Redis for beginners

Introduction

Caching

Caching - Web Development

Caching is an intermediate storage technique for application data, which can be done through hardware or software. It serves to provide faster access to certain information than directly accessing the application's database.

While a database is focused on managing information, the cache is aimed at reading this data, not worrying about other operations, as this task is already being assigned.

With that we have faster and more direct access, gaining in time and performance of access to this data.

Caching is an important application feature, as it gains speed and performance in obtaining data. However, there is a big “it depends” on this solution because not all applications necessarily really need the use of caching. Before applying such a concept, we must always analyze the application and the real need for the use of caching.

What is Redis?

What is Redis and What Does It Do?

Redis, which stands for Remote Dictionary Server, is a fast, open-source, in-memory, key-value data store. The project started when Salvatore Sanfilippo, the original developer of Redis, wanted to improve the scalability of his Italian startup. From there, he developed Redis, which is now used as a database, cache, message broker, and queue.

Redis delivers sub-millisecond response times, enabling millions of requests per second for real-time applications in industries like gaming, ad tech, financial services, healthcare, and IoT. Today, Redis is one of the most popular open-source engines today, named the "Most Loved" database by Stack Overflow for five consecutive years. Because of its fast performance, Redis is a popular choice for caching, session management, gaming, leaderboards, real-time analytics, geospatial, ride-hailing, chat/messaging, media streaming, and pub/sub-apps.

What Redis is not

Just as it is very important to understand what Redis is, it is essential to understand also what Redis is not and what you cannot accomplish with it.
Among the things that Redis is not or does not are:
• It is not a relational database like MySQL or Oracle;
• It is not a document-oriented database like MongoDB;
• It's not a database you should be using to store all your data;
• Does not have official support for Windows;
• Does not use the HTTP protocol.

Redis Architectures

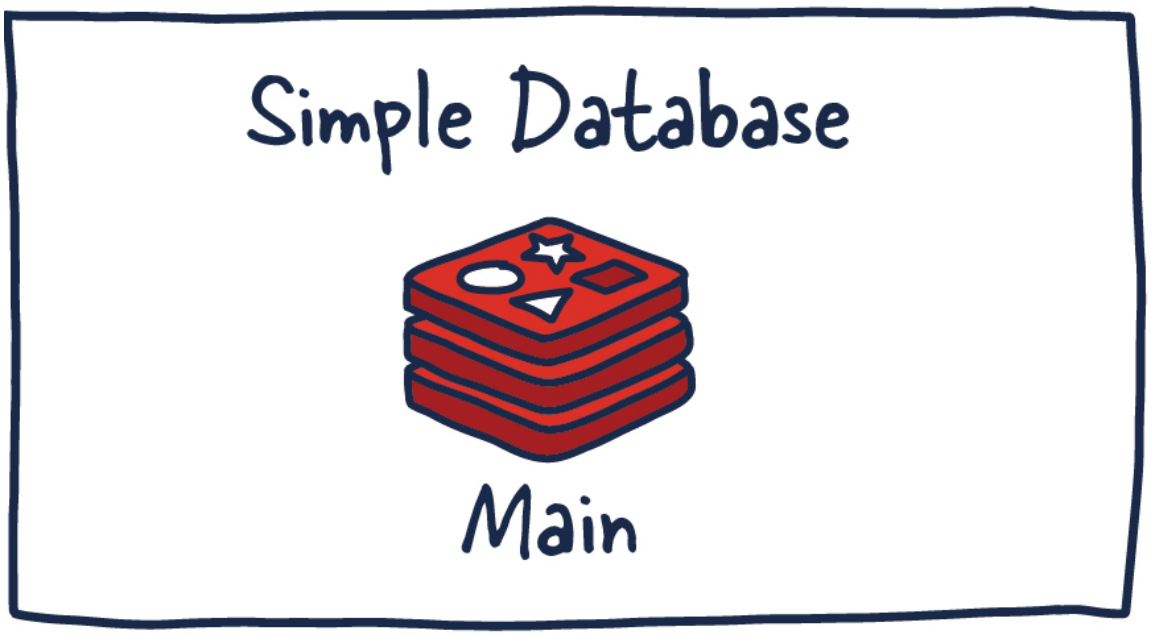
Before we start the hands-on, let's discuss the various Redis deployments and their trade-offs.

We will be focusing mainly on these configurations:

- Single Redis Instance
- Redis High Availability
- Redis Sentinel
- Redis Cluster

Depending on your use case and scale, you can decide to use one setup or another.

