**# Scalable Services Assignment**

**Student name: Vidhya V**

**Student ID: 2023TM93607**

**## Microservice Based Blogging Application**

**A logo of a whale and a whale

Description automatically generated** **A blue and white logo

Description automatically generated** A black text on a white background

Description automatically generated

**This project is a microservices-based blogging platform.**

**It demonstrates how to design and implement a scalable, maintainable application using a microservices architecture.**

**\*\*The platform consists of the following services:\*\***

**- \*\*User Service\*\*: Manages user registration, authentication, and profile management.**

**- Link to Repo: https://git.foss.life/axel/ss-assignment-user-service**

**- \*\*Post Service\*\*: Handles creating, retrieving, updating, and deleting blog posts.**

**- Link to Repo:** [**https://github.com/vidhy27/SCALABLE\_SERVICES**](https://github.com/vidhy27/SCALABLE_SERVICES)

**- \*\*Comment Service\*\*: Allows users to add comments to posts, retrieve comments, and delete their own comments.**

**- Link to Repo: https://git.foss.life/axel/ss-assignment-comment-service**

**- \*\*Like Service\*\*: Enables users to like or unlike posts and keeps track of the number of likes per post.**

**- Link to Repo: https://github.com/freakcap/LikeMicroservice**

**- \*\*API Gateway\*\*: Acts as a single entry point to the system, routing client requests to the appropriate services and handling cross-cutting concerns like authentication and logging.**

**- Link to Repo:** [**https://github.com/vidhy27/SCALABLE\_SERVICES/blob/main/ss-assignment-api-gateway.zip**](https://github.com/vidhy27/SCALABLE_SERVICES/blob/main/ss-assignment-api-gateway.zip)

**Services communicate with each other using synchronous HTTP/REST calls. Authentication is managed using JSON Web Tokens (JWT), with the API Gateway centralizing authentication and authorization to ensure secure and consistent access control across all services.**

**This application serves as a foundation for building complex, distributed systems and can be extended with additional features and services as needed.**

**### Video Links of Vidhya:**

[**https://drive.google.com/drive/folders/11C4baGeSn21cX\_RCuN5ZXiY32DIFI\_b3?dmr=1&ec=wgc-drive-hero-goto**](https://drive.google.com/drive/folders/11C4baGeSn21cX_RCuN5ZXiY32DIFI_b3?dmr=1&ec=wgc-drive-hero-goto)

**API Validation:** [**https://github.com/vidhy27/SCALABLE\_SERVICES/blob/main/blogging-platform/API\_VALIDATION.docx**](https://github.com/vidhy27/SCALABLE_SERVICES/blob/main/blogging-platform/API_VALIDATION.docx)

**### Video Links of team:**

**- Video #1: https://drive.google.com/file/d/1syYOzdtyGnBaRrZ3QlkEVrRPYwLpBoXh/view?usp=drive\_link**

**- Video #2: https://drive.google.com/file/d/1b5sYK2bK\_PYpOY8RbW7n9lTSqIEsG\_d9/view?usp=drive\_link**

**Table of Contents**

**Introduction**

**Architecture**

**Components of the Application**

**Communication and Integration**

Security

-Authentication and Authorization

-Data Validation and Sanitization 7

-API Gateway

Implementation Details :

**Database Schema**

**-**User Service

-Post Service

-Comment Service

-Like Service

**API Endpoints :**

User Service

Post Service

Comment Service

Likes Service

**Insomnia Screenshots**

**Docker Implementation:**

Integrated user & post service

**Team Contributions**

|  |  |  |
| --- | --- | --- |
| Kuchibhotla Abhirama Krishna | 2023TM93673 | ● UserService ● CommentService ● APIGateway ● Documentation |
| Vidhya V | 2023TM93607 | ● PostService ● Documentation |
| Kulkarni Pranav Jagadish | 2023TM93617 | ● LikeService ● Documentation |
| Anusha K | 2023TM93552 | ● Documentation |

**Introduction**

This document provides a comprehensive overview of a microservices-based blogging platform designed with scalability, maintainability, and extensibility in mind.

The various services are built using Node.js, Express, JavaScript, TypeScript, MongoDB, and PostgreSQL, leveraging a microservices architecture to decouple functionalities and enable independent development and deployment of services.

