

16 APRILE 2021



VIRTUAL EVENT

**REDIS ON AZURE  
A NEW RISING  
STAR WITH V.6?**



**RICCARDO ZAMANA**

**@ZAMANARICCARDO**

# SPONSOR



## **managed/designs**

*il partner tecnologico per chi ha idee ambiziose. Innovazione pratica da 15 anni.*



*empower every person and every organization on the planet to achieve more.*

**#GLOBALAZURE**



# Swag and more

- Claim your attendee Learner Badge here:
- 30 Days to learn it: [aka.ms/global-azure/30D2L](https://aka.ms/global-azure/30D2L)
- Virtual background and ANOTHER Badge: [blog.globalazure.net/Swag](https://blog.globalazure.net/Swag)



# AGENDA

AZURE CACHE REDIS ON AZURE: A NEW RISING STAR WITH V.6?



- REDIS FUNDAMENTALS (3m)
- REDIS ON AZURE (7m)
- REDIS ANATOMY (15m)
- REDIS ADVANCED (8m)
- DEEP DIVE ON NEW AZURE MODULES (7m)

16 APRILE 2021



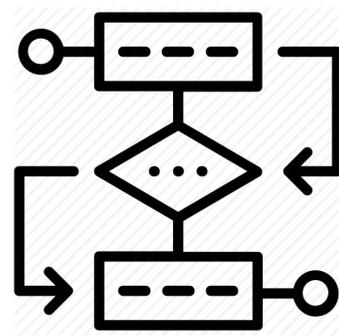
VIRTUAL EVENT

REDIS  
FUNDAMENTALS

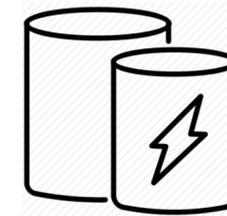
#GLOBALAZURE

# WHAT IS REDIS

## REMOTE DICTIONARY SERVICE



Redis is an open source (BSD licensed), in-memory data structure store



K, V cache



Message broker



database

- Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams. But can be expanded too.

# KEY FEATURES



## DEVELOPER POV

- TRANSACTIONS
- PIPELINING
- PUB/SUB
- LUA SCRIPTING
- MANAGED TTL EVICTION OF KEYS (LRU, ECC)
- <KEY, DS> ... NOT <KEY,V>

## ARCHITECT POV

- ONE CONFIG FILE
- DIFFERENT LEVELS OF ON-DISK PERSISTENCE
- BUILT-IN REPLICATION
- AUTOMATIC PARTITIONING WITH REDIS CLUSTER
- AUTOMATIC FAILOVER
- HIGH AVAILABILITY VIA REDIS SENTINEL

# IT'S VERY FAST

- It works with an in-memory dataset.
- You can persist your data either
  - by periodically dumping the dataset to disk
  - by appending each command to a disk-based log.



In Memory, so It's Faster  
> 1.5 million ops/second  
<1 ms latency

In Memory, so It's Faster  
> 80k ops/second  
With a B2S on Azure

It also supports asynchronous replication with:  
very fast non-blocking first synchronization  
auto-reconnection with partial resynchronization on net split

**#GLOBALAZURE**

# REDIS IS USEFUL FOR

- Realtime analytics
- High speed data ingest
- High Speed transactions
- Message queues
- Session Storage
- In-app social functionality
- Application job management
- Machine learning
- Search

... and caching.



## Transactional (OLTP) use cases:

- Financial transactions
- Pricing Management
- Advertising bids
- User profile Management
- Location-based processing
- User session management

## ANALYTICS (OLAP) use cases:

- Counting
- Leaderboards
- Page ranking
- Recommendation engine
- Time-series Analysis
- Session Analysis

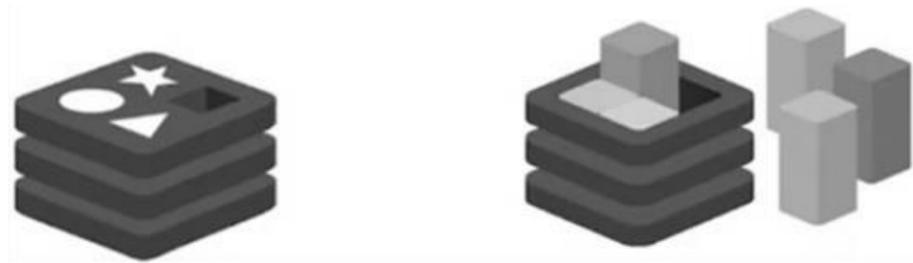
#GLOBALAZURE

# REDIS MODULES



REDIS is Modular.

You can update commands and datatypes.



Redis to cover most popular use cases in a wide variety of industries.

## Modules

- are add-ons to Redis that extend Redis
- are Redis v4.0+ compatible
- seamlessly plug into Redis
- can be created by anyone

## Two set of modules

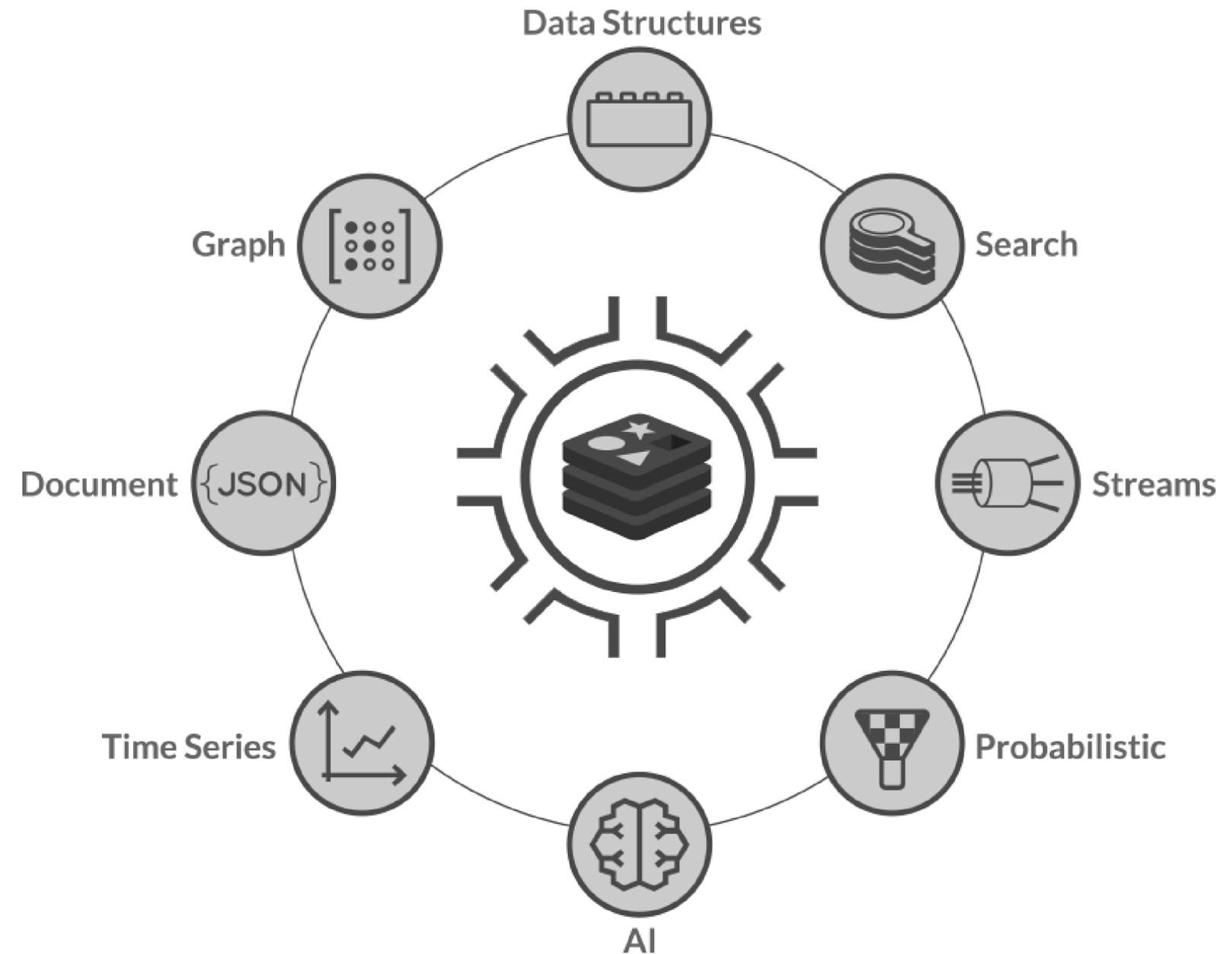
- modules under an OSI approved license
- modules that are under some proprietary license

- Develop your own Module:
- <https://github.com/RedisLabs/RedisModulesSDK>
- YOU can build your own modules.
- It can embed compute complexity.