Single Redis Instance



Single Redis instance is the most straightforward deployment of Redis. It allows users to set up and run small instances that can help them grow and speed up their services. However, this deployment isn't without shortcomings. For example, if this instance fails or is unavailable, all client calls to Redis will fail and therefore degrade the system's overall performance and speed.

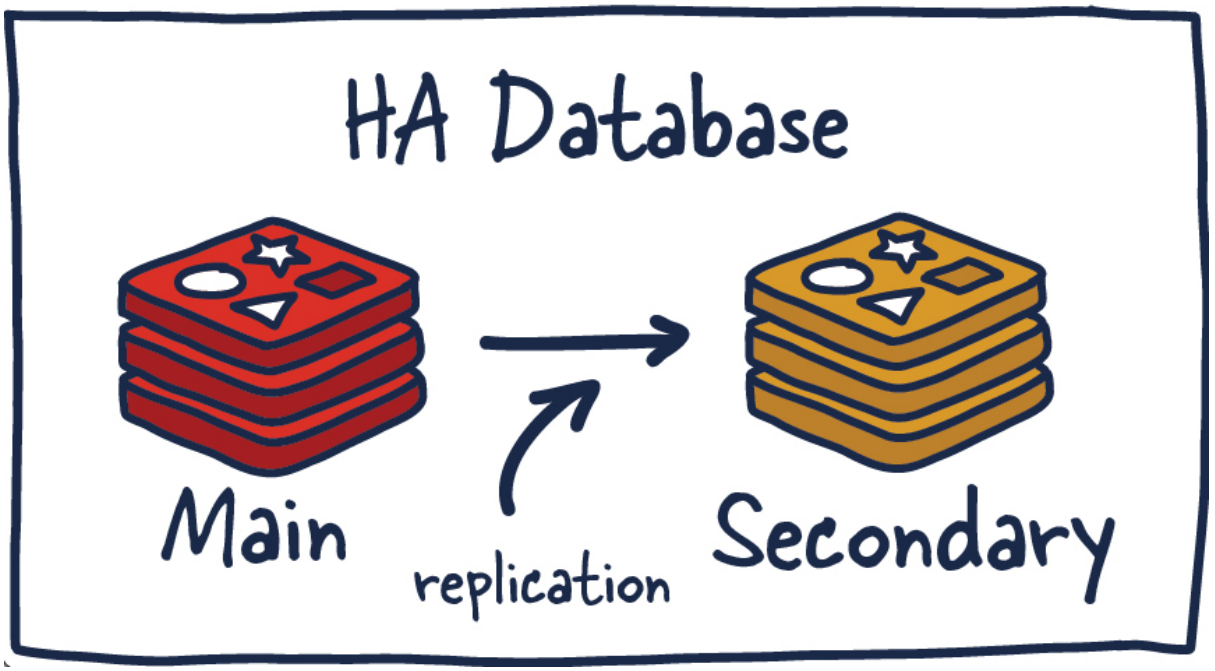
Given enough memory and server resources, this instance can be powerful. A scenario primarily used for caching could result in a significant performance boost with minimal setup. Given enough system resources, you could deploy this Redis service on the same box the application is running.