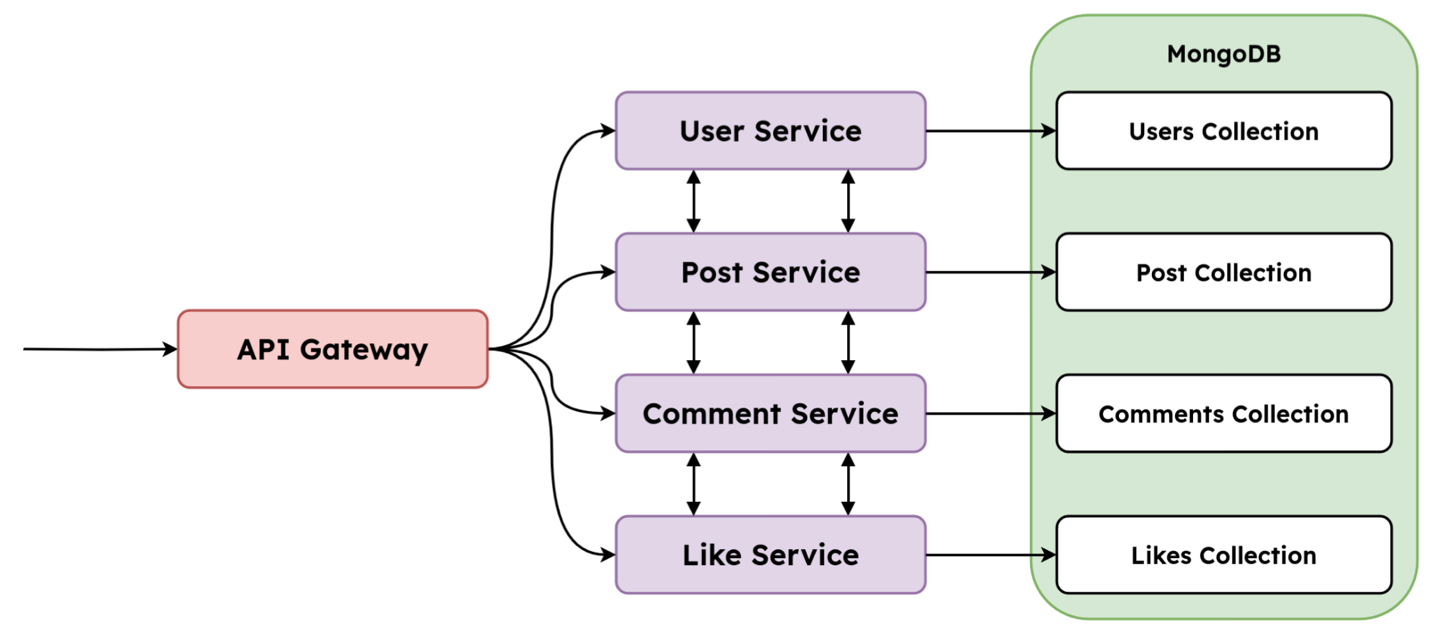
The platform includes the following core services:

* **●  User Service:** Manages user registration, authentication, and profile retrieval.
* **●  Post Service:** Handles creation, retrieval, updating, and deletion of blog posts.
* **●  Comment Service:** Manages comments on posts, including adding, retrieving, and deleting comments.
* **●  Like Service:** Enables users to like or unlike posts.
* **●  API Gateway:** Acts as a single entry point for clients, routing requests to appropriate services and handling cross-cutting concerns like authentication and logging.

**Architecture**

The application adopts a microservices architecture, where each service is responsible for a specific domain of functionality. This approach offers several advantages:

* **●  Scalability:** Services can be scaled independently based on demand.
* **●  Maintainability:** Smaller codebases are easier to manage and update.
* **●  Flexibility:** Services can be developed using different technologies if desired.
* **●  Fault Isolation:** Issues in one service are less likely to impact others.



The services communicate primarily through **HTTP/REST** APIs, and an **API Gateway** is introduced to simplify client interactions and centralize concerns like authentication and request logging.

