Vector Similarity Search with Redis Enterprise Cloud on AWS

Learning objectives

In this workshop, you will learn

- How to configure Redis Enterprise as a vector database?
- How to invoke embedding and LLM models programmatically using the Sagemaker Jupiter notebook
- How does vector similarity search work with Redis using a sample dataset?
- How do you query vectors stored as Hash/JSON in Redis using search in Redis?

Pre-requisites

Following are some of the skillsets required to complete this training/workshop successfully:

1. Comfortable with AWS management console and cloud formation templates
2. Good in Python as the exercise and coding will be done in Python only
3. General idea of how Machine Learning works like training, testing and deployment of models
4. No prior knowledge of Redis Enterprise is required

Target Personas

Developers, Data architects, Devops, SAs, Team lead and other technical stakeholders who have coding skills

Duration

1-1.5 hours

Tasks

The entire workshop is divided into several labs the details of which are mentioned below and also present in the following Jupyter notebook:

https://github.com/bestarch/redis-vss-getting-started/blob/main/vector_similarity_with_redis.ipynb

Infrastructure set-up

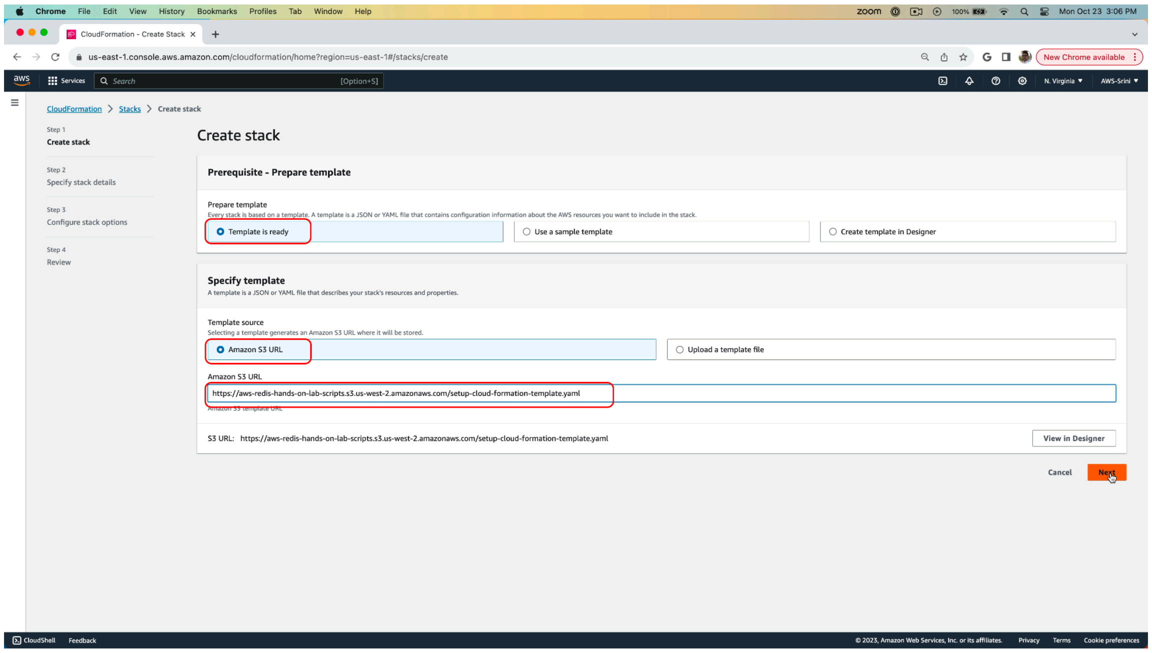
Provision Sagemaker Jupiter instance

Log in using either the root user or the IAM user. In case you are login in using an IAM user, make sure the following permissions are enabled for you:

- AdministratorAccess
- AmazonEC2FullAccess
- AmazonElasticContainerRegistryPublicFullAccess
- CloudWatchFullAccess
- IAMUserChangePassword
- PowerUserAccess
- AWSCloudFormationFullAccess
- AmazonSageMakerFullAccess