High Availability

High Availability with Redis Enterprise

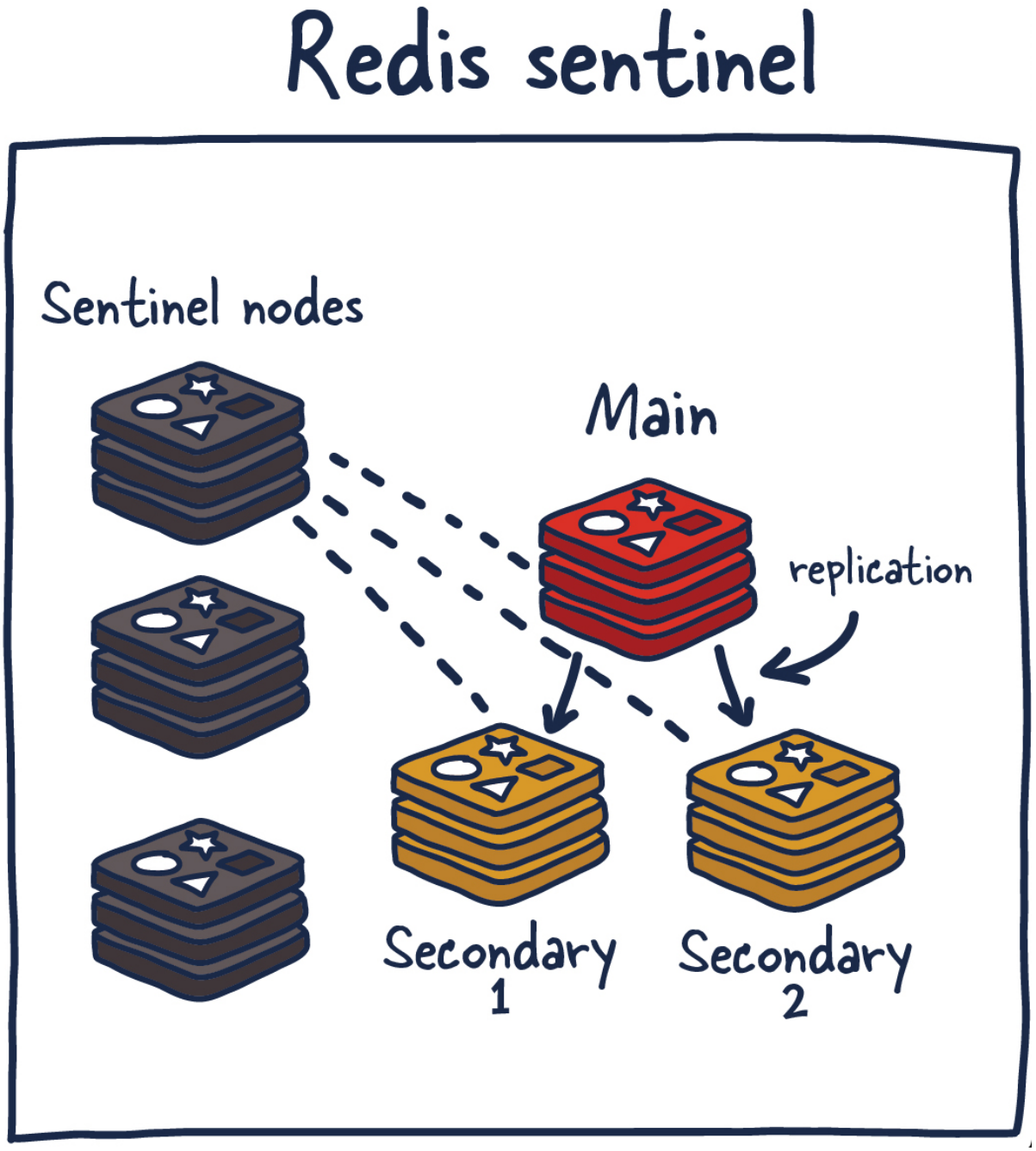
Another popular setup with Redis is the main deployment with a secondary deployment that is kept in sync with replication. As data is written to the main instance it sends copies of those commands, to a replica client output buffer for secondary instances which facilitates replication. The secondary instances can be one or more instances in your deployment. These instances can help scale reads from Redis or provide failover in case the main is lost.

High availability (HA) is a characteristic of a system that aims to ensure an agreed level of operational performance, usually uptime, for a higher-than-average period.



In these HA systems, it is essential to not have a single point of failure so systems can recover gracefully and quickly. This results in a reliable crossover, so data isn't lost during the transition from primary to secondary, in addition to automatically detecting failure and recovery from it.

Redis Sentinel



Sentinel is a distributed system. As with all distributed systems, Sentinel comes with several advantages and disadvantages. Sentinel is designed in a way where there is a cluster of sentinel processes working together to coordinate state to provide high availability for Redis. After all, you wouldn't want the system protecting you from failure to have its own single point of failure.

Sentinel is responsible for a few things. First, it ensures that the current main and secondary instances are functional and responding. This is necessary because sentinel (with other sentinel processes) can alert and act in situations where the main and/or secondary nodes are lost. Second, it serves a role in service discovery much like Zookeeper and Consul in other systems. So when a new client attempts to write something to Redis, Sentinel will tell the client what the current main instance is.

So sentinels are constantly monitoring availability and sending out that information to clients so they can react to them if they indeed do failover.