**Components of the Application**

**1. UserService**

* ●  Manages user accounts, including registration and authentication.
* ●  Provides JWT tokens upon successful login for authentication. 2. **PostService**
* ●  Handles CRUD (Create, Read, Update, Delete) operations for blog posts.
* ●  Ensures only authenticated users can create, update, or delete posts.
* ●  Publicly accessible endpoints for retrieving posts.

**3. CommentService**

* ●  Allows users to add comments to posts.
* ●  Retrieves comments associated with a specific post.
* ●  Enables users to delete their own comments.

**4. LikeService**

* ●  Enables users to like or unlike posts.
* ●  Keep track of likes per post.
* ●  Ensures users can only like a post once.

**5. APIGateway**

* ●  Serves as the single entry point for client requests.
* ●  Routes requests to the appropriate microservice.
* ●  Handles authentication and authorization centrally.
* ●  Implements logging for monitoring and debugging.

**Communication and Integration**

**Security  
Authentication and Authorization**

* **●  JWT Tokens:** The User Service issues JSON Web Tokens (JWT) upon successful authentication.
* **●  Token Verification:** The API Gateway and each service are capable of verifying JWT tokens for protected routes.
* **●  Shared Secret:** All services and the API Gateway share the same JWT\_SECRET for token verification.
* **●  Role-Based Access:** Currently, the system assumes basic user roles; enhancements can include roles like admin, editor, etc.

**Data Validation and Sanitization**

* **●  Input Validation:** Services validate incoming data to prevent malformed requests.
* **●  Sanitization:** Inputs are sanitized to protect against injection attacks.

**API Gateway**

The API Gateway simplifies client interactions by:

* **●  Routing:** Directs incoming requests to the correct service based on the URL path.
* **●  Authentication Middleware:** Centralises authentication, ensuring consistent security across services.
* **●  Logging Middleware:** Captures request and response information for monitoring.
* **●  Error Handling:** Provides consistent error responses and handles exceptions gracefully.

**Implementation Details**

* ●  Built with Express and http-proxy-middleware.
* ●  Uses TypeScript for type safety and maintainability.
* ●  Environment variables configure service URLs and security settings.

**Database Schema**

Each microservice uses its own MongoDB database or collection, adhering to the **Database Per Service** pattern.

**User Service**

**● Collection:** users **● Schema:**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Unique identifier (automatically generated) |
| username | String | Unique username, required |
| email | String | Unique email address, required |
| password | String | Hashed password, required |
| createdAt | Date | Timestamp of creation |
| updatedAt | Date | Timestamp of last update |

**Post Service**

**● Collection:** posts **● Schema:**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Unique identifier |
| userId | String | ID of the user who created the post |
| title | String | Title of the post, required |
| content | String | Content of the post, required |
|  |  | Timestamp of creation |
| updatedAt | Date | Timestamp of last update |

**Comment Service**

**● Collection:** comments **● Schema:**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Unique identifier (automatically generated) |
| postId | String | ID of the post being commented on |
| userId | String | ID of the user who made the comment |
| content | String | Content of the comment, required |
| createdAt | Date | Timestamp of creation |
| updatedAt | Date | Timestamp of last update |

**Like Service**

**● Collection:** likes **● Schema:**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| \_id | ObjectId | Unique identifier (automatically generated) |
| postId | String | ID of the liked post |
| userId | String | ID of the user who liked the post |
| createdAt | String | Timestamp of when the like was made |

**API Endpoints**

The API gateway exposes endpoints under the /api/v1 path, routing requests to the respective services.

**User Service**

**Base URI:** http://localhost:3000/api/v1/users  
**Source Code:** https://git.foss.life/axel/ss-assignment-user-service

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Endpoint** | **Status Code** | **Description** |
| POST | /register | 201 | Creates a new User |
| POST | /login | 200 | Allows a User to log in |
| GET | /:userId | 200 | Returns the profile of a user |

**Post Service**

**Base URI:** http://localhost:4000/api/v1/posts  
**Source Code:** https://github.com/vidhy27/SCALABLE\_SERVICES

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Endpoint** | **Status Code** | **Description** |
| POST | / | 201 | Create a new blog post |
| GET | / | 200 | Get all the blog posts in the system |
| GET | /:postId | 200 | Retrieve one single blog post |
| PUT | /:postId | 200 | Update a blog post |
| DELETE | /:postId | 204 | Delete a blog post |