# MODERN DATA MODELS



- **Redisearch (as Elasticsearch)**
  - [redisearch.io](https://redisearch.io)
- **RedisTimeSeries (as Influx)**
  - [redistimeseries.io](https://redistimeseries.io)
- **RedisJson (come MongoDB)**
  - [rejson.io](https://rejson.io)
- **RedisGraph (come Neo4j)**
  - [redisgraph.io](https://redisgraph.io)
- **RedisBloom (probabilistic filter)**
  - [rebloom.io](https://rebloom.io)
- **RedisAI (Execute tensor or pytorch)**
  - [redisai.io](https://redisai.io)



# I DIDN'T LIKE IT BEFORE BECAUSE..



## YESTERDAY

- Many OSS fragile GUI
- Windows version was not updated
- Management (HA, Backups, etc) was not well documented
- It was «a cache» for me, not the fulcrum neither the end of «data's story».

## NOW

- Redis is WSL Compatible
- Official client, rather than redis cli: **Redis Insight**
- Redis is a certifiable knowledge: **Redis University**
- ... and YES, its a cache **ALSO**.

*..AND IT RUNS ON IOT EDGE  
with 5Mb LOAD 😊*

# I WAS A BIT CONFUSED BECAUSE..



## YESTERDAY

- *Azure version was 4*
- *Windows version was 5*
- *Version 6 was STABLE on REDIS.IO*

## TODAY

- **WSL is my friend**
- **6.0.3 is available on Azure PaaS**
- **AKS and ACI can do all the rest**

My conclusion, in many IoT contexts is:

$$\text{Sum}(\text{IoT} \Rightarrow \text{Cost}(\text{Services}(i,n))) > \text{Cost}(\text{Redis.Premium})$$

16 APRILE 2021



VIRTUAL EVENT

REDIS  
ON AZURE

#GLOBALAZURE

# WHAT IS AZURE CACHE FOR REDIS



- Managed VM based deployment
- Offers both the Redis open-source (OSS Redis) and a commercial product from Redis Labs (Redis Enterprise) as a managed service
- It provides secure and dedicated Redis server instances and full Redis API compatibility
- In case of Redis Labs managed service, there are two fees

## From Zero to Hero !!!!

C0 Basic	F700 Enterprise Flash
250 MB Cache	715 GB Cache
Low network bandwidth	Highest network bandwidth
Shared infrastructure	All Standard features
SSL	Redis Cluster
Up to 256 connections	Up to 50,000 connections
13.80 EUR/MONTH (ESTIMATED)	13,625.91 EUR/MONTH (ESTIMATED) PER 3 C...

# TIERS AND MONTHLY COSTS



Level	Redis Version	SLA	Deployment	Notes	Starts from
Basic	OSS	No SLA	1 VM		13 €
Standard	OSS	99.9%	2 VM replicated		33 €
Premium	OSS	99.9%		Data persistence, net security, Passive geo Replica, up to 10 shard Cluster	340 €
Enterprise	RedisLabs	Up to 99.999%		HA, Redis Modules, Active Geo replica	820 €
Ent. flash	RedisLabs	Up to 99.999%		Large Datasets, Cache on Disk	6.700 €

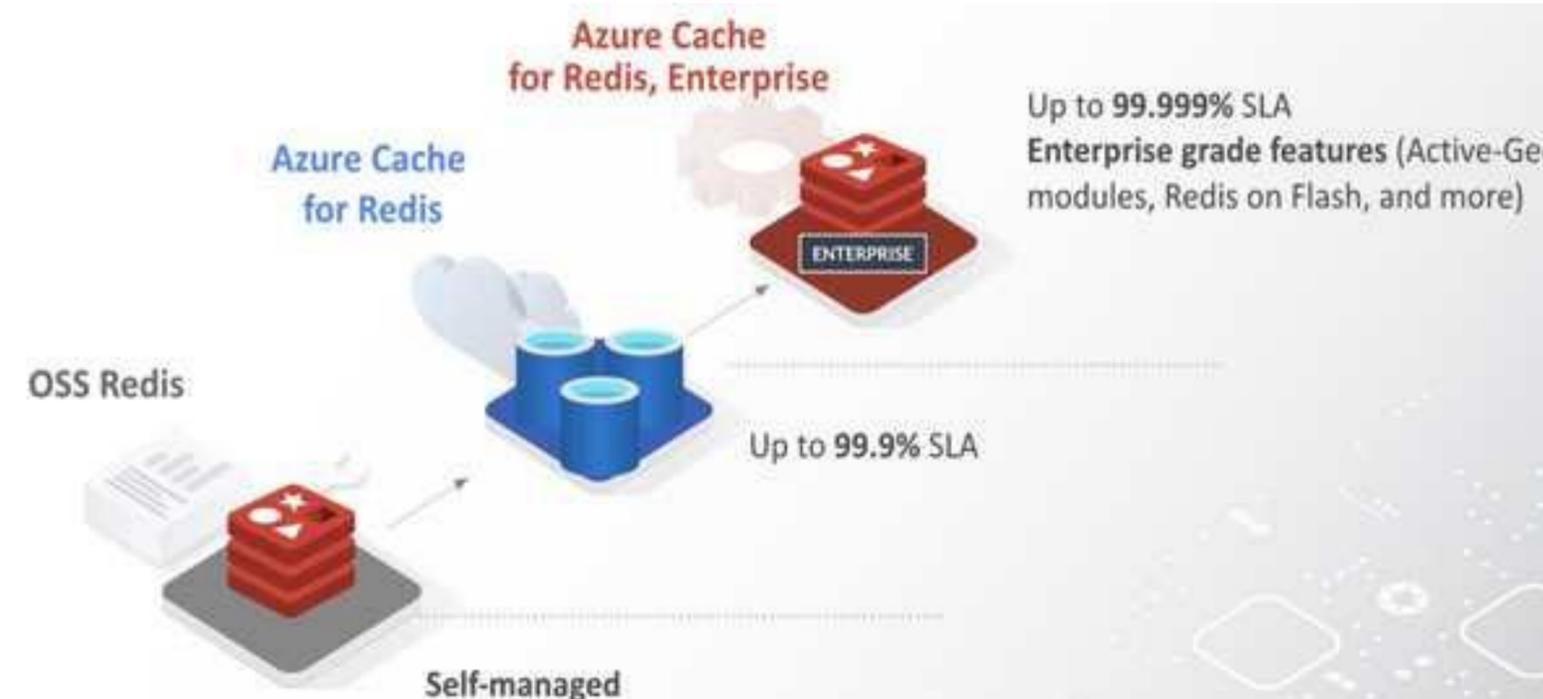
## My 2 cents:

- From Standard to Premium ... there is AKS
- Enterprise Flash: is not applicable to any market landscape

# WHAT IS ENTERPRISE OFFERING



- Active Geo-Replication (in Preview)
- Higher availability—architected for 99.99% with zone redundancy, and 99.999% with active geo-replication.
- Redis on Flash, enabling ten times larger cache sizes on NVMe flash storage.
- Integrated billing and the ability to use Azure spend commitment.
- Supported Redis Enterprise Modules
  - **Redis Search**
  - **Redis Bloom**
  - **Redis TimeSeries**
- Scaling
  - Datasets up to 13TB
  - Up to 2,000,000 concurrent client connections
  - More than 1,000,000 ops/sec



# INCREASED AVAILABILITY

## REMEMBER THE NUMBERS

- 99.9% => **Monthly: 43m 49s**
- 99.99% => **Monthly: 4m 22s**
- 99.999% => **Monthly: 26s**



When do you need .999?