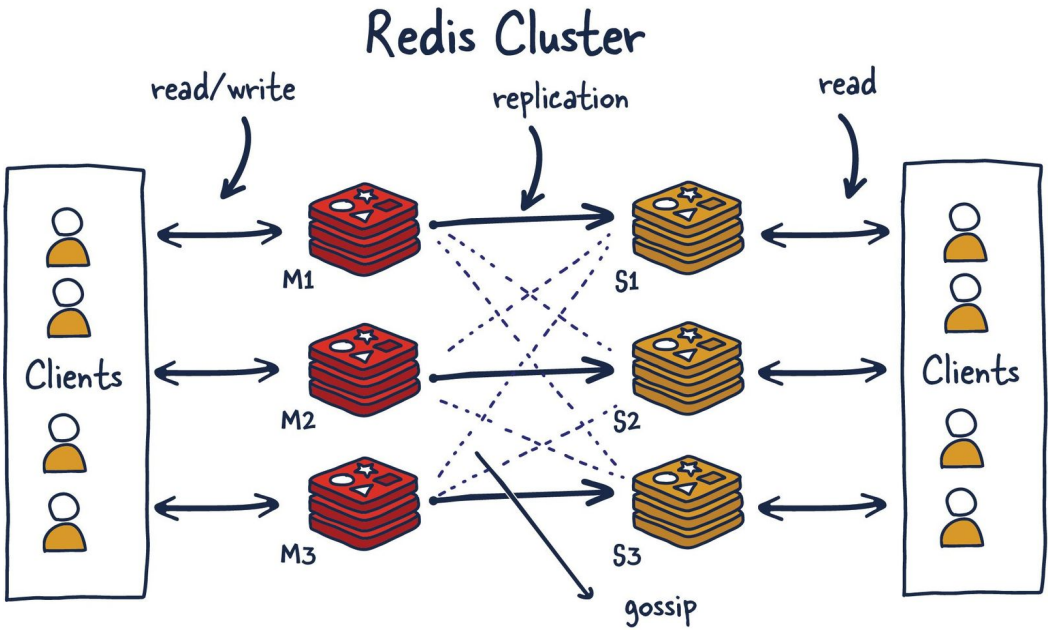
Here are its responsibilities:

1. **Monitoring:** ensuring main and secondary instances are working as expected.
2. **Notification:** notify system admins about occurrences in the Redis instances.
3. **Failover management:** Sentinel nodes can start a failover process if the primary instance isn't available and enough (quorum of) nodes agree that is true.
4. **Configuration management:** Sentinel nodes also serve as a point of discovery of the current main Redis instance.

Using Redis Sentinel in this way allows for failure detection. This detection involves multiple sentinel processes agreeing that the current main instance is no longer available. This agreement process is called Quorum. This allows for increased robustness and protection against one machine misbehaving and being unable to reach the main Redis node.

Redis Cluster

Clustering in Redis



Once you can't store all your data in memory on one machine Redis Cluster allows for the horizontal scaling of Redis which means spreading the workload across multiple smaller machines responsible for smaller parts of the whole.

This process is known as sharding, and each Redis instance in the cluster is considered a shard of the data as a whole. So, to know which Redis instance (shard) is holding that data there are several ways to do this, but Redis Cluster uses algorithmic sharding.

To find the shard for a given key, we hash the key and mod the total result by the number of shards. Then, using a deterministic hash function, meaning that a given key will always map to the same shard, we can reason about where a particular key will be when we read it in the future.

When we need to add a new shard into the system we start a process called re-sharding.

Assuming the key 'foo' was mapped to shard zero after introducing a new shard, it may map to shard five. However, moving data around to reflect the new shard mapping would be slow and unrealistic if we need to grow the system quickly. It also has adverse effects on the availability of the Redis Cluster.

Redis Cluster has devised a solution to this problem called Hashslot, to which all data is mapped. There are 16K hash slots. This gives us a reasonable way to spread data across the cluster, and when we add new shards, we simply move hash slots across the systems. By doing this, we just need to move hash slots from shard to shard and simplify the process of adding new primary instances into the cluster.

This is possible without any downtime and minimal performance hit.

Hands-on

Now, after we've learned a bit of Redis theory, let's get down to business.

Installation

You can compile and install Redis from source on a variety of platforms and operating systems including Linux and macOS.

In this course, we are going to install Redis using docker.

As the focus of the course is not docker, here is its documentation so you can install it on your machine: https://docs.docker.com/desktop/.

I will assume that at this point you already have your docker working so let's move on.

To do that, run the next command in your terminal:

```
docker run --name redis -p 6379:6379 -d redis
```

If you need to install Redis on another platform, nothing better than following the official documentation. Here is the link: https://redis.io/docs/getting-started/installation.

To see if our container is running, you can run:

```
docker ps
```

Then you will have something like this in your terminal:

```
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS                     NAMES
b267809a9bb3   redis     "docker-entrypoint.s..."   9 minutes ago   Up 9
minutes   0.0.0.0:6379->6379/tcp   redis
```

It means the container is running.