Before starting the lab exercises, we will first create a Sagemaker notebook instance. We will use the CloudFormation template to do this.
Follow these steps:

1. Open the AWS management console and navigate to the CloudFormation service.
2. Create a new stack from the following template: https://aws-cloud-formation-template-sagemaker.s3.ap-south-1.amazonaws.com/cloud-formation-template.yaml


us-east-1.console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/create

CloudFormation > Stacks > Create stack

Step 1
Create stack

Step 2
Specify stack details

Step 3
Configure stack options

Step 4
Review

Create stack

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Template is ready] Use a sample template Create template in Designer

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

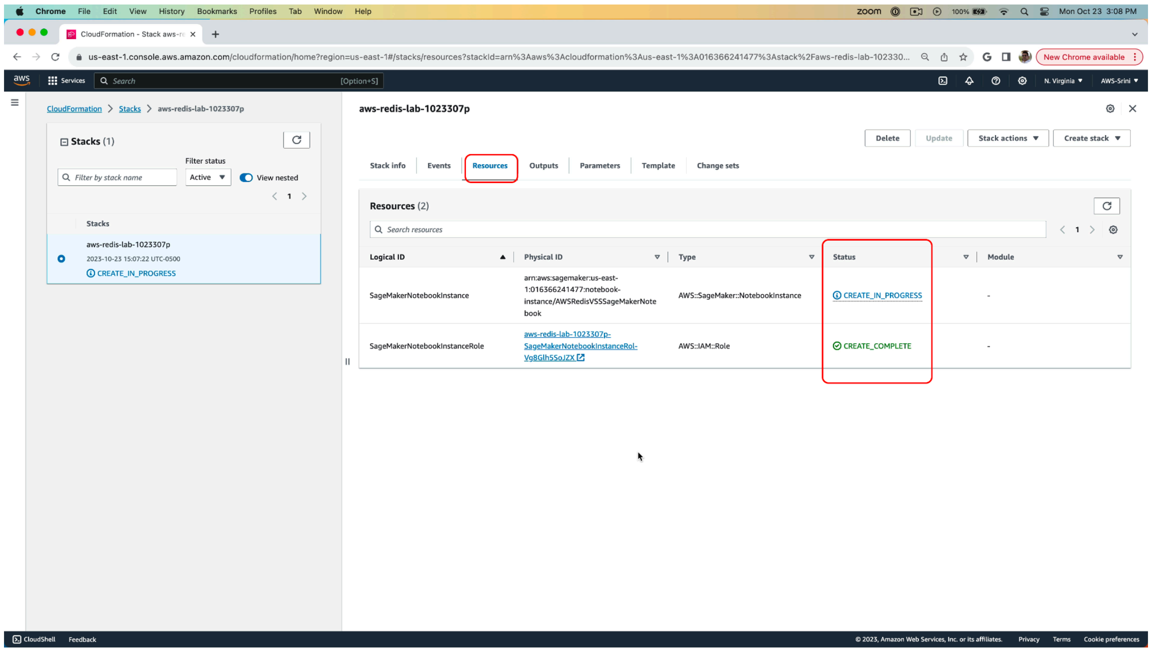
Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

[✓ Amazon S3 URL] Upload a template file

Amazon S3 URL
https://aws-redis-hands-on-lab-scripts.s3.us-west-2.amazonaws.com/setup-cloud-formation-template.yaml
Amazon S3 template URL

S3 URL: https://aws-redis-hands-on-lab-scripts.s3.us-west-2.amazonaws.com/setup-cloud-formation-template.yaml