Retail: world Inventory

Finance: Pricing systems

99.9%

**Default**

Built-in replication with automatic failover to replica node

Up to  
99.99%

**Multi-AZ**

Placing replicas in different availability zones

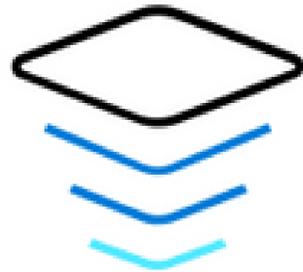
Up to  
99.999%

**Active Geo-Replication**

Adding multi-primary writes across regions

**#GLOBALAZURE**

# REDIS MODULES + REDIS 6.0



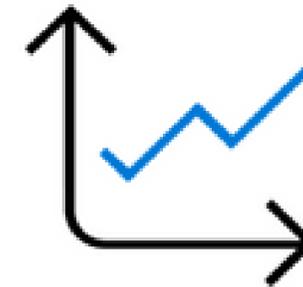
## RedisBloom

Probabilistic data structure to check for item membership in a highly memory-efficient way



## RedisSearch

Real-time indexing, querying, and search engine for Redis.



## RedisTimeSeries

Time series data structure to ingest and query millions of samples and events

# PERSISTENCE MODES



- AOF (append-only file) data persistence: **logs every write operation or accumulates one second of write operations** with minimal to non-effect on Redis performance.

## Data Persistence

Data persistence allows you to persist data stored in Redis. You can also take snapshots and back up the data which you can load in case of a failure.

Backup file  Disabled  RDB  AOF

First Storage Account \* ⓘ

\*First Storage Key ⓘ

Second Storage Account ⓘ

Second Storage Key ⓘ

- Snapshot (RDB) data persistence: **performs point-in-time snapshots of your dataset at specified intervals** that can be used to rebuild your dataset if needed.

## Data Persistence

Data persistence allows you to persist data stored in Redis. You can also take snapshots and back up the data which you can load in case of a failure.

Backup file  Disabled  RDB  AOF

Backup Frequency \*

Storage Account \* ⓘ

\*Storage Key ⓘ

# AZURE BILLING



- The purchasing process is made seamless through integral billing.
- Customer with MACC agreement, will find Redis Enterprise spend in their Azure commitment

## New Redis Cache ...

Basics Networking Advanced Tags Review + create

### PRODUCT DETAILS

Redis Enterprise

by Redis Labs

[Terms of use](#) | [Privacy policy](#)

[Azure benefit eligible](#)

235.10 EUR/Month (Estimated)

[Pricing details for the plan](#)

Hosting infrastructure

by Microsoft

[Terms of use](#) | [Privacy policy](#)

Subscription credits apply

603.57 EUR/Month (Estimated)

### TERMS

By selecting the checkbox and clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the [Azure Marketplace Terms](#) for additional details.

# PERFORMANCE - NETWORK



- Network bandwidth and latency usually have a direct impact on the performance.
- **Use the ping program** to quickly check the latency between the client and server hosts
- **Estimate the throughput** in Gbit/s and compare it to the theoretical **bandwidth** of the network.
- Redis throughput is limited by the network well before being limited by the CPU.

**Bandwidth and Network latency is the KEY. MUST BE CALCULATED!!!**

**#GLOBALAZURE**

# PERFORMANCE - CPU AND RAM



- CPU is another very important factor. Being single-threaded, **Redis favors fast CPUs with large caches** and not many cores.
- **Speed of RAM and memory** bandwidth seem less critical for global performance especially for small objects. **For large objects (>10 KB)**, it may become noticeable though.

# PERFORMANCE – SERVICE TIER



- **Different key size:** Number may change if you use different KEY sizes or VALUE sizes
- **Throughput:** Throughput for the caches that are the same size is higher in the Premium tier as compared to the Standard tier
- **Shards:** With Redis clustering, throughput **increases linearly** as you increase the number of shards (nodes) in the cluster
- **Service tier:** Throughput for bigger key sizes is higher in the Premium tier as compared to the Standard Tier

# MY 2 CENTS

CLOUD SPENDING DURING DEV / TEST / PROD



1. Start small (but COMPLETE) using the **Magic «K»**
2. Evolve to Enterprise (with balanced offering)
3. Scale Globally without Headache (with AAS offering)

# THREE SIMPLE RECIPES TO START



- **ONLY BUILT IN DATA STRUCTURE** => Azure Cache for Redis
- **ALL MODULE + FIXED SIZE:** ACI Redis node (with persistence)
- **ALL MODULES + DYNAMIC SIZE:** AKS Redis cluster

# AZURE CACHE REDIS



- Version 6 PREVIEW
- ACLs bring the concept of “users” to Redis
- encrypt traffic over SSL
- while it retains a core single-threaded data-access interface, I/O is now threaded
- client-side caching layer

## New Redis Cache

Basics   Networking   Advanced   Tags   Review + create

Non-TLS port  Enable

Redis version  4  (PREVIEW) 6

# REDIS BY ACI



```
az container create \  
  --resource-group myRG \  
  --name redisinstance \  
  --image redislabs/redismod \  
  --dns-name-label myFQDN \  
  --ports 6379 \  
  --azure-file-volume-account-name *** \  
  --azure-file-volume-account-key *** \  
  --azure-file-volume-share-name *** \  
  --azure-file-volume-mount-path /
```

- Persistence with FileShare
- Docker based run

# REDIS BY AKS



```
az login
```

```
az account set --subscription *****
```

```
az aks get-credentials --resource-group **RG** --name **N**
```

```
az aks proxy --resource-group **RG** --name **N**
```

```
az aks browse --resource-group **RG** --name **N**
```

```
az aks enable-addons --addons kube-dashboard --resource-group **RG** --name **N**
```

```
kubectl delete clusterrolebinding kubernetes-dashboard
```

```
kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard --user=clusterUser
```

```
az aks browse --resource-group **RG** --name **N**
```

**#GLOBALAZURE**

# REDIS BY AKS

## REPLICATED CLUSTER IN 4 SIMPLE STEPS !!



### Primary Service

```
apiVersion: v1
kind: Service
metadata:
  name: primary
spec:
  type: LoadBalancer
ports:
  - protocol: TCP
    port: 6379
  targetPort: 6379
  name: redis
  selector:
    name: redis-primary
```

### Replica Service

```
apiVersion: v1
kind: Service
metadata:
  name: replica
spec:
  type: LoadBalancer
ports:
  - protocol: TCP
    port: 6379
  targetPort: 6379
  name: redis
  selector:
    name: redis-replica
```

### Primary Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: primary-deployment
labels:
  name: redis-primary
spec:
  replicas: 1
  selector:
    matchLabels:
      name: redis-primary
  template:
    metadata:
      labels:
        name: redis-primary
    spec:
      subdomain: primary
      containers:
        - name: redis
          image: redis:3.2.0-alpine
          command:
            - "redis-server"
          args:
            - "--protected-mode"
            - "no"
      ports:
        - containerPort: 6379
```

### Replica Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: replica-deployment
labels:
  name: redis-replica
spec:
  replicas: 2
  selector:
    matchLabels:
      name: redis-replica
  template:
    metadata:
      labels:
        name: redis-replica
    spec:
      subdomain: replica
      containers:
        - name: redis
          image: redis:3.2.0-alpine
          command:
            - "redis-server"
          args:
            - "--slaveof"
            - "primary.default.svc.cluster.local"
            - "6379"
            - "--protected-mode"
            - "no"
      ports:
        - containerPort: 6379
```

AND CHECK!

**rscli -h X.X.X.X -p 6379 info replication**

**#GLOBALAZURE**

16 APRILE 2021



VIRTUAL EVENT

REDIS  
ANATOMY

#GLOBALAZURE

# HOW REDIS MUST BE USED



- Keys are **strings**, and must be **unique** within a redis instance.
- By convention, redis keys are separated by colon (:)
  - “users:123” is a key representing a user with id = 123
  - “users:123:favourites” is a key representing the favourites of user with id = 123 Keys always have an associated value.
- If the value is empty or null, redis automatically deletes the key.