Some important commands at this point:

```
# To stop the redis container
docker stop redis

# To see all the containers, even those are not running
docker ps -a

# To start redis
docker start redis
```

Redis CLI

To run the Redis CLI, you can run the following command in your terminal.

```
 docker exec -it redis redis-cli
```

At this point, is supposed to open the prompt inside CLI with something like this:

```
127.0.0.1:6379>
```

To be sure if everything is ok, run the next command:

```
127.0.0.1:6379> info
```

Then, will be shown a lot of information in your terminal, like this fragment:

```
# Server
redis_version:7.0.11
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:f1ad580a4464b9ef
redis_mode:standalone
os:Linux 5.10.102.1-microsoft-standard-WSL2 x86_64
arch_bits:64
monotonic_clock:POSIX clock_gettime
multiplexing_api:epoll
atomicvar_api:c11-builtin
gcc_version:12.2.0
process_id:1
process_supervised:no
run_id:5413e8fbb3d14fd9e3218b1e47658d2df92a0299
tcp_port:6379
server_time_usec:1687682488068704
uptime_in_seconds:1345
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:9960888
executable:/data/redis-server
config_file:
io_threads_active:0
```

Redis GUI

There are numerous GUI tools for Redis, however, for this course, we will be using a tool called Another Redis Desktop Manager. As the site itself describes it, it is a faster, better, and more stable Redis desktop manager (GUI Client), compatible with Linux, Windows, and MacOS.

The official site of this tool is available at https://goanother.com/#download, and there you can find alternative ways to install it in different operational systems.

For this course, we will use the fastest way to install it on Windows, Linux, and MacOS.

For Windows, you can use the next command:

```
winget install qishibo.brew install --cask another-redis-desktop-manager
```

For Linux, run the next command:

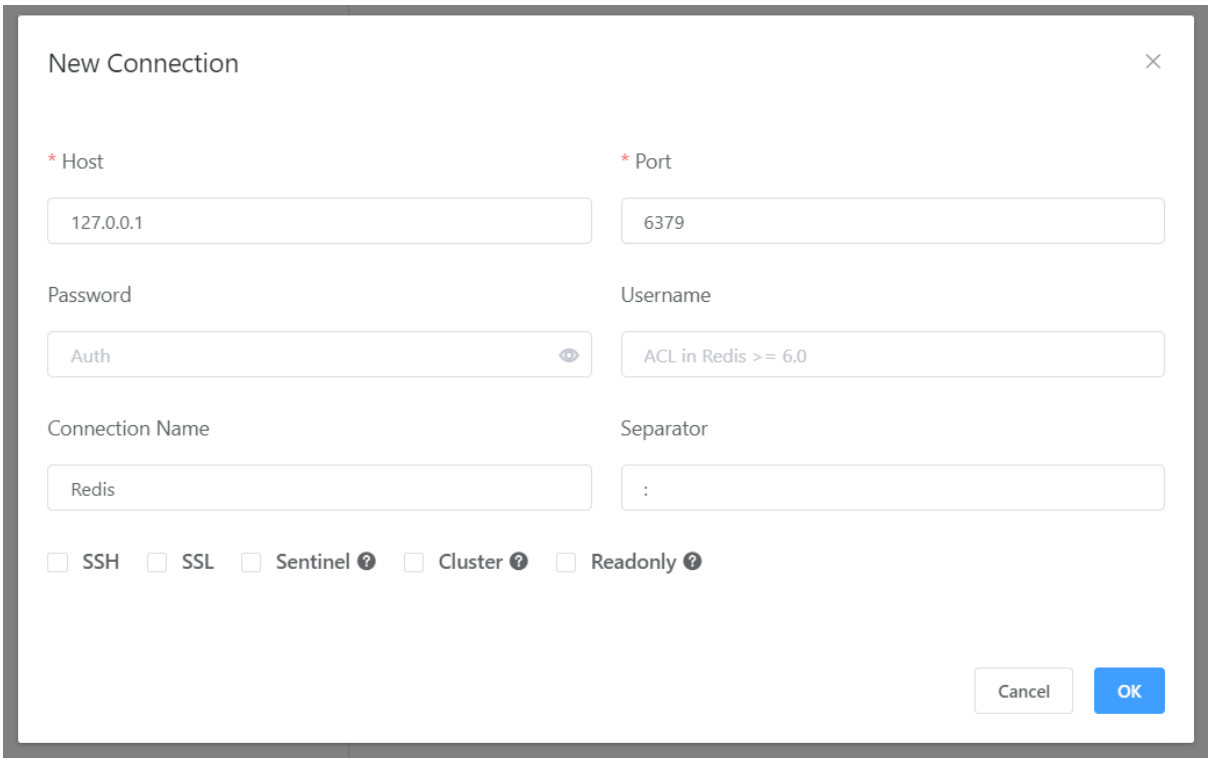
```
sudo snap install another-redis-desktop-manager
```

For MacOS run the command:

```
brew install --cask another-redis-desktop-manager
```

Once installed, run it and you will have something like this:

So, click on the “New Connection” button and fill in the fields like this:



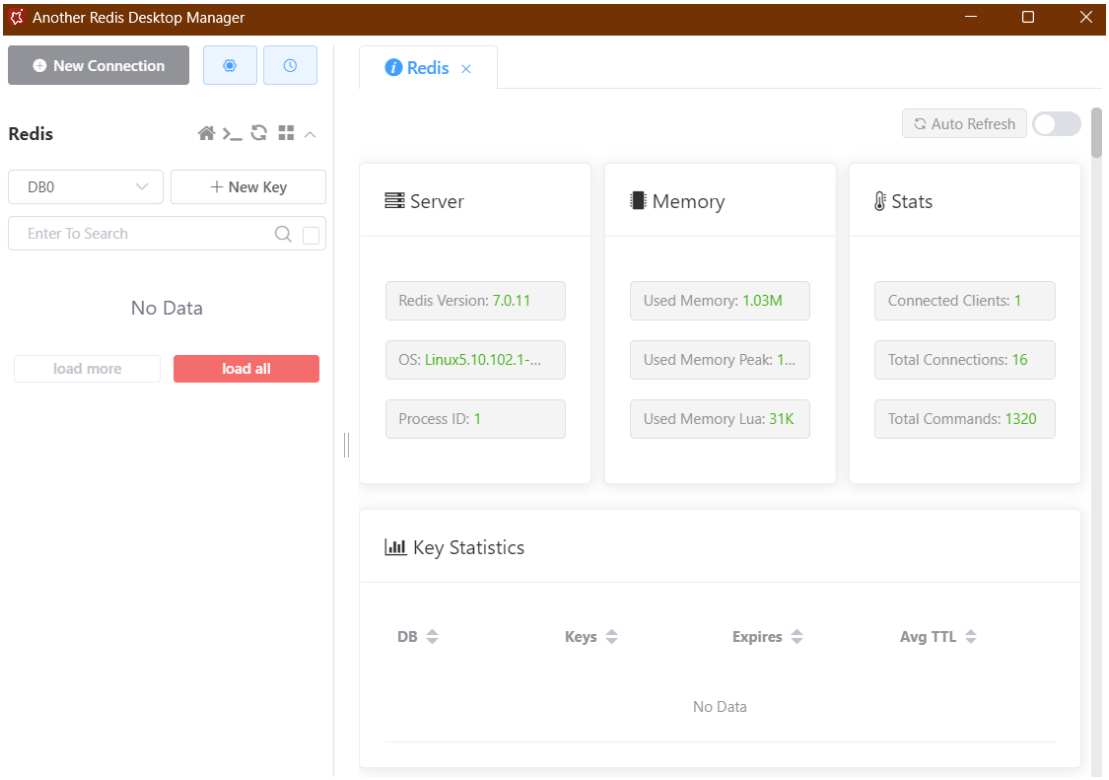
In fact, the only field you will need to fill is the connection name, because the other ones are already set with defaults values that match with our configuration:

```
Host: 127:0.0.1
Port: 6379
```

After that, click on “OK” button.

In the next screen, click on the name or our Connection, Redis.

You will see some important pieces of information about or Redis server.



The second icon (**>_**) to the right of our connection name opens a CLI.

Redis on the browser

If you can’t install a Redis server on your machine, you can use a version of it running in your browser.

All you need to do is access try.redis.io.

You will have an interface like this:

Now you are able to run all the commands we'll see in this course.

Basic Concepts

Redis Tutorial - SET, GET, DEL Key-Value Pair Commands

Get

The GET command is used to retrieve the value stored in a specific key in Redis.

```
GET name
```

This command will return the value stored in the "name" key, if it exists.

Set

The SET command is used to set the value of a key in Redis. It accepts a key and a value as arguments.

```
SET name "John"
```

This command sets the "name" key with the value "John".

Del

The DEL command is used to remove one or more keys from Redis. It accepts one or more arguments representing the keys to be deleted.

```
DEL name
```

This command removes the "name" key and its associated value from Redis.

Keys

The KEYS command is used to retrieve all existing keys in Redis that match a specified pattern. It accepts a search pattern as an argument and returns all keys that match that pattern.

```
KEYS name*
```

This command returns all keys in Redis that start with "user:".

Here, the (*) works similarly to (%) on the command LIKE on SQL queries. It means if you have the keys “name” and “family_name” and run the command KEYS *name*, it should return your two keys.

```
> set name jhon
OK
> set family_name Lennon
OK
> keys *name*
name
family_name
> keys name
name
```

Try all these commands above in your environment and see the result on the “Another Redis Desktop Manager” program we installed early.