**Comment Service**

**Base URI:** http://localhost:5000/api/v1/comments  
**Source Code:** https://git.foss.life/axel/ss-assignment-comment-service

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Endpoint** | **Status Code** | **Description** |
| POST | / | 201 | Creates a new comment |
| GET | /post/:postId | 200 | Get the comments for a single post |
| DELETE | /:commentId | 204 | Delete a specific comment |

**Likes Service**

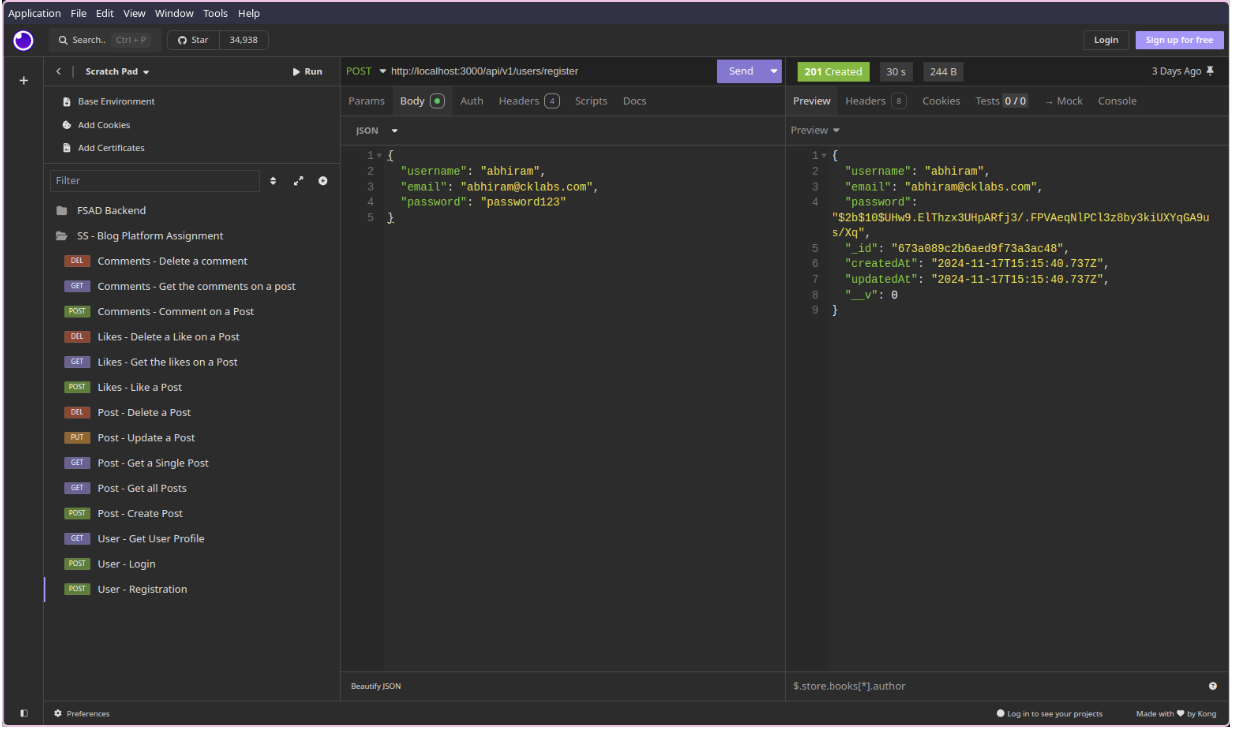
**Base URI:** http://localhost:6000/api/v1/likes  
**Source Code:** https://github.com/freakcap/LikeMicroservice

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Endpoint** | **Status Code** | **Description** |
| GET | /posts/:postId/count | 200 | Get the likes for a post |
| GET | /comments/:commentId/count | 200 | Get the likes for a comment |
| POST | /posts/:postId | 201 | Create a new like for a post |
| POST | /comments/:commentId | 201 | Create a new like for a comment |
| DELETE | /posts/:postId | 204 | Delete a like on a post |
| DELETE | /comments/:commentId | 204 | Delete a like on a comment |

