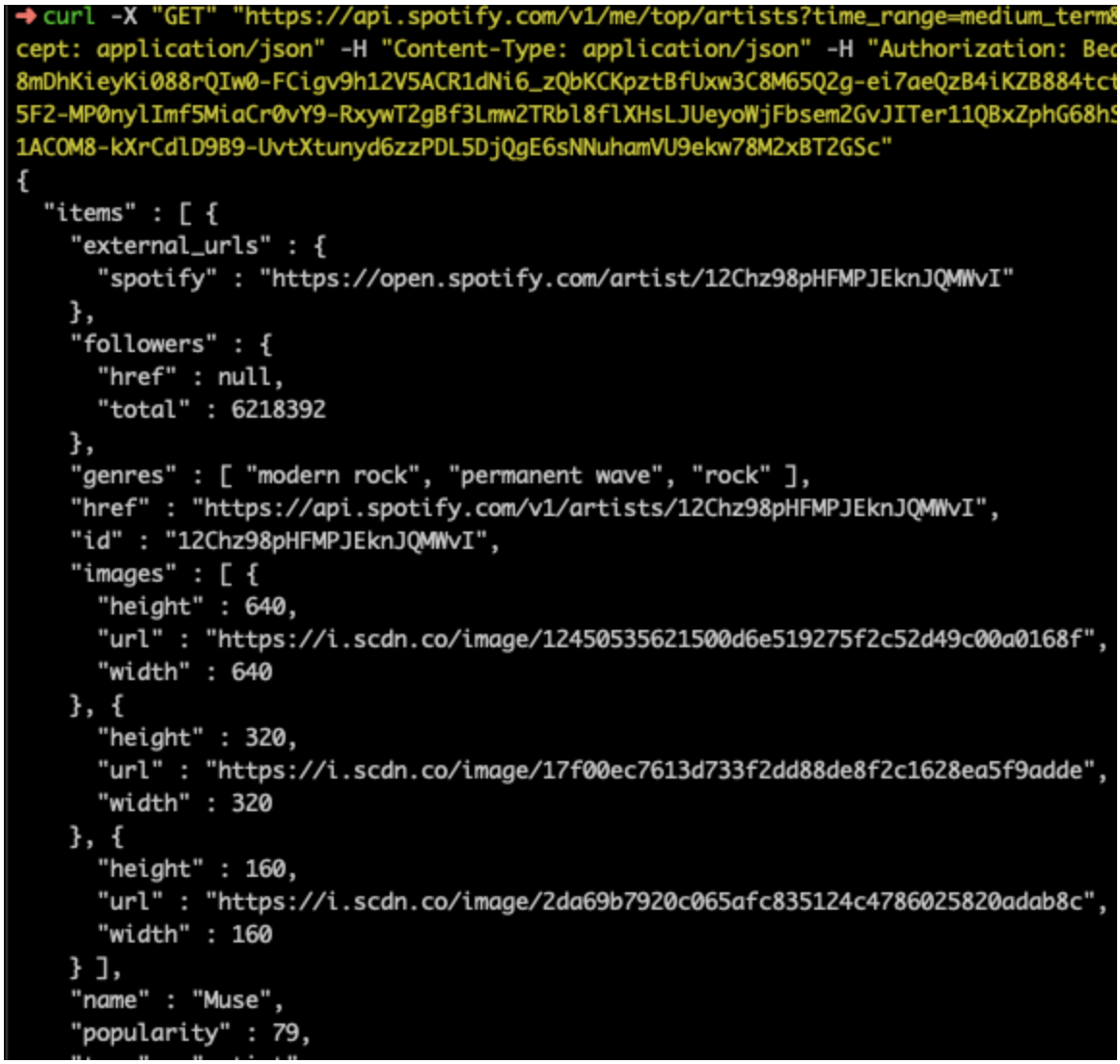
CloudSchool final project

- **Python application**:
  Be the developer - code your own application server.

- **Open REST APIs**
  An open(public) API is a publicly available application programming interface that provides developers with programmatic access to a web service.
  There are thousands of companies that expose APIs for their services freely, such as “IMDB”, “Instagram”, “Spotify” and so forth, and there are even API marketplace sites like https://rapidapi.com that offer lots of free, public REST APIs.
  Your application is required to query at least one public REST API for valuable data that will provide an added value to the user.



Example of using a Spotify service API

Create a Python app that can query the external API service of your choice:

- Use a configuration file which will include static configurations (e.g. credentials, URLs, etc.)
- Support multiple query types in your script that returns data from the API - you can aggregate data / use the RDS as caching / extend the API to support more query types etc.

- **Database:**
  - Create MySql RDS
  - Use it to save data that you pull from the API above
  - Add the URL & credentials to your script config file

- **Chef:**
  - Create a cookbook that installs your script and executes it

- **Hashicorp:**
  - Terraform
    - Create your infrastructure as described in the diagram using terraform
      - Launch Configurations
        - The user-data should install chefdk & git, download your cookbook code and execute it
      - Auto Scaling group using the LaunchConfiguration above
        - Define scaling rules using metrics that your service sends to CloudWatch
      - Load Balancer for forwarding traffic to the instances in the group above
      - VPC
      - Key-pair
      - Subnets
      - Security groups
  - Vault
    - Use credentials from vault instead of your configuration file
  - Consul
    - Use consul-template to generate your configuration file dynamically

- **Jenkins:**
  - Create CI/CD process with basic tests for your script
  - Upload the artifact to S3
  - Update the LaunchConfiguration to use the new artifact on each change

- **Monitoring:**

    - Create a grafana / cloudwatch dashboard with your service metrics
    - The script should send some metrics to AWS CloudWatch for basic monitoring and auto scaling features

---

Level 1 (manual)

Infra: (all manually created)

- Launch Configuration
- Auto scaling group
- Load Balancer
- Security groups
- RDS

Script:

- Able to respond for multiple query types and support scale up/down automatically

---

Level 2 (semi automated)

Infra: (using terraform)

- Launch Configuration
- Auto scaling group
- Load Balancer
- Security groups
- RDS
- Cloudwatch dashboard

Script:

- Able to respond for multiple query types and support scale up/down automatically
- Installed automatically using a cookbook

Level 3 (fully automated)

Infra: (using terraform)

- VPC
- Subnets
- Launch Configuration

- Auto scaling group
- Load Balancer
- Security groups
- RDS
- Cloudwatch dashboard

Script:

- Able to respond for multiple query types and support scale up/down automatically
- Installed automatically using a cookbook
- Credentials should be using Vault

Level 4 (real DevOps)

Infra: (using terraform)

- VPC
- Subnets
- KeyPair
- Launch Configuration
- Auto scaling group
- Load Balancer
- Security groups (Concrete, For every resource!)
- RDS
- Grafana dashboard using cloudwatch metrics
- Support instances (Jenkins / Consul / Vault / Grafana - can all be on a single instance)

Script:

- Able to respond for multiple query types and support scale up/down automatically
- Installed automatically using a cookbook
- Credentials should be using Vault - passwords should be temporary per-demand and auto generated on runtime only
- Configuration file should be automated with consul template
- Full CI/CD process with tests. If all tests pass - it uploads the artifact to S3 and updates consul key with the new version number (Chef uses consul to check which version to install)