - [GEORADIUSBYMEMBER](#)



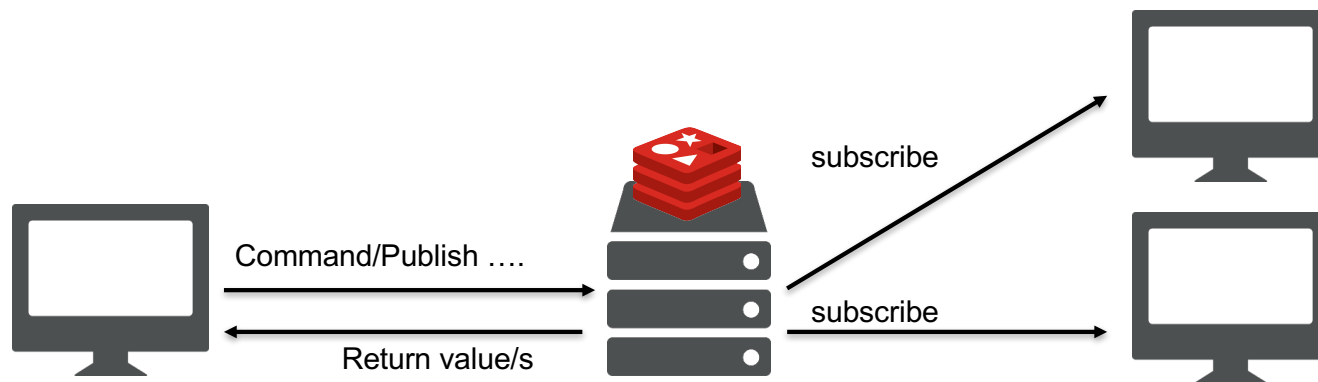
Transactions

- A Redis transaction is entered using the [MULTI](#) command
 - At this point the user can issue multiple commands
 - Instead of executing these commands, Redis will queue them
 - All the commands are executed once [EXEC](#) is called
 - commands are executed as a single isolated operation
 - Calling [DISCARD](#) instead will flush the transaction queue and will exit the transaction

```
> MULTI
OK
> INCR foo
QUEUED
> INCR bar
QUEUED
> EXEC
1) (integer) 1
2) (integer) 1
```

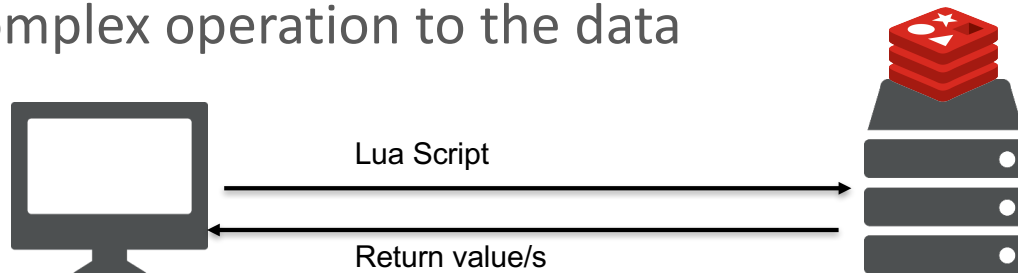
Publish Subscribe

- Redis is also a Pub-Sub server
 - Example
 - Much like a topic subscription
 - Keyspace notifications allows clients to subscribe to Pub/Sub channels and receive events affecting the Redis data set in some way



Lua Scripts

- It is possible to run [Lua scripts server side](#) to win latency and bandwidth
- [EVAL](#) and [EVALSHA](#) are used to evaluate scripts using the Lua interpreter
- Can speed up data processing
 - Less roundtrips are required to complete a complex task
 - Bring the complex operation to the data



Modules

- [Redis Modules](#) are add-ons to Redis which extend Redis to cover most of the popular use cases for any industry
- It is possible to extend Redis functionality using Redis Modules
 - Seamlessly plug into open source Redis or enterprise-class Redis
- Modules can be created by anyone
- The Module Hub marketplace includes Modules created by Redis Labs as well as by others
 - Examples: Image processing, Full text search, Secure password DB
 - For a list of available modules go to [Redis Modules](#)

Agenda

- Redis Data Store
- Strings
- Lists
- Hashes
- Sets
- Sorted Set
- Advanced API introduction
- **Redis CLI additional Features**
- Lab