3. Wait for a few minutes before the resources are ready.



aws-redis-lab-1023307p

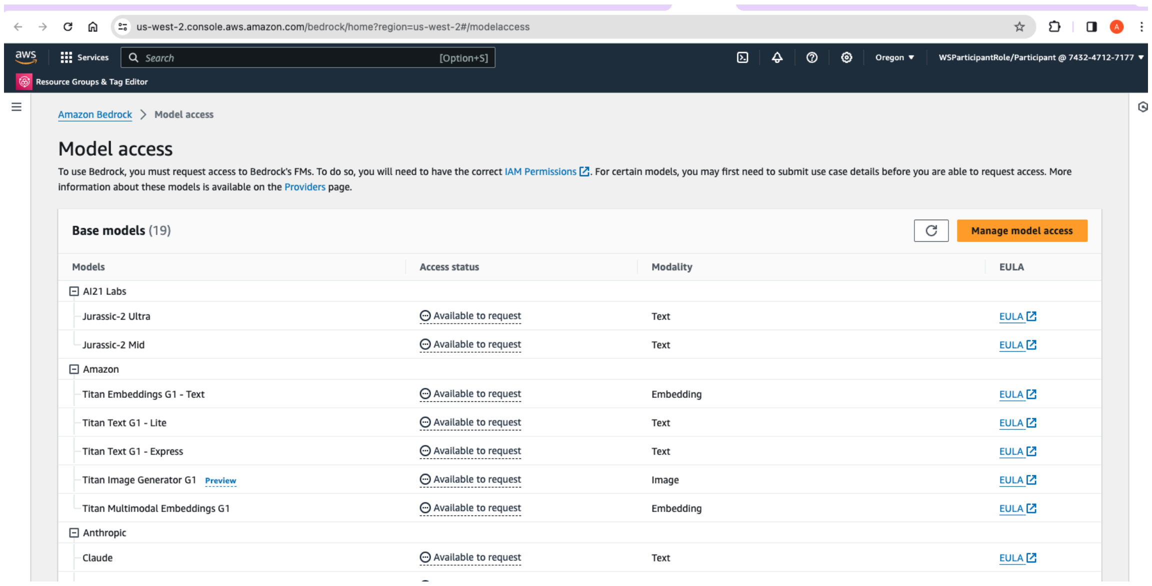
Resources (2)

Logical ID	Physical ID	Type	Status	Module
SageMakerNotebookInstance	arn:aws:sagemaker:us-east-1:016366241477:notebook-instance/AWSRedisVSSSageMakerNotebook	AWS::SageMaker::NotebookInstance	CREATE_IN_PROGRESS	-
SageMakerNotebookInstanceRole	aws-redis-lab-1023307p-SageMakerNotebookInstanceRol-Vg8Glh5SoJZX	AWS::IAM::Role	CREATE_COMPLETE	-

Stacks (1): aws-redis-lab-1023307p, 2023-10-23 15:07:22 UTC-0500, CREATE_IN_PROGRESS

4. Now we have created 2 resources:
    - A Sagemaker notebook instance
    - Suitable IAM roles (to access Sagemaker & Bedrock APIs) needed to complete our exercise using the notebook instance created above

But still, you can't use Amazon Bedrock FM at this time. To access these FMs you need to request them from your AWS management console.



Amazon Bedrock > Model access

Model access

To use Bedrock, you must request access to Bedrock's FMs. To do so, you will need to have the correct IAM Permissions. For certain models, you may first need to submit use case details before you are able to request access. More information about these models is available on the Providers page.

Base models (19)

Models	Access status	Modality	EULA
AI21 Labs			
Jurassic-2 Ultra	Available to request	Text	EULA
Jurassic-2 Mid	Available to request	Text	EULA
Amazon			
Titan Embeddings G1 - Text	Available to request	Embedding	EULA
Titan Text G1 - Lite	Available to request	Text	EULA
Titan Text G1 - Express	Available to request	Text	EULA
Titan Image Generator G1 Preview	Available to request	Image	EULA
Titan Multimodal Embeddings G1	Available to request	Embedding	EULA
Anthropic			
Claude	Available to request	Text	EULA

