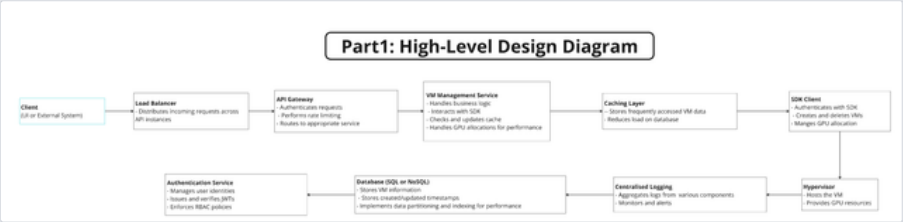
Nitish-(Architect-Techtest)

`All the collaterals, screen shots and logs has been provided inside the result folder as a part of the GitHub repo.` GitHub - nitish-gautam/architect-techtest: This is a FastAPI project that involves creating endpoints to provision and delete VMs, with all operations saved to a local database including created/updated timestamps.

Part 1: High-Level Design

Design a platform that will allow end-users to create virtual machines and leverage GPUs.
Create a high-level design, including a diagram.



Overview

The platform will provide an API for managing virtual machines (VMs) with GPU capabilities, including provisioning and deleting VMs. The API will interact with a hypervisor SDK to perform the actual VM operations. The system will maintain a database to keep track of the VMs and their states.

Design Considerations

Scalability: The system should handle an increasing number of VM creation and deletion requests efficiently.

Security: Ensure that only authorised users can create or delete VMs.

Performance: Optimise the system for quick response times.

Reliability: Ensure the system is highly available and fault-tolerant.

Best Practices: Use appropriate design patterns and best practices to ensure maintainability and extensibility.

Components

Client: The end-user interface or external systems that interact with the API.

Load Balancer: Distributes incoming API requests across multiple instances of the API Gateway to ensure high availability and scalability.

API Gateway: Handles incoming API requests, performs authentication and rate limiting, and routes the requests to the appropriate service. Ensures security by validating JWTs and enforcing RBAC policies.

VM Management Service: Contains the business logic for managing VMs, including provisioning, deletion, and GPU allocation. Implements caching strategies to enhance performance and reduce load on the database.

Caching Layer: A caching mechanism (like Redis) to store frequently accessed VM data, reducing the load on the database and improving response times.

SDK Client: Provides methods for interacting with the hypervisor to perform VM operations and manage GPU allocation.

Hypervisor: The actual environment where VMs are created and managed, providing necessary GPU resources..

Centralised Logging: Aggregates logs from various components, monitors the system, and sends alerts in case of issues. Ensures traceability and aids in debugging.

Database: Stores information about the VMs, including their states and timestamps for creation and updates. Implements data partitioning and indexing strategies to enhance performance and scalability. Can be a SQL or NoSQL database depending on the specific use case and load.

Authentication Service: Manages user identities, issues and verifies JWTs, and enforces Role-Based Access Control (RBAC) policies. Ensures that only authorized users can perform certain actions.

Design Patterns and Best Practices consideration

Microservices Architecture: Each component (API Gateway, VM Management Service, etc.) can be developed as a microservice, allowing independent deployment and scaling.

Circuit Breaker Pattern: Protects the system from cascading failures by stopping calls to a failing service after a threshold is reached.

Retry Pattern: Handles transient failures by retrying a failed operation a certain number of times before giving up.

CI/CD Pipelines: Ensures that code changes are tested and deployed automatically, improving development efficiency and reliability.

Scalability

Horizontal Scaling: The API Gateway and VM Management Service can be horizontally scaled by adding more instances and using a load balancer to distribute traffic.

Database Partitioning: The database can be partitioned to handle large volumes of data efficiently.

Caching: Implement caching for frequently accessed data to reduce database load.

Security

Authentication and Authorization: Use JWTs for stateless authentication and enforce RBAC policies to ensure that only authorized users can access certain endpoints.

Rate Limiting: Implement rate limiting at the API Gateway to prevent abuse and ensure fair usage.

Encryption: Encrypt sensitive data both at rest and in transit.

Performance

Load Balancing: Use a load balancer to distribute requests evenly across API instances.

Caching: Implement caching strategies to reduce the load on the database and speed up response times.

Efficient Database Access: Use indexing and partitioning in the database to speed up data retrieval and updates.

Asynchronous Processing: Use asynchronous processing for long-running tasks to improve responsiveness.