# THREE SIMPLE OPERATIONS

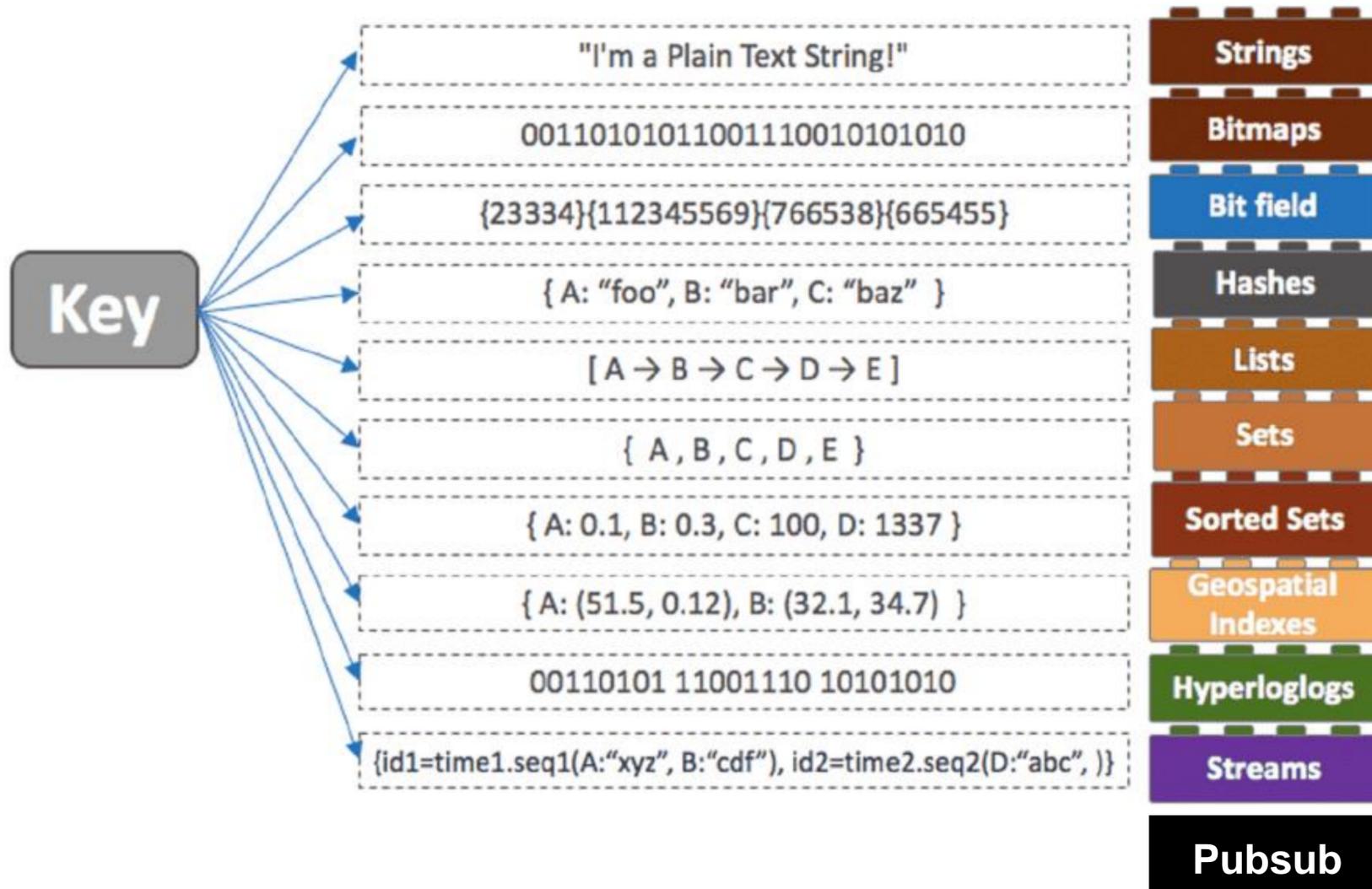
## YOU MUST MANAGE MEMORY



- **Expiry:** You can set an expiry on any key, regardless of the type. Once a key expires, Redis automatically deletes it from memory.
- **Existence:** You can check if a key exists in memory
- **Type:** You can check the type of a key - i.e. whether it's a string, set or list etc.

# DATA STRUCTURES

## DEFAULT DATA STRUCTURE



Redis has 5 basic data types:

Strings , Hashes , Lists , Sets , Sorted Sets

In addition, it the following advanced data structures:

- **Bitmaps:** built on string
- **Bitfield:** built on string
- **HyperLogLog:** built on string
- **Geospatial:** built on sortedset

# DATA TYPES: HASH

UP TO  $2^{32}$  KEY VALUE PAIR IN EACH HASH

- Fields in hash can be **incremented atomically (counters)**

Sample usage:

- Saving properties of a B.O. (plain table)
- Saving Sessions
- Maintain Many-to-many mappings



userid	8754
name	dave
ip	10:20:104:31
hits	1
lastpage	home

Hashes can store key=value pairs.

Hashes cannot be further nested, the value must be a binary string.

If you have a persisted class with Dictionary/HashMaps, you would typically store that as a Hash in Redis

# DATA TYPES: HASH

## EXAMPLES



- **User Profiles:** Many web applications use Redis Hashes for their user profiles, as they can use a **single hash for all the user fields**, such as name, surname, email, password, etc.
- **User Posts:** Social platforms like Instagram leverage Redis Hashes to **map all the archived user photos or posts back to a single user**. The hashing mechanism allows them to look up and return values very quickly, fit the data in memory, and leverage data persistence in the event one of their servers dies.
- **Storing Multi-Tenant Metrics:** Multi-tenant applications can leverage Redis hashes to **record and store their product and sales metrics** in a way that guarantees solid separation between each tenant, as hashes can be encoded efficiently in a very small memory space.

# DATA TYPES: LIST

UP TO  $2^{32}$  ELEMENTS IN EACH KEY (4 BILLION)

- Internally maintained as Linked Lists safe
- $O(N)$ : extremely fast near both ends
- Ideal for:
  - Queues
  - Stacks
  - Top N (Recent News)
- Sample Uses:
  - Social network Timeline (LPUSH to add, LRANGE to retrieve recent items)
  - LPUSH+LTRIM to keep top N
  - User action / error logging
  - RPOPLPUSH using same source/dest to rotate items in a ring



<u>bus:messages</u>	1	4	{age:3}	1
---------------------	---	---	---------	---

# DATA TYPES: LIST

## EXAMPLES



- **Social Networking Sites:** Social platforms like Twitter use Redis Lists to populate **their timelines or homepage feeds**, and can customize the top of their feeds with trending tweets or stories.
- **RSS Feeds:** Create news feeds from custom sources where you can pull the **latest updates** and allow interested followers to subscribe to your RSS feed.
- **Leaderboards:** Forums like Reddit and other voting platforms leverage Redis Lists to add articles to the leaderboard and sort by **most voted entries**.

# DATA TYPES: SET

UP TO  $2^{32}$  ELEMENTS IN EACH KEY (4 BILLION)

- Maintains unique set of items (unordered)
- Support intersection/union
- Quickly Check if a member exists

Ideal for:

- Storing relations (K=UserID, V = set of Followers)

Sample use:

- Tagging
- Extract random members



page:home:uniquevisitors:20140225	80.55.203.40	201.3.14.185	55.67.100.120	40.77.232.204
-----------------------------------	--------------	--------------	---------------	---------------

# DATA TYPES: SET

## EXAMPLES



- **Analyzing Ecommerce Sales:** Many online stores use Redis Sets to **analyze** customer behavior, such as **searches or purchases for a specific product category or subcategory**.
- **IP Address Tracking:** Redis Sets are a great tool for developers who want to **analyze all of the IP addresses** that visited a specific website page or blog post, and to be able to **ignore all of the duplicates** for unique visitors with their SADD function.

# DATA TYPES: STRING

UP TO 2<sup>32</sup> BITS ( 512 MB) IN EACH KEY,



## Ideal for:

- Plain strings (page/post cache)
- Full JSON Objects
- Raw bits/flags (realtime metrics- daily active users)
- Binary file content (binary file manipulation)

## Sample Uses:

- Lock (SET resourcename anyvalue NX EX max-lock-time)
- General cache
- Picture masking (BITOP)
- Stats for A/B testing

users:1	{name:'John',email:'a@b.com' }
images:5	#h ChhpLT5 OJ8« z<÷H_ ·*„

They are the simplest and most complicated data structure.

First, strings are binary.

Which means you can store text like “hello world”, or numbers like 42, or a floating point number like 3.14

# DATA TYPES: STRING

## EXAMPLES



- **Session Cache:** Many websites leverage Redis Strings to create a session cache to speed up their website experience by caching HTML fragments or pages. Since data is stored temporarily in the RAM, this attribute makes Redis a perfect choice as a session cache. It is able to temporarily store user-specific data, for instance, items stored in a shopping cart in an online store, which is crucial in that your users do not lose their data in the event they log out or lose connection.
- **Usage & Metered Billing:** A lesser known use case for Redis Strings is **the real-time metering for consumption-based pricing models**. This allows SaaS platforms that bill based on actual usage to meter their customers activity, such as in the telecommunications industry where they may charge for text messages or minutes.