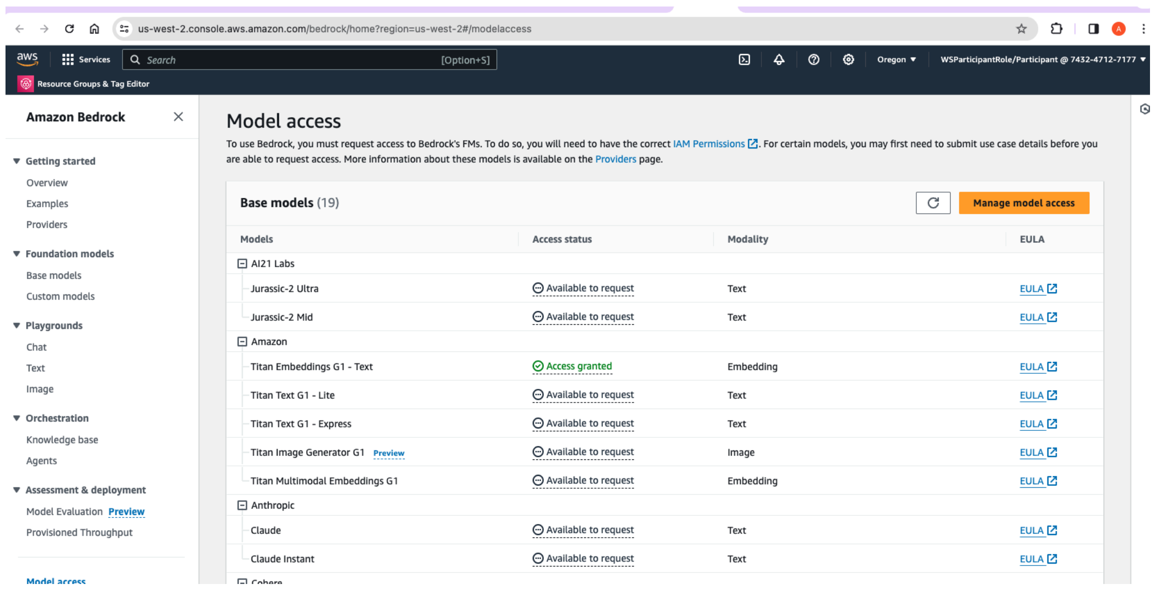
We will use the following FM for our vector embeddings:

```
'amazon.titan-embed-text-v1'
```

5. Goto Bedrock service from the AWS management console and click on the "Base Model" link present in the left navigation menu

6. Locate the 'Amazon Titan Embeddings G1 - Text' model and raise a request for this model by clicking on the 'Available to request' link.

7. This takes a few seconds and if everything is right, this model will be immediately available. Now we will use this model in our coding exercise.