Part 2: Technical Implementation

You will implement an API for managing VMs based on the design from Part 1.
This involves creating endpoints to provision and delete VMs, with all operations saved to a local database including created/updated timestamps.

Task

- **Define the Database Model**: Define the database model for storing VM information.
- **Implement the API Endpoints**: Define the endpoints for creating and deleting VMs.
- **Save VM States**: Save the state of the VMs in the local database.

Additional Info

An SDK is provided to simulate the provisioning and deletion of VMs on the hypervisor layer; `from sdk import Client`.
It includes methods for authentication, creating VMs, and deleting VMs.
The SDK client's API key is set at the environment level and is not required in requests.

In the implementation of Part 2, we focused on a streamlined approach to managing VMs through a FastAPI application. However, several aspects from the high-level design diagram in Part 1 were simplified or omitted. Here's a detailed comparison:

PART1 Design elements NOT implemented in Part2:

1. **Load Balancer**: The current implementation does not include a load balancer, as it runs a single instance of the FastAPI application.
2. **API Gateway**: The current implemenation directly handle requests within the FastAPI application without an intermediate gateway. Authentication is managed directly via FastAPI's OAuth2PasswordBearer.
3. **Caching Layer**: The current implementation directly interacts with the database without an intermediate cache.
4. **Centralised Logging**: Basic logging is used within the FastAPI application, but there is no centralised logging system to aggregate logs from different sources.
5. **Authentication Service**: Authentication and JWT token management are directly integrated within the FastAPI application without a separate authentication service. RBAC policies are not explicitly enforced.
6. **Hypervisor and GPU Utilisation**: While the SDK simulates hypervisor interactions, actual GPU utilisation management is not implemented. The SDK handles VM creation and deletion without specifying GPU allocations.
7. **Database Performance Features**: The SQLite database is used for simplicity, without specific configurations for data partitioning or advanced indexing.

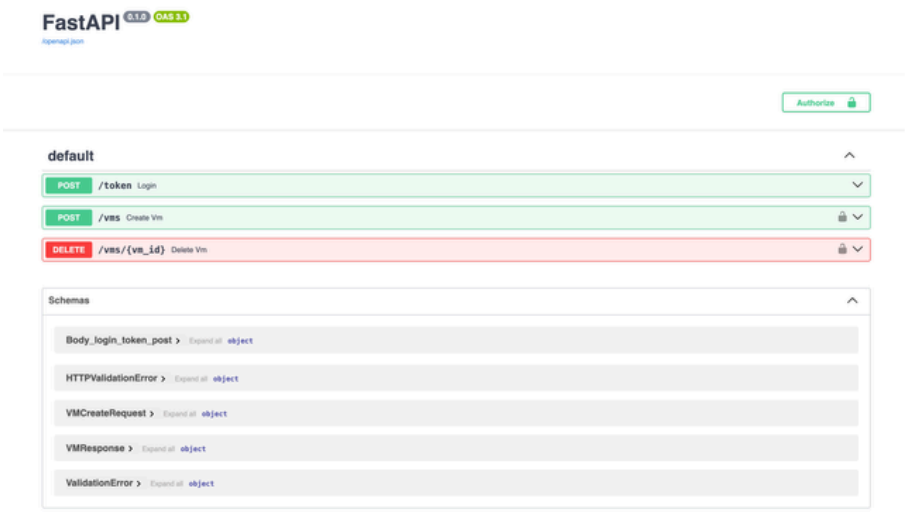
Implemented Features:

- **FastAPI Application**: Manages API endpoints for VM creation and deletion.
- **SQLite Database**: Stores VM data including creation and update timestamps.
- **JWT Authentication**: Secures API endpoints with token-based authentication.
- **SDK Interaction**: Uses an SDK to simulate VM provisioning and deletion.

```
(venv) ➜  architect-techtest /Users/nitishgautam/.local/bin/poetry shell
Virtual environment already activated: /Users/nitishgautam/Desktop/code/test/architect-techtest/venv
(venv) ➜  architect-techtest uvicorn main:app --reload

INFO:     Will watch for changes in these directories: ['/Users/nitishgautam/Desktop/code/test/architect-techtest/architect-techtest']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [86697] using StatReload
INFO:     Started server process [86699]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:51617 - "GET / HTTP/1.1" 404 Not Found
INFO:     127.0.0.1:51618 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:51618 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:51622 - "POST /token HTTP/1.1" 200 OK
```

url: http://127.0.0.1:8000/docs#/



1. **GET JWT TOKEN**

Please note you will need Keys to run that shall be extracted from `.env` file.

```
SDK_API_KEY=your_api_key_here
SECRET_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0dXNlciJ9.hOROJV-rUAAGWs029hBpc5Unx-7un4g4-TcQxeTS4N4
REDIS_HOST=localhost
REDIS_PORT=6379
```