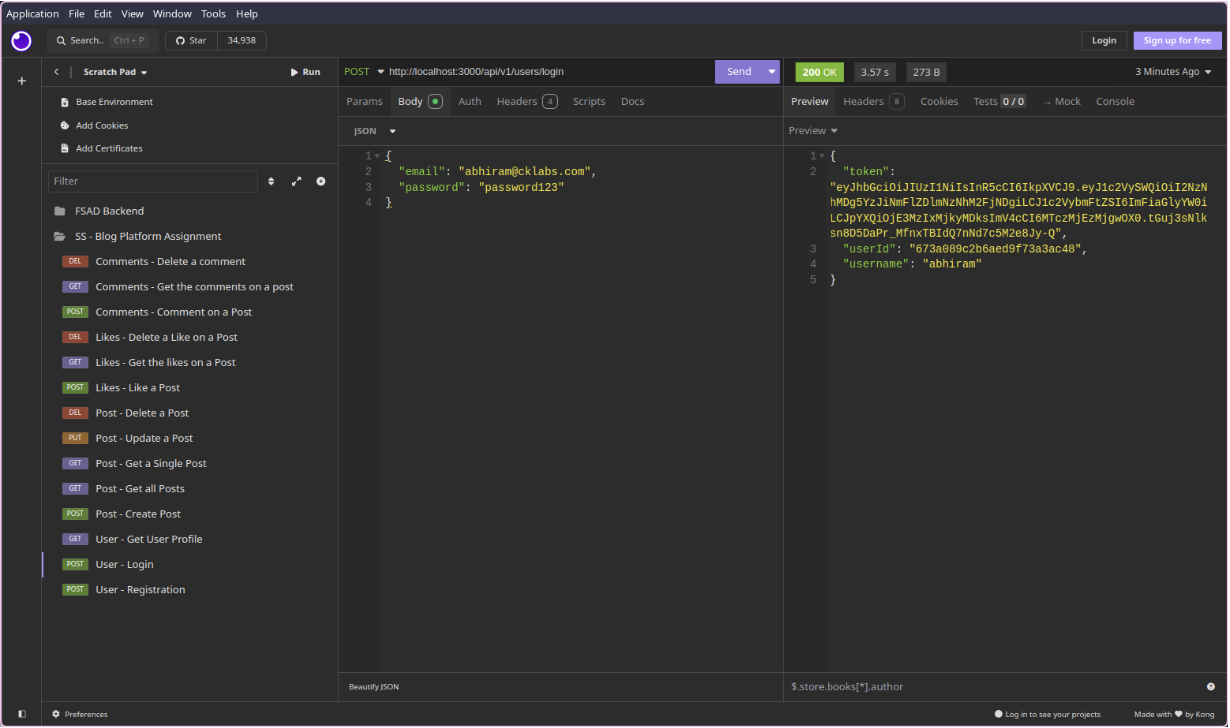
**Insomnia Screenshots**

Insomnia is a tool like Postman that helps us develop and debug APIs. You can use Insomnia to make HTTP requests and observe the responses.