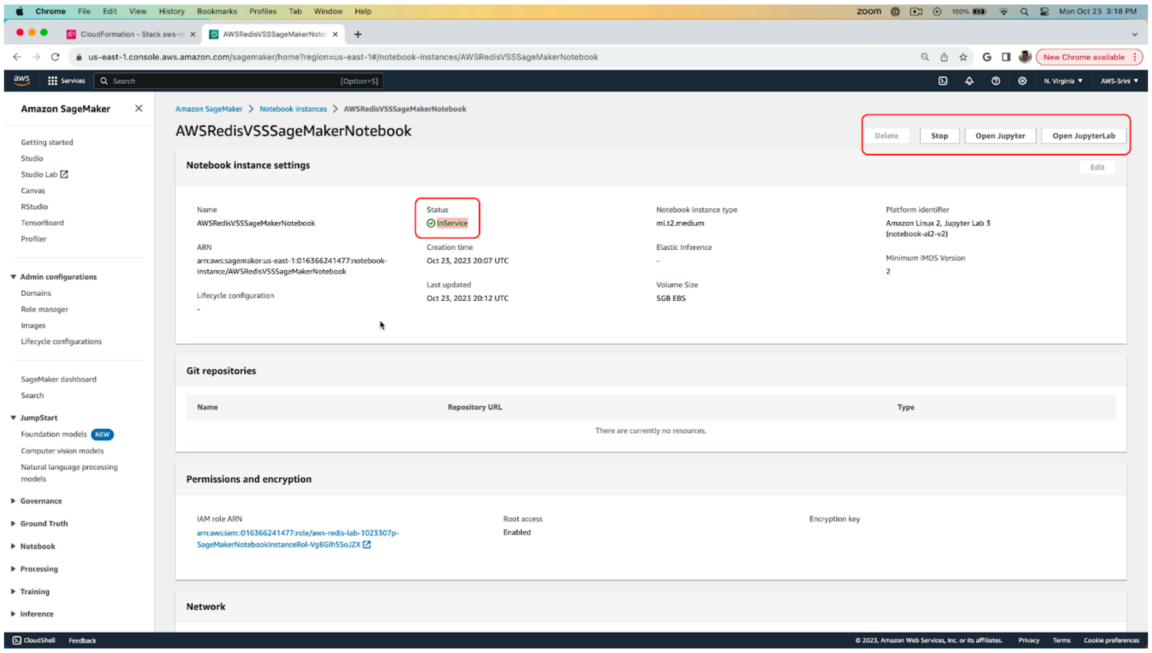
Model access

To use Bedrock, you must request access to Bedrock's FMs. To do so, you will need to have the correct IAM Permissions. For certain models, you may first need to submit use case details before you are able to request access. More information about these models is available on the Providers page.

Base models (19)

Models	Access status	Modality	EULA
AI21 Labs			
Jurassic-2 Ultra	Available to request	Text	EULA
Jurassic-2 Mid	Available to request	Text	EULA
Amazon			
Titan Embeddings G1 - Text	Access granted	Embedding	EULA
Titan Text G1 - Lite	Available to request	Text	EULA
Titan Text G1 - Express	Available to request	Text	EULA
Titan Image Generator G1 Preview	Available to request	Image	EULA
Titan Multimodal Embeddings G1	Available to request	Embedding	EULA
Anthropic			
Claude	Available to request	Text	EULA
Claude Instant	Available to request	Text	EULA
Cohere			

8. Now that all the prerequisites are met, let's navigate to the notebook instance we created above using the Sagemaker service



Amazon SageMaker > Notebook instances > AWSRedisVSSSageMakerNotebook
Delete
Stop
Open Jupyter
Open JupyterLab
Edit

AWSRedisVSSSageMakerNotebook

Notebook instance settings

Name: AWSRedisVSSSageMakerNotebook
ARN: arn:aws:sagemaker:us-east-1:016366241477:notebook-instance/AWSRedisVSSSageMakerNotebook
Lifecycle configuration: -
Status: InService
Creation time: Oct 23, 2023 20:07 UTC
Last updated: Oct 23, 2023 20:12 UTC
Notebook instance type: ml.t2.medium
Elastic Inference: -
Volume Size: 5GB EBS
Platform identifier: Amazon Linux 2, Jupyter Lab 3 (notebook-al2-v2)
Minimum IMDS Version: 2