Additional redis-cli Modes

1. Monitoring tool to show continuous stats about a Redis server
2. Scanning a Redis database for very large keys
3. Key space scanner with pattern matching
4. Acting as a Pub/Sub client to subscribe to channels
5. Monitoring the commands executed into a Redis instance
6. Checking the latency of a Redis server in different ways
7. Checking the scheduler latency of the local computer
8. Transferring RDB backups from a remote Redis server locally
9. Acting as a Redis slave for showing what a slave receives
10. Simulating LRU workloads for showing stats about keys hits
11. A client for the Lua debugger

Continuous stats mode

```
$ redis-cli --stat
----- data ----- load ----- child -
keys      mem      clients blocked requests      connections
506        1015.00K 1         0         24 (+0)         7
506        1015.00K 1         0         25 (+1)         7
506        3.40M    51        0         60461 (+60436)   57
506        3.40M    51        0         146425 (+85964)  107
507        3.40M    51        0         233844 (+87419)  157
507        3.40M    51        0         321715 (+87871)  207
508        3.40M    51        0         408642 (+86927)  257
508        3.40M    51        0         497038 (+88396)  257
```

- A new line is printed every second with useful information and the difference between the old data point
- Use `-i <interval>` to control the frequency

Additional useful API

- Monitoring commands executed in Redis

```
$ redis-cli monitor  
OK  
1460100081.165665 [0 127.0.0.1:51706] "set" "foo" "bar"  
1460100083.053365 [0 127.0.0.1:51707] "get" "foo"
```

- For more info on more usages and API
<http://redis.io/topics/rediscli>

Agenda

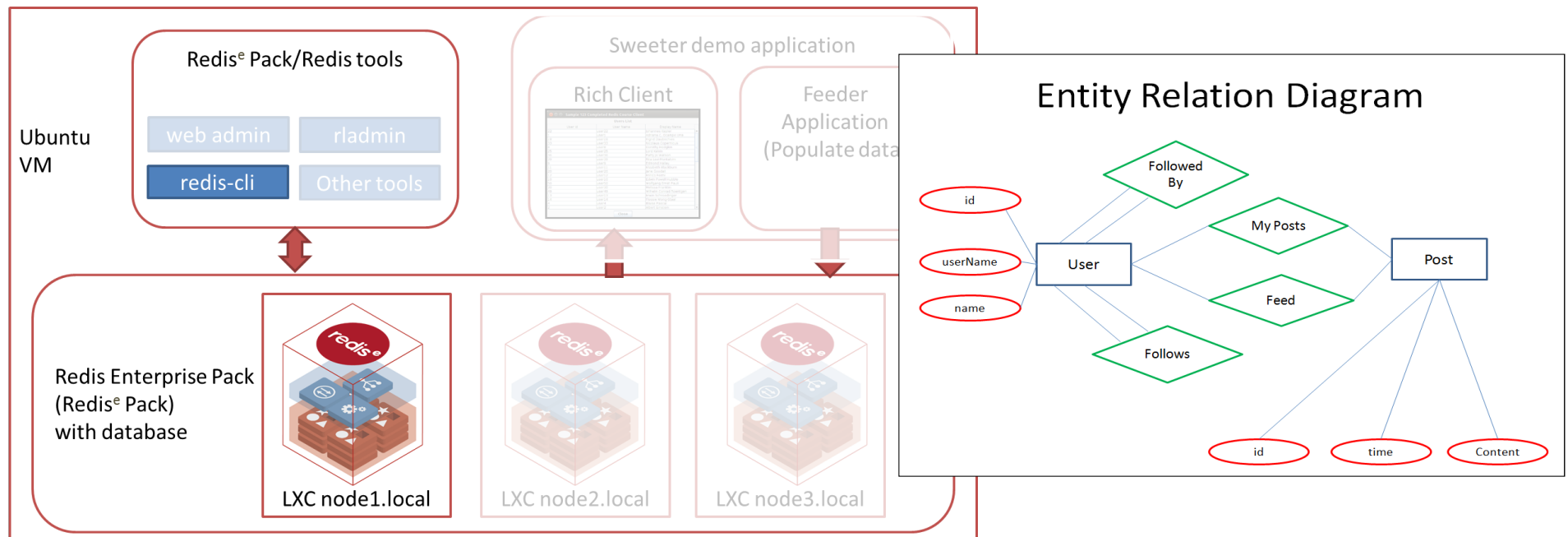
- Strings
- Lists
- Hashes
- Sets
- ZSets (Sorted Set)
- Bitmaps
- HyperLogLogs
- Advanced API introduction
- Redis CLI additional Features
- **Lab**

Lab 4

- Practice `redis-cli` common data types
 - Strings
 - Lists
 - Hashes
 - Sets
 - Sorted Sets



Lab 4 cont'



Q&A