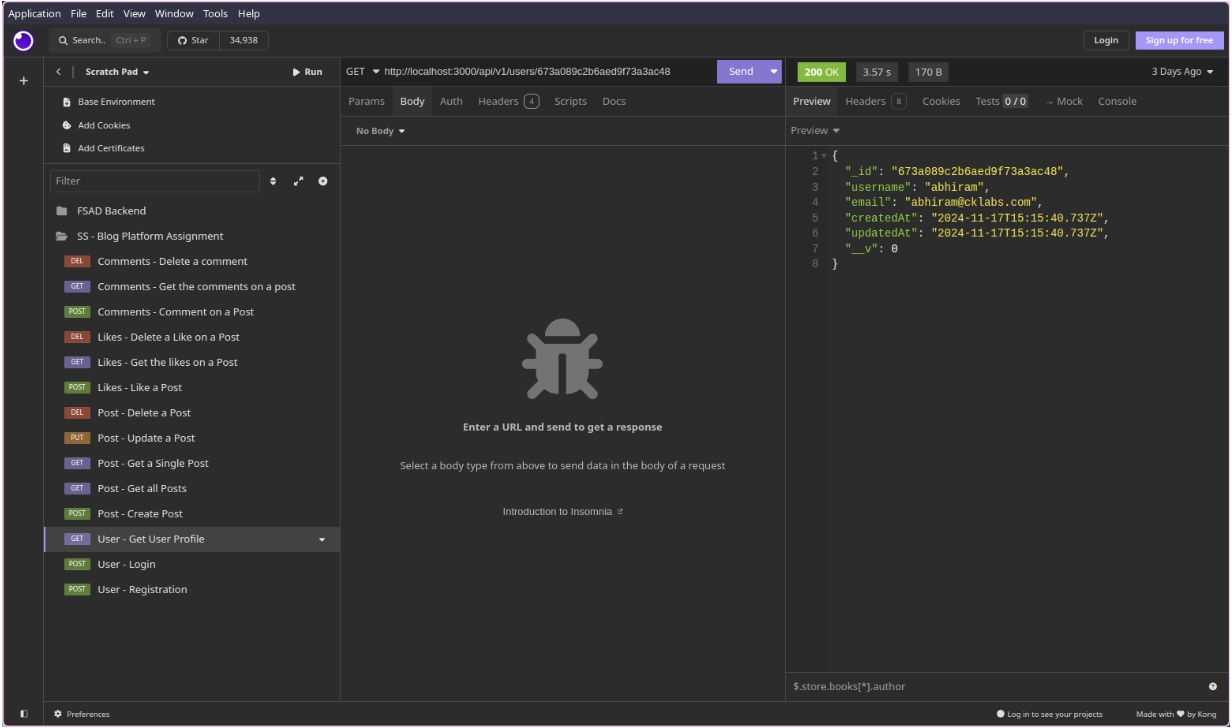
**1. UserRegistration**



**2. UserLogin**



**3. GetaUser’sProfile**



**4. CreateanewPost**

A screenshot of a computer

Description automatically generated

**5. GetALLPosts**

A screenshot of a computer

Description automatically generated

**6. GetaSinglePost**

A screenshot of a computer

Description automatically generated

**7. DeleteaPost**

A screenshot of a computer

Description automatically generated

**8. UpdateaPost**

A screenshot of a computer

Description automatically generated

**9. LikeaPost**

A screenshot of a computer

Description automatically generated

**10. Get the likes on a Post**

A screenshot of a computer

Description automatically generated

**11. Unlike a Post**

A screenshot of a computer

Description automatically generated

**12. Comment on a Post**

A screenshot of a computer

Description automatically generated

**13. Delete a Comment**

A screenshot of a computer

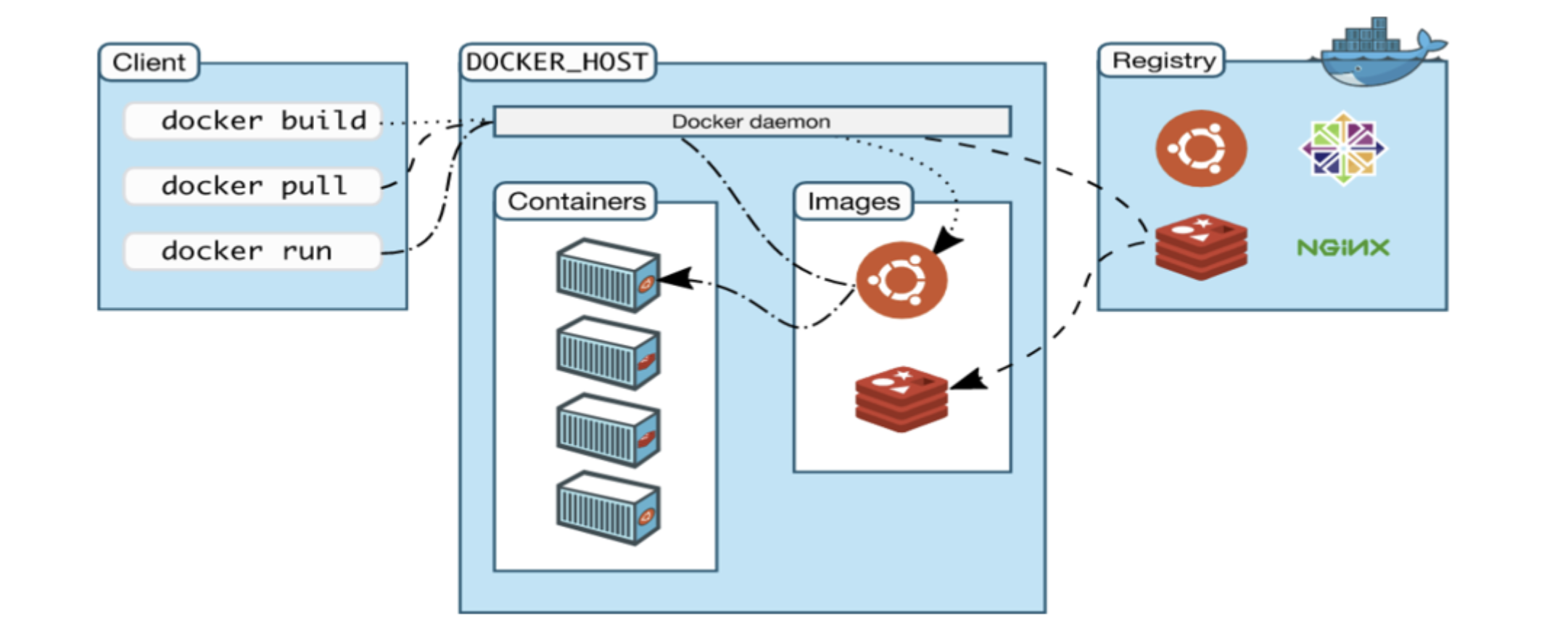
Description automatically generated

A screenshot of a computer

Description automatically generated

**Docker Desktop: Integrated User & Post service:**

Docker is a platform that enables quick application development through containers. Using containers, developers can package up an application with all its dependencies and ship it out as one package. This makes deploying and managing applications much more accessible, especially in a cloud-based or microservices-based architecture.

****

**How Docker Desktop Starts Applications**

1. **Initialization:**
   * Start Docker Desktop: Launch the Docker Desktop application. It sets up the Docker engine and its associated services.
   * Environment Setup: The Docker daemon (server) starts running in the background. It listens for Docker CLI commands from the user.
2. **Application Deployment:**
   * **Pull or Build Images:**
     + Use docker pull to fetch pre-built images from Docker Hub or private registries.
     + Use docker build to create a custom image based on a Dockerfile.
   * **Run Containers:**
     + Use docker run or docker-compose up to create and start containers from the specified images.