Redis has dozens of commands that you can see here: https://redis.io/commands/.

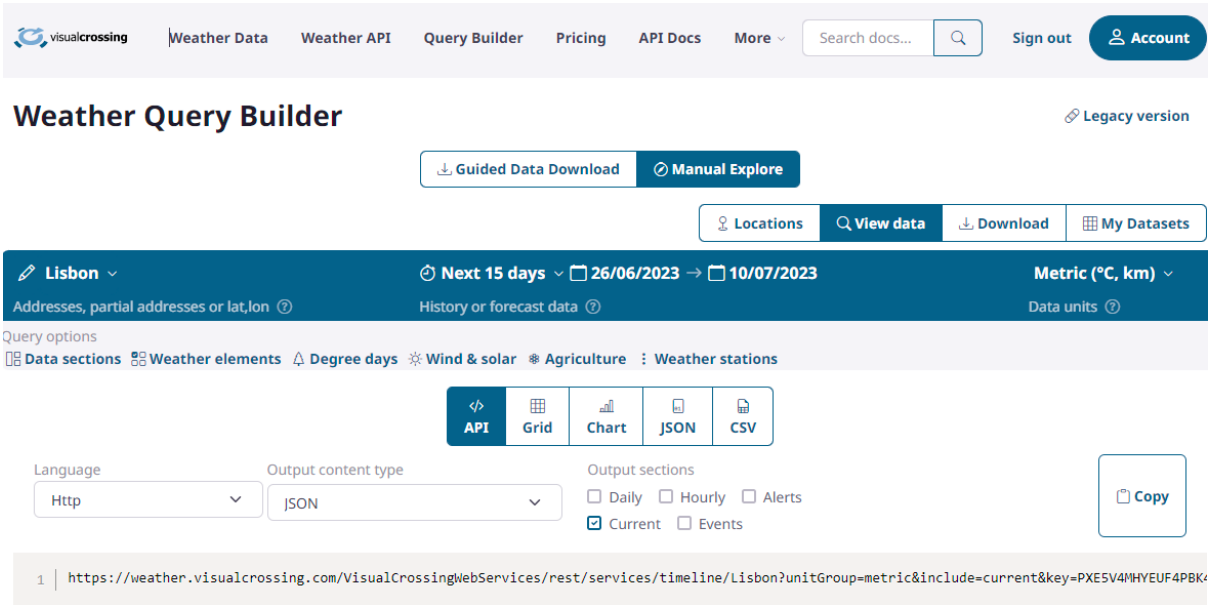
Redis with JavaScript

Now let's create a small project in Javascript to see Redis working in practice.

For this, we will access an API that brings information about the weather.

All you need to do at this point is access the API website, available at **https://www.visualcrossing.com/weather/weather-data-services**, and register, which is fast and free, in order to acquire your key.

Once that's done, you'll have something like this:



Click on the “Copy” button and save this information for later.

Now let's start our javascript project.

For this, we will use node, so if you do not have it installed on your machine, you will have to proceed with the installation.

It's quite simple and the process can be learned here:

https://nodejs.org/en/download.

Once installed, you can continue the course.

To do so, create a directory called **redis-project**, or whatever name you prefer.

We will need some libs to create our application.

Now, inside the folder, run the command:

```
npm init -y
```

This command is used to create a new NodeJS project. It also generates a package.json file, where there will be some settings.

Now we need to install the lib Express, which simplifies the development of applications and APIs.

Run the command:

```
npm install express
```

We also need an HTTP library that makes it easy to send HTTP requests and handle responses in Node.js and the browser. For this, we will use Axios. So, run:

```
npm install axios
```

We can’t forget Redis, of course, so, run:

```
npm install redis
```

To measure the performance of our requests to the API and the cache, we will use the lib performance-now.

Run the command:

```
npm install performance-now
```

At this point, your package.json file is expected to seem like this:

```
{
  "name": "redis-cache",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "keywords": [],
  "author": "Caetano Burjack",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.4.0",
    "express": "^4.18.2",
    "performance-now": "^2.1.0",
    "redis": "^4.6.7"
  }
}
```

Now, let's create our code.

Basically, our application will collect weather information from the API, however, considering that the weather does not change so quickly, we can cache the weather information for a predefined time.

As long as there is such information in the cache, the system will avoid making a request to the API, and when the cache expires, it will update that information again, fetching the API.

A mechanism was also created to calculate the time of requests, so we can clearly see the efficiency between fetching from the cache and the API.

Let's get our hands dirty.

Creates a file in the root called server.js.