# DATA TYPES: SORTED SET

UP TO  $2^{32}$  ELEMENTS IN EACH KEY (4 BILLION)



- Maintains unique set of items (ordered)
- Order is defined by SCORE
- Same features of SET

Leaders	<u>sam</u>	<u>neville</u>	<u>devon</u>	<u>matt</u>
	4236	5870	7615	7615

## Sorted Set = Hash + List

- Hash because you can store a “score” with every element
- List because you can retrieve a subset of elements in a sorted order

Sample use:

- Maintain a leader board
- Any TOP something
- Index other REDIS data (member is USERID, score is AGE. Query USER with given age Range)

# DATA TYPES: SORTED SET

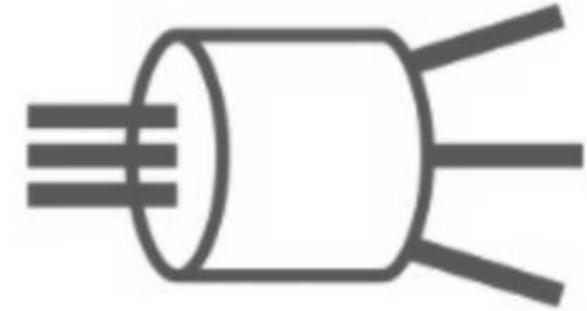
## EXAMPLES



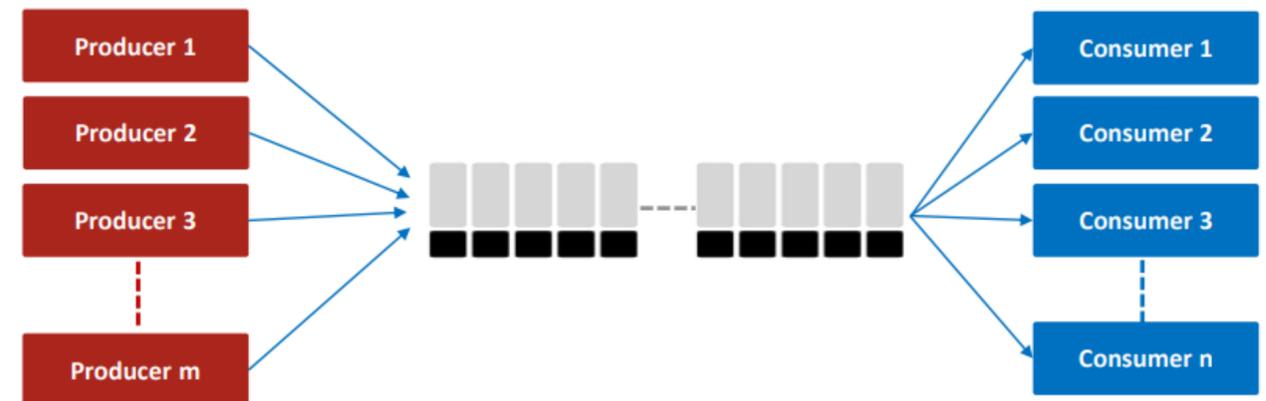
- **Q&A Platforms:** Many Q&A platforms like Stack Overflow and Quora use Redis **Sorted Sets to rank the highest voted answers** for each proposed question to ensure the best quality content is listed at the top of the page.
- **Gaming App Scoreboards:** Online gaming apps leverage Redis Sorted Sets to **maintain their high score lists**, as scores can be repeated, but the strings which contain the unique user details cannot.
- **Task Scheduling Service:** Redis Sorted Sets are a great tool for a task scheduling service, as you can **associate a score to rank the priority of a task in your queue**. For any task that does not have a score noted, you can use the WEIGHTS option to a default of 1.
- **Geo Hashing:** The Redis geo indexing API uses a Sorted Set for the Geo Hash technique which allows you to **index locations based on latitude and longitude**, turning multi dimensional data into linear data.

# REDIS STREAMS

A STEP AHEAD IN «IN-ORDER INGESTION»



- Available with Redis 5.0+
- **BUILT TO SOLVE STREAMING PROBLEMS**
  - **Sorted set:** memory hungry, client's can't block awaiting new records
  - **List:** linear time to scan, no fan-out
  - **Pub/Sub:** no history, no windowed aggregation



Streams enable asynchronous data exchange between producers and consumers and historical range queries

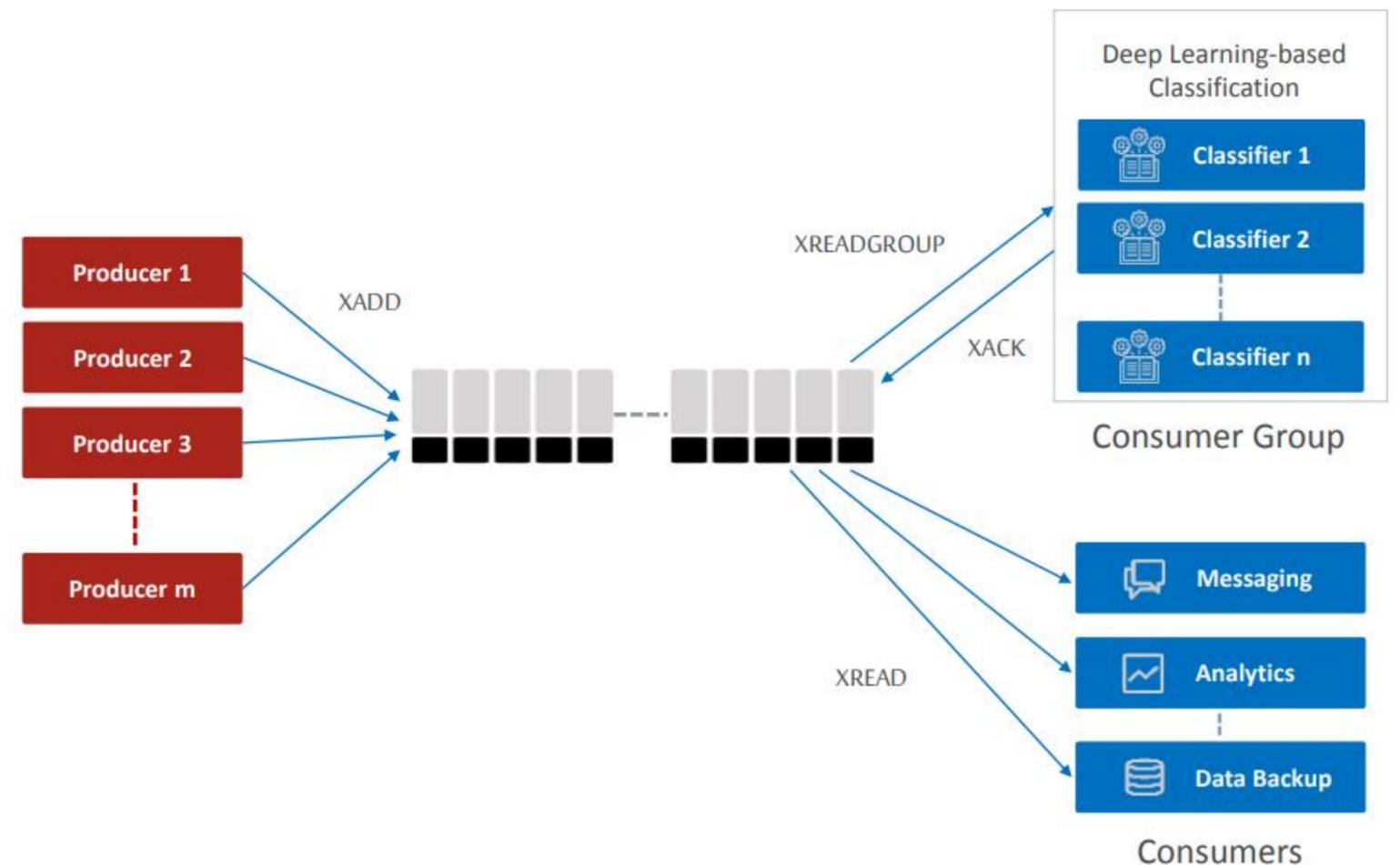
**GOAL:** handle asynchronous communication between producers and consumers.