Git repositories

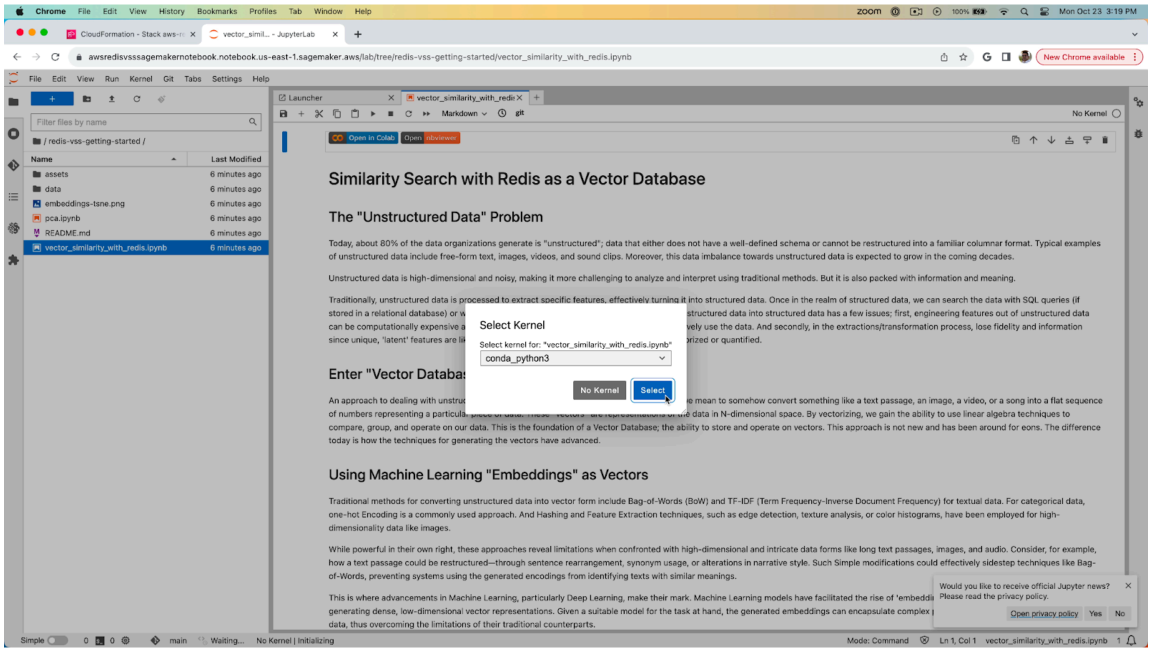
Name	Repository URL	Type

There are currently no resources.

Permissions and encryption

IAM role ARN: arn:aws:iam::016366241477:role/aws-redis-lab-1023307p-SageMakerNotebookInstanceRol-Vg8Gih5SoJZX
Root access: Enabled
Encryption key

Network



Launcher
vector_similarity_with_redis
Markdown
No Kernel
Open in Colab
Open nbviewer
Would you like to receive official Jupyter news? Please read the privacy policy.
Open privacy policy
Yes
No

/ redis-vss-getting-started /

Name	Last Modified
assets	6 minutes ago
data	6 minutes ago
embeddings-tsne.png	6 minutes ago
pca.ipynb	6 minutes ago
README.md	6 minutes ago
vector_similarity_with_redis.ipynb	6 minutes ago

Similarity Search with Redis as a Vector Database

The "Unstructured Data" Problem

Today, about 80% of the data organizations generate is "unstructured"; data that either does not have a well-defined schema or cannot be restructured into a familiar columnar format. Typical examples of unstructured data include free-form text, images, videos, and sound clips. Moreover, this data imbalance towards unstructured data is expected to grow in the coming decades.

Unstructured data is high-dimensional and noisy, making it more challenging to analyze and interpret using traditional methods. But it is also packed with information and meaning.

Traditionally, unstructured data is processed to extract specific features, effectively turning it into structured data. Once in the realm of structured data, we can search the data with SQL queries (if stored in a relational database) or w[illegible] structured data into structured data has a few issues; first, engineering features out of unstructured data can be computationally expensive a[illegible] vely use the data. And secondly, in the extractions/transformation process, lose fidelity and information since unique, 'latent' features are li[illegible] orized or quantified.

Select Kernel

Select kernel for: "vector_similarity_with_redis.ipynb"

conda_python3

No Kernel [✓ Select]

Enter "Vector Databas[illegible]

An approach to dealing with unstruc[illegible] e mean to somehow convert something like a text passage, an image, a video, or a song into a flat sequence of numbers representing a particula[illegible] the data in N-dimensional space. By vectorizing, we gain the ability to use linear algebra techniques to compare, group, and operate on our data. This is the foundation of a Vector Database; the ability to store and operate on vectors. This approach is not new and has been around for eons. The difference today is how the techniques for generating the vectors have advanced.

Using Machine Learning "Embeddings" as Vectors

Traditional methods for converting unstructured data into vector form include Bag-of-Words (BoW) and TF-IDF (Term Frequency-Inverse Document Frequency) for textual data. For categorical data, one-hot Encoding is a commonly used approach. And Hashing and Feature Extraction techniques, such as edge detection, texture analysis, or color histograms, have been employed for high-dimensionality data like images.