Place the following content in the file.

```
//redis-cache/server.js
const express = require("express");
const axios = require("axios");
const performance = require("performance-now");

const app = express();
const { createClient } = require("redis");
const client = createClient();

//URL copied from API
const requestUrl =
`https://weather.visualcrossing.com/VisualCrossingWebServices/rest/servi
ces/timeline/Lisbon?unitGroup=metric&include=current&key=PXE5V4MHYEUF4PB
K4BL9J7NMN&contentType=json`;

app.get("/", async (req, res) => {
  //Creating route (localhost:3000/)
  const startTime = performance(); //Starting timer to measure the
fetching time

  //First, we try to get weather information from cache
  const weatherFromCache = await client.get("weather");
  if (weatherFromCache) {
    //Once this information exists in the cache...
    res.send(weatherFromCache); //It is send as response
    const endTime = performance(); //Recornding the end fetching time
    const loadTime = endTime - startTime; //Calculating the time spent
    console.log(`Fetching time: [Redis] ${loadTime.toFixed(2)}
ms`);//Displaying this information in the console log
    return; //Preventing further code execution
  }

  //Once the weather information does not exists in the cache...
  try {
    const response = await axios.get(requestUrl); //Fetching from
Weather API
    const weatherData = response.data;//Getting data from response
    await client.set("weather", JSON.stringify(weatherData), { EX: 5
});//Creating a register on Redis and defining the expiration time to 5
seconds

    const endTime = performance();//Recornding the end fetching time
    const loadTime = endTime - startTime;//Calculating the time spent
    console.log(`Fetching time: [__API] ${loadTime.toFixed(2)}
ms`);//Displaying this information in the console log

    res.send(weatherData);//It is send as response
  } catch (error) {
    console.error(
      "An error occurred while getting the data from the API:",
```

```
      error
    );
    res
      .status(500)
      .send("An error occurred while getting the data from the API");
  }
});

//Function responsible for starting the server and connecting Redis
client
const startup = async () => {
  await client.connect();//Connecting Redis Client
  app.listen(3000, () => {//Starting the server
    console.log("Server is running on port 3000!");
  });
};

startup(); //Starting this mandatory function
```

Note that most of the instructions are commented out so that you understand what is being done.

Informações importantes:
Remember to change the **requestUrl** for yours.
See that the cache expiration time was set to 5 seconds. You will control this depending on your context.

Spend some time reading this code and all the comments, to understand each step.

Now, let's execute this code. So run:

```
node server.js
```

If everything is ok, you will have something like this in your terminal:

```
Server is running on port 3000!
```

Now, access localhost:3000/.

Considering our cache expiration is set to 5 seconds, reload the page every few seconds so you can see interesting information.

```
Server is running on port 3000!
Fetching time: [__API] 691.20 ms
Fetching time: [Redis] 2.74 ms
Fetching time: [__API] 433.89 ms
```

```
Fetching time: [Redis] 2.38 ms
Fetching time: [Redis] 3.12 ms
Fetching time: [__API] 474.19 ms
Fetching time: [Redis] 2.80 ms
Fetching time: [__API] 658.59 ms
Fetching time: [Redis] 2.29 ms
Fetching time: [__API] 451.36 ms
```

The scenario we use is not fair for testing speed, as our API is on an external server, while our cache is on the local server.

But the important thing is that you now have a working redis implementation with a mechanism to measure its efficiency.

To create a fair comparison environment, you can measure the difference between fetching information from the database and the cache, both on localhost.

Conclusion

If you've come this far, you're expected to have learned the fundamentals of caching with Redis.

The codebase for this project is available at: https://github.com/caetanoburjack/redis-cache.

Now you can understand the potential of caching in the context of web applications and have the necessary knowledge to apply it practically in real projects.

Always remember to resort to official documentation, as many points that I do not cover here are described there.

In the end, if you want to go the extra mile, I'll propose a challenge.

See you on the next course.

Quiz

1 - What does Redis stand for?

a) Recovery Dictionary Server

b) Remote Data Server

c) Replaced Data Server

d) Remote Dictionary Server

2 - What is the primary function of Redis?

a) Storing data in spreadsheet format

b) Managing message queues

c) Relational database

d) In-memory data caching

3 - How is a key-value pair represented in Redis?

a) Linked list

b) Binary tree

c) Hash table

d) Stack

4 - Which programming language is most commonly used to interact with Redis?

a) Python

b) JavaScript

c) Ruby

d) All of the above

Extra Mile

Now that you know the basics of Redis, you can go the extra mile trying to create a fair comparison environment, creating a local database, and testing the caching efficiency in this new scenario.

For that, I'm going to leave a video that I've already tested and that explains in a very simple way how to create the MySQL database and integrate it with NodeJS.