# REDIS STREAM

## FEATURES & EXAMPLES

### FEATURES:

- Push & Pop like lists
- Support **lookup by ID**
- Multiple producers and **consumers** can interact with the **same queue**
- Auto generated ID, no duplicates problem, pair field/value per sample

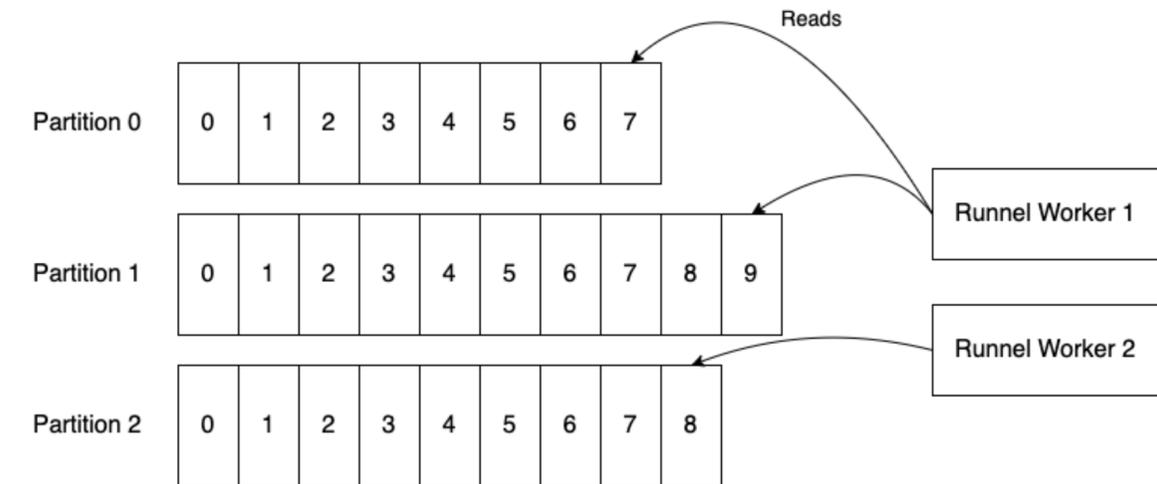


# REDIS STREAM

## IS A NEW EVENTHUB/KAFKA COMPETITOR?



- 1. Event partitioning:** you must create N STREAMS and a HASH to decide Partition
- 2. A worker-partition assignment system:** you need to create an algorithm to distribute partitions amongst workers, ensuring "rebalancing"
- 3. In-order processing with acknowledgement.** Each worker needs to iterate over each of its partitions, keeping track of its offsets.
- 4. Error handling:** forward errors to a "dead-letter" stream and continue processing.



My main motivation for working on this problem was the ease-of-use and low cost involved in deploying and operating Redis. That's why it's attractive vs Kafka.

# HYPERLOGLOG

## ESTIMATE YOUR DATA

- HyperLogLog can tell you an estimate of the number of unique items it's been supplied.
- HyperLogLog is space efficient — the maximum size of the data structure is about 12kb.
- HyperLogLog is pretty quick.
- HyperLogLog has a low error rate — 0.81%.
- HyperLogLog is stored, internally, as a normal string in Redis.



15 users

User	Hashed Bitstring
sean	01111101
todd	11010001
aaron	10000111
kat	01110001
don	01011010
sara	01000001
linda	01010011
eric	<b>0000</b> 1001
jack	01101001
steph	10001100
terry	00111110
tim	00010000
wanda	11110001
chris	01101110
jane	00010010

Because the longest leading sequence of zeros is 4 bits long, we can say that there may be approximately 16 users

→ Approximate Count =  $2^4 = 16$

Example: Imagine to list process and count unique Ids, with 8.000.000 Ids (2 bytes each) = 16Gb of Load

Hyperloglog constant size: 1.5 kB

# HYPERLOGLOG

## USE CASES



### SIMPLE:

- The unique search terms or unique IPs that connected today to your web site
- The number of unique guests to your batchroom area
- The number of unique tourist in a city during a weekend
- The number of unique cars that travel in a city during a weekend
- The number of unique devices that send data

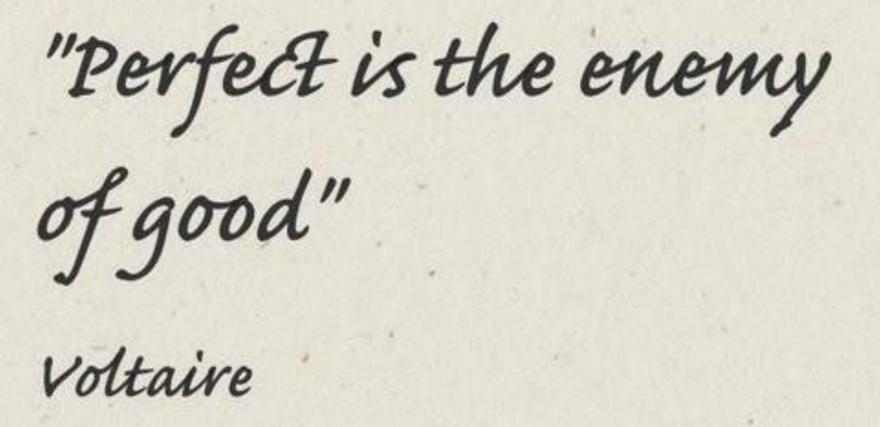
### LESS SIMPLE:

- all the unique elements encountered so far in order to match the next element with the set of already seen elements
- increment a counter only if the new element was never seen before.
- All unique people around you in a day
- Make a formal verification of industrial systems telemetry threshold

# HYPERLOGLOG

## DATA SKETCHES – WHAT ARE THEY?

- Sketch is a data structure that can represent extremely large sets with constant space complexity.
- That means that you don't need to scale up your compute when you change from your first thousand, to your first million, to your first billion transactions!



## Hyperloglog - a *probabilistic* method

*General Idea:* Count **leading zeros** in a randomly generated binary number

Given a random number,  
what is the probability of seeing...?

1 x x x x x x x... → 0.5 (1 out of every 2)  
0 1 x x x x x x... → 0.25 (1 out of every 4)  
0 0 1 x x x x x... → 0.125 (1 out of every 8)  
...  
0 0 0 0 0 1 x x... → 0.008 (1 out of every 128)  
...

**Question:**

I have a list of  $N$  unique numbers.  
The one with the **longest string**  
of leading zeros is

0 0 0 0 0 0 1 x x...

What is  $N$ ?

16 APRILE 2021



VIRTUAL EVENT

REDIS  
ADVANCED

#GLOBALAZURE

# PIPELINING

## ONE PAGER EVALUATION



### Definition

Sending multiple commands to the server in the same message, separating commands by newline.

The server buffers all the answers in memory and sends all at once when the pipeline is done.

### Use pipeline when

- you need performance
- you have several commands to send to the Redis server
- you don't need the response of a previous command as input for a subsequent command (because you only get all responses in the end).

### PROS

- It saves round trip amount
- It avoids context switching

### CONS

- It gives the response after executing all the pipeline.
- If some command fails, the pipeline continues and an error is returned at the end

**NO ROLLBACK!**

# ATOMICITY OF PIPELINING

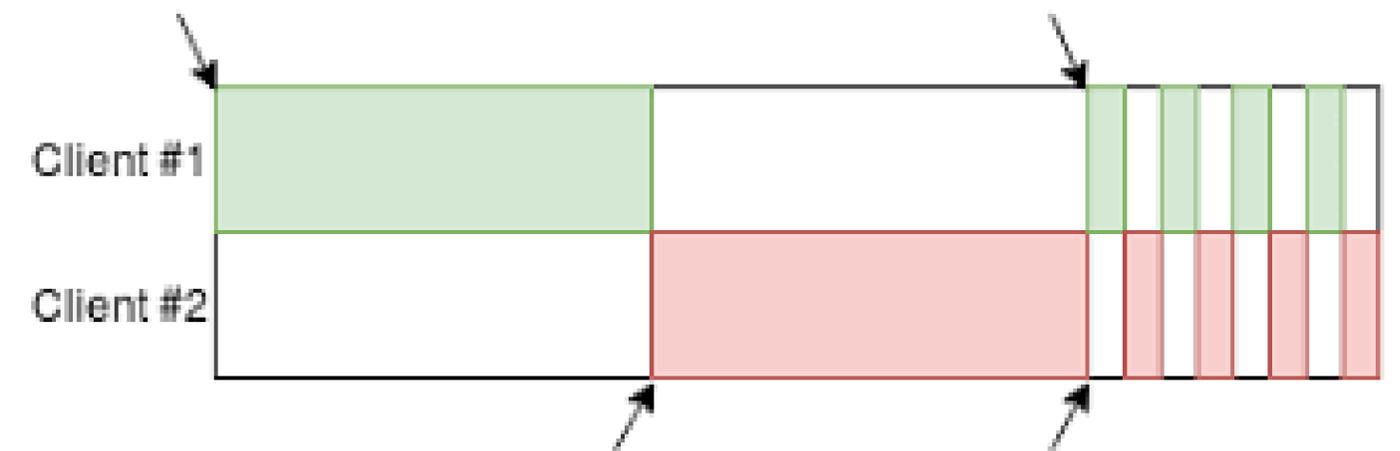


The agent that executes client commands is single-threaded. All commands in Redis are atomic, executed individually.