While powerful in their own right, these approaches reveal limitations when confronted with high-dimensional and intricate data forms like long text passages, images, and audio. Consider, for example, how a text passage could be restructured—through sentence rearrangement, synonym usage, or alterations in narrative style. Such Simple modifications could effectively sidestep techniques like Bag-of-Words, preventing systems using the generated encodings from identifying texts with similar meanings.

This is where advancements in Machine Learning, particularly Deep Learning, make their mark. Machine Learning models have facilitated the rise of 'embeddi[illegible] generating dense, low-dimensional vector representations. Given a suitable model for the task at hand, the generated embeddings can encapsulate complex [illegible] data, thus overcoming the limitations of their traditional counterparts.

No Kernel | Initializing   Mode: Command   Ln 1, Col 1   vector_similarity_with_redis.ipynb

Lab 1: Load "Bikes" Dataset

To investigate Vector Similarity, we'll use a subset of the Bikes dataset, a relatively simple synthetic dataset.

The dataset has 11 bicycle records in a JSON file named `bikes.json` and includes the fields `model`, `brand`, `price`, `type`, `specs`, and `description`.
The `description` field is particularly interesting for our purposes since it consists of a free-form textual description of a bicycle.

For hands-on exercise, navigate to the '**Lab 1**' section of the Jupyter notebook.

Lab 2: Store the 'bikes' dataset in Redis

At this point, we have loaded our **'bikes'** dataset in memory.
Now, let's configure **Redis** and store this dataset as a collection of JSON documents. For this purpose, we will use the **Redis Enterprise Cloud** database.
In the next section, we will leverage the same Redis database to store vector embeddings as well.

***Note**: In case Redis Enterprise Cloud is not available, use Redis Stack open-source.*

This can be done by installing Redis Stack Server using:

```
!curl -fsSL
https://packages.redis.io/redis-stack/redis-stack-server-6.2.6-v7.focal.
x86_64.tar.gz -o redis-stack-server.tar.gz
!tar -xvf redis-stack-server.tar.gz
!./redis-stack-server-6.2.6-v7/bin/redis-stack-server --daemonize yes
oss = True
```

For hands-on exercise, navigate to the '**Lab 2**' section of the Jupyter notebook.

Lab 3: Vectorize the 'description' field inside JSON documents

Now, let's vectorize the **'description'** field present inside the JSON documents and finally load these vector embeddings inside the same document.
We will first collect all the Redis keys for the bikes.
We'll use the keys as a parameter to the `JSON.MGET` command, along with the JSONPath expression `$.description` to collect the descriptions in a list. We will iterate through this list to create the embeddings.

To create the vector embeddings we can leverage:

1. **Recommended**: Use AWS Bedrock API and invoke the Titan embedding model
2. **Alternative**: Use the SentenceTransformers framework to generate embeddings for the bike's descriptions. Sentence-BERT (SBERT) is a BERT model modification that produces consistent and contextually rich sentence embeddings. SBERT improves tasks like semantic

search and text grouping by allowing for efficient and meaningful comparison of sentence-level semantic similarity.

Both of these options are discussed in detail in the notebook.
For hands-on exercise, navigate to the ‘**Lab 3**’ section of the Jupyter notebook.

Lab 4: Execute queries & verify the data in Redis

At this point, we should have created all the vector embeddings and loaded them into our Redis database.
Next, we will perform the following tasks in this lab:

- Create an index to make the bikes searchable
- Write queries to search the documents
- Semantic searching using VSS query, hybrid query & range query

For hands-on exercise, navigate to the ‘**Lab 4**’ section of the Jupyter notebook.

Final: Wrap-up

In this lab exercise, we learned how Redis Enterprise provides powerful search capabilities over structured and unstructured data. Redis support for vector data can enrich and enhance the user's search experience. Although we focused on generating embeddings for unstructured data, the vector similarity approach can equally be employed with structure data, as long as a suitable vector generation technique is used.

The references below can help you learn more about Redis search capabilities:

- https://redis.com/solutions/use-cases/vector-database/
- https://redis.io/docs/stack/search/
- https://redis.io/docs/stack/search/indexing_json/