     + Containers are lightweight, standalone, and isolated instances of applications.
3. **Application Execution:**
   * **Containerized Execution:**
     + The application runs inside a container with its dependencies and environment pre-configured.
     + The container maps ports, file systems, and networks as defined in the configuration (e.g., Dockerfile or docker-compose.yml).
4. **Networking:**
   * Containers are connected to Docker's networking bridge by default.
   * They communicate with each other or the host machine using ports, specified in configurations.
5. **Orchestration (Optional):**
   * **Docker Compose:**
     + If multiple services (e.g., frontend, backend, database) are involved, docker-compose.yml orchestrates them.
     + A single command (docker-compose up) spins up multiple containers with shared networks and dependencies.
6. **Monitoring and Logging:**
   * Docker Desktop provides a UI to view running containers, logs, and resource usage.
   * Logs can also be accessed via CLI (docker logs <container\_id>).

**Flow of Execution in Docker Desktop**

1. **Build/Prepare the Application:**
   * Write application code (e.g., a Node.js app).
   * Create a Dockerfile that defines how to build the application's image.
   * (Optional) Define services and relationships in docker-compose.yml.
2. **Run Commands:**
   * CLI commands like:
     + docker build -t app-name . (build the image).
     + docker run -p 3000:3000 app-name (run the app in a container).
3. **Container Initialization:**
   * Docker uses the image to start a container.
   * It executes the entrypoint or command defined in the Dockerfile.
4. **Application Serves Requests:**
   * The app listens on the exposed port (e.g., 3000).
   * Requests are routed from the host machine to the containerized application via port mappings.
5. **Persistent Services (if using Compose):**
   * Docker Compose brings up related services in the correct order (e.g., the database before the app).
6. Application Lifecycle Management:
   * Monitor running containers via Docker Desktop's GUI or CLI.
   * Stop/restart containers using docker stop/docker restart.

**Visualized Flow**

1. User → Sends a docker run or docker-compose