`This command should return a JSON response with an access_token. Save this token for the subsequent requests.`

```
curl -X 'POST' \
  'http://127.0.0.1:8000/token' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'username=testuser&password=testpassword'
```

```
➜  architect-techtest-tested-version git:(main) ✗ curl -X 'POST' \
  'http://127.0.0.1:8000/token' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'username=testuser&password=testpassword'
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0dXNlciJ9.hOROJV-rUAAGWs029hBpc5Unx-7un4g4-TcQxeTS4N4","token_type":"bearer"}%
```

2. **Create a VM**

Replace `<your_access_token>` with the actual token you received from the previous step.

```
curl -X 'POST' \
  'http://127.0.0.1:8000/vms' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer <your_access_token>' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "test-vm",
  "cpu_cores": 2,
  "memory": 4096,
  "disk_size": 50,
  "public_ip": "192.168.1.1",
  "labels": ["test"]
}'

```

```
➜  architect-techtest git:(main) ✗ curl -X 'POST' \
  'http://127.0.0.1:8000/vms' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0dXNlciJ9.hOROJV-rUAAGWs029hBpc5Unx-7un4g4-TcQxeTS4N4' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "test-vm",
  "cpu_cores": 2,
  "memory": 4096,
  "disk_size": 50,
  "public_ip": "192.168.1.1",
  "labels": ["test"]
```

```
}'

{"id":"b91e46cf-98f7-449d-8ec9-7367eac3cf45","name":"test-vm","cpu_cores":2,"memory":4096,"disk_size":50,"public_ip":"192.168.1.1","labels":["test"],"status":"create
 ➜  architect-techtest git:(main) ✗
```

3. **Delete a VM**

   Replace `<your_access_token>` with the actual token you received from the first step and `<vm_id>` with the actual VM ID you received from the VM creation step.

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/vms/<vm_id>' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer <your_access_token>'

```

```
➜  architect-techtest git:(main) ✗ curl -X 'DELETE' \
  'http://127.0.0.1:8000/vms/6c7e3b27-7159-4e0a-9e30-3d44284574cd' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ0ZXN0dXNlciJ9.hOROJV-rUAAGWs029hBpc5Unx-7un4g4-TcQxeTS4N4'

{"id":"6c7e3b27-7159-4e0a-9e30-3d44284574cd","name":"test-vm","cpu_cores":2,"memory":4096,"disk_size":50,"public_ip":"192.168.1.1","labels":[],"status":"deleted","cre

```

PYTEST for the above endpoints

```
(venv) ➜  architect-techtest git:(main) ✗ /Users/nitishgautam/.local/bin/poetry run pytest

============================================================================================================================================================ test session sta
platform darwin -- Python 3.10.7, pytest-8.2.2, pluggy-1.5.0
rootdir: /Users/nitishgautam/Desktop/code/customer_test/architect-techtest
configfile: pyproject.toml
plugins: anyio-4.4.0
collected 10 items

tests/test_client.py .
tests/test_main.py ..
tests/unit/sdk/test_client.py .......

============================================================================================================================================================ warnings summar
../../test/architect-techtest/venv/lib/python3.10/site-packages/fastapi/openapi/models.py:55
  /Users/nitishgautam/Desktop/code/test/architect-techtest/venv/lib/python3.10/site-packages/fastapi/openapi/models.py:55: DeprecationWarning: `general_plain_validat
    return general_plain_validator_function(cls._validate)

../../test/architect-techtest/venv/lib/python3.10/site-packages/pydantic_core/core_schema.py:4034
  /Users/nitishgautam/Desktop/code/test/architect-techtest/venv/lib/python3.10/site-packages/pydantic_core/core_schema.py:4034: DeprecationWarning: `general_plain_va
    warnings.warn(

main.py:21
  /Users/nitishgautam/Desktop/code/customer_test/architect-techtest/main.py:21: MovedIn20Warning: The ``declarative_base()`` function is now available as sqlalchemy.
    Base = declarative_base()

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
============================================================================================================================================ 10 passed, 3 warnings
(venv) ➜  architect-techtest git:(main) ✗
(venv) ➜  architect-techtest git:(main) ✗
```