## HIGHLIGHTS

- Execution in order not guaranteed
- Pipeline is not atomic

## Pipelines interleaving Phenomenon



pipelining is non-blocking on the server

# TRANSACTION

## ONE PAGER EVALUTION



### Definition

Mechanism for queuing commands, and later deciding whether we want to executed all of them (atomically) or give up and don't execute any command.

### Use transaction when

- you need atomic execution of commands
- you don't need intermediate values to compose subsequent commands

### PROS

- If pipelining is used on the library, all the performance benefits from using pipeline will apply.
- Transactions also provide a "check-and-set" with the WATCH command.

### CONS

Like pipelining, using transactions we don't have the ability to use intermediate values for subsequent commands. We only get the whole list of responses at the end.

If an error Happens... IT DEPENDS: Semantic or Syntax error?

**NO ROLLBACK, AGAIN!**

**#GLOBALAZURE**

# ATOMICITY OF TRANSACTIONS



- Transactions are atomic. **There is no interleaving of commands** inside a transaction with commands from outside that transaction.
- **Transactions are blocking**, in front of other clients.

## Scenario

1. before the transaction, we read a value from key my-key and store it in our program
2. we start the transaction with MULTI
3. we queue commands inside the transaction, using the value read from key my-key
4. we EXEC the transaction

```
WATCH sampleKey  
num = GET sampleKey  
num = num + 1  
MULTI  
SET sampleKey $num  
EXEC
```

# LUA SCRIPTS

## ONE PAGER EVALUTION



### Definition

Redis can execute client-provided scripts written in Lua. A Lua script is loaded on the Redis server and can be later invoked with parameters. You can also send the whole script on every invocation, but you should avoid doing this for performance reasons.

### Use transaction when

- you need atomic execution of commands
- you need intermediate values to compose subsequent commands
- you need intermediate values to conditionally execute commands

### PROS

- In a Lua script we can manipulate intermediate results.

### CONS

- While there is no limitations in terms of functionality of Lua scripts, it might be weird to use it to implement a huge pipeline with atomicity.

# ATOMICITY OF LUA SCRIPTS



- Lua scripts are atomic. Likewise transactions, they are blocking and can make other clients wait for a long time if the script is slow.

```
local key = KEYS[1]
local new = ARGV[1]

local current = redis.call('GET', key)
if (current == false) or (tonumber(new) <
    tonumber(current)) then
    redis.call('SET', key, new)
    return 1
else
    return 0
end
```

# MY 2 CENTS



- Roundtrip amount matters
- If you don't care of exceptions: PIPELINE are ok, TRANSACTION are ok.
- If you want a «Atomic Multi-Command» : use Transactions
- If you want to perform, managing WHAT IF inside code: Use LUA

Don't bore me with talk about using multiple programming languages.

Remember SQL STORE PROCEDURE / PL-SQL.

# DATA EXPIRATION & EVICTION POLICIES



- Data structures in Redis can be marked with a Time To Live (TTL) set in seconds, after which they will be removed.
- A series of configurable intelligent “eviction policies” are available.
- Optional TTL permits the creation of a tiered hierarchy of memory objects
- In some use cases a least recently used (LRU) or least frequently used (LFU) metric makes more sense for eviction.

## CACHE POLICIES:

- Least recently used (LRU)
- Time aware least recently used (TLRU)
- Most recently used (MRU)
- Pseudo-LRU (PLRU)
- Random replacement (RR)
- Segmented LRU (SLRU)
- Least-frequently used (LFU)
- Least frequent recently used (LFRU)
- LFU with dynamic aging (LFUDA)
- Low inter-reference recency set (LIRS)

# KEY SPACE NOTIFICATIONS

## WHAT HAPPENS ON AZURE?



Keyspace notifications allow clients to subscribe to two channels in order to receive events affecting the Redis data set in some way. Examples:

- All the commands affecting a given key.
- All the keys receiving a specific operation (Set, Delete, etc)
- All the keys expiring in the database.

Two distinct types of events for every operation affecting the Redis data space:

- **Key-space** notification: to monitor specific keys
- **Key-event** notification: to monitor specific events

# KEYSPACE NOTIFICATIONS

## WHAT HAPPENS ON AZURE?



Two types of events can be subscribed using PSUBSCRIBE command, in this format:

```
psubscribe __keyspace@<database>__:<mykey>
```

```
psubscribe __keyevent@<database>__:<operation>
```

```
Welcome to secure redis console!

This console connects to your live redis server and all commands are run on the server.

WARNING: Use expensive commands with caution as they can impact your server load!

>set key1 value1
OK
>
```

```
Connection closed: your next command will be run on a new connection.
>psubscribe '__key*__:*'
Reading messages... (press ENTER to quit)
1) "psubscribe"
2) "__key*__:*"
3) (integer) 1

1) "pmessage"
2) "__key*__:*"
3) "__keyspace@0__:key1"
4) "set"

1) "pmessage"
2) "__key*__:*"
3) "__keyevent@0__:set"
4) "key1"
```

# KEY SPACE NOTIFICATIONS

## WHAT HAPPENS ON AZURE?



A screenshot of the Azure portal's "Advanced settings" page for an Azure Cache for Redis instance named "redis236". The left sidebar shows navigation options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main content area shows various settings: "Allow access only via SSL" is set to "Yes"; "Non-SSL Port" is "Disabled"; "SSL Port" is "6380"; "Minimum TLS version" is "1.1"; "maxmemory-policy" is "volatile-lru"; "maxmemory-reserved" and "maxfragmentationmemory-reserved" are shown as sliders; and "notify-keyspace-events" is set to "AKE". A warning message states "TLS 1.0 and 1.1 support is being retired soon, which may impact your application".

- K Keyspace events, published with `__keyspace@<db>__` prefix.
- E Keyevent events, published with `__keyevent@<db>__` prefix.
- g Generic commands (non-type specific) like DEL, EXPIRE, RENAME, ...
- \$ String commands
- l List commands
- s Set commands
- h Hash commands
- z Sorted set commands
- t Stream commands
- x Expired events (events generated every time a key expires)
- e Evicted events (events generated when a key is evicted for maxmemory)
- m Key miss events (events generated when a key that doesn't exist is accessed)**
- A Alias for "g\$lshztxe", so that the "AKE" string means all the events except "m".

The string AKE can be used to enable every possible event.

16 APRILE 2021



VIRTUAL EVENT

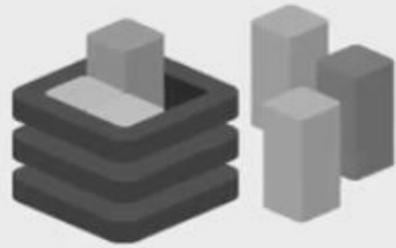
DEEP DIVE  
ON REDIS V.6 AZURE MODULES

#GLOBALAZURE

# REDIS TIME SERIES (IN 2 MINUTES)



RedisTimeSeries implements an a time series database as a module in Redis



- WHAT IS RDTS: time + value = SAMPLE
- Analysis with time-bounded range
- Separate results into time units
- Example: average measure per hour



# PRIOR TO REDIS TIMESERIES

THERE WERE SORTED SETS!



- SORTED SET: MEMBER + SCORE (Time as Member, Score as Result)
- ZADD mysortedset 123456789ts 1000
- **THE PROBLEM WAS... DUPLICATE SCORE during TIME !!!!**
- If you put TWO identical values with 2 different timestamps ... it updates the timestamp.
- Trick: Always append a random to the score to make string unique
- ZADD mysortedset 123456789ts 1000:[random]

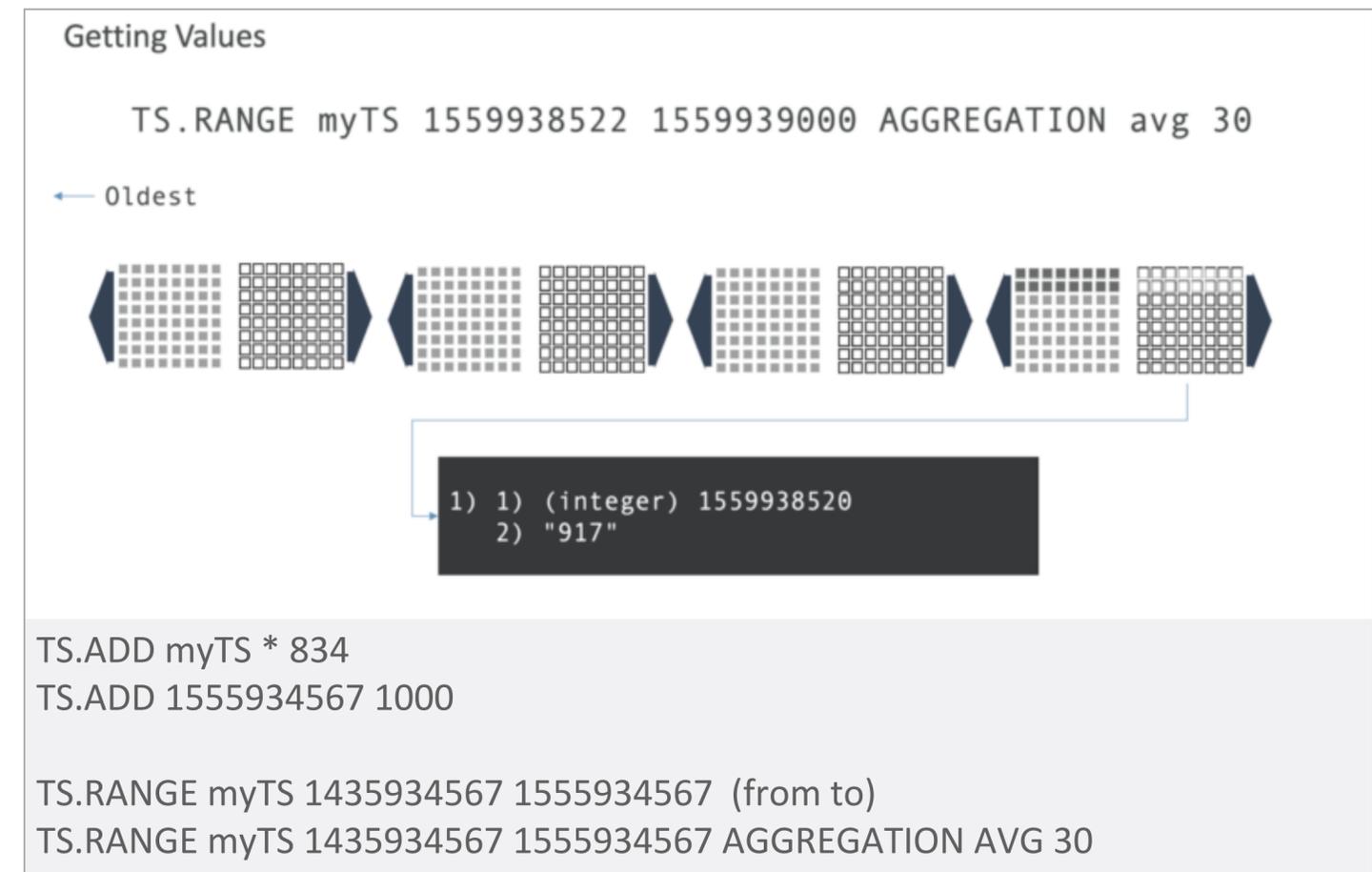
#GLOBALAZURE

# REDIS TIMESERIES INTERNALS



## IT'S A LINKED LIST

- Based on CHUNKS
- 128bit per sample array 64bit (64 timestamp array + 64 value array)
- It's a Linked list
- TS CREATE (creates the linked list).



- **Every series can have labels** (es: plant: plant-1, sensor: temp-4)
- **TTL in seconds is permitted** (TS.CREATE myTS LABELS plant plant-1 sensor temp-4 RETENTION 60)

# REDIS TIMESERIES MAIN FEATURES

Downsampling.. EVERYWHERE!



- PERFORM AGGREGATIONS:

- MIN, MAX, COUNT, LAST, SUM, AVG

- **PERFORM REALTIME DOWNSAMPLING :**

- TS.CREATE RULE myTS myTS2  
AGGREGATION AVG 30

- (every 30 seconds, it creates new sample)

OTHER COMMANDS:

- TS.INCRBY / DECRBY (variation history)
- TS.GET it takes most recent
- TS.ALTER change label metadata
- TS.MRANGE & TS.MGET to query more series

# REDIS BLOOM (IN 2 MINUTES)

## A MODULE FOR DATA SKETCHES



### KEY ASPECTS:

- LOW LATENCY
- PROBABILISTIC DATA STRUCTURE
- CONSTANT MEMORY ALLOCATION
- LOW ERROR RATE

The RedisBloom module provides four data structures:

- **Bloom filter:** A data structure designed to rapidly determine if an element is present in a set in a highly memory-efficient manner.
- **Cuckoo filter:** An alternative to Bloom filters with additional support for deletion of elements from a set.
- **Count-Min Sketch:** Calculates frequency of events in data samples.
- **Top-K:** A deterministic algorithm that approximates frequencies for the top k items.



# REDIS BLOOM



- must be created with a foreknowledge of how many entries they contain.
- The *bpe* number needs to be fixed, and likewise, the width of the bit array is also fixed.
- They may be stacked to accommodate more elements
- When creating a Bloom filter - even a scalable one, it's important to have a good idea of how many items it is expected to contain.

BF.ADD bloom kirk

BF.EXISTS bloom kirk

BF.MADD bloom elem1 elem2 elem3

BF.MEXISTS bloom elem1 elem2 elem3

# REDIS SEARCH (IN 2 MINUTES)



- **Unified Search and analytics:** Query data, content, and documents using a variety of indexing, querying, and full-text search capabilities.
- **Horizontally scalable:** Scale out and partition indexes over several shards and nodes for greater speed and memory capacity.
- **Enterprise-grade reliability:** Enjoy continued operations in any scenario with five-nines availability and Active-Active failover.

# REDIS SEARCH 2.0



## Indexing and Querying

- create indexes on Hashes
- incremental indexing approach (more rapid)
- perform aggregations, filter by properties, numeric ranges

## Full-Text and Fuzzy Search

- full-text indexing and **stemming-based query expansion in multiple languages**
- It provides a rich **query language** that can perform **text searches**
- It can **implement auto-complete suggestions** using 'fuzzy' searches

# REDIS SEARCH 2.0

## A QUICK EXAMPLE



Create Index with 4 secondary indexes (title, release, rating, genre)

```
FT.CREATE idx:movie ON hash PREFIX 1 "movie:" SCHEMA title TEXT SORTABLE release_year
NUMERIC SORTABLE rating NUMERIC SORTABLE genre TAG SORTABLE
```

### Full-text search queries

```
FT.SEARCH idx:movie "war" RETURN 3 title release_year rating
```

- 1) (integer) 1
- 2) "movie:11002"
- 3) 1) "title"
  - 2) "Star Wars: Episode V - The Empire Strikes Back"
  - 3) "release\_year"
  - 4) "1980"
  - 5) "rating"
  - 6) "8.7"

### Search in specific fields:

```
> FT.SEARCH idx:movie "@title:war" RETURN 3 title release_year rating
```

### Prefix matches

```
FT.SEARCH idx:movie "emp*" RETURN 3 title release_year rating
```

### Fuzzy search

```
> FT.SEARCH idx:movie "%gdfather%" RETURN 3 title release_year rating
```

### Union

```
> FT.SEARCH idx:movie "war | %gdfather%" RETURN 3 title release_year rating
```

16 APRILE 2021



VIRTUAL EVENT

Q&A

#GLOBALAZURE

16 APRILE 2021



VIRTUAL EVENT



# Riccardo Zamana

*IoT and Telemetry passionate  
Proud Azure Fellow*



**THANK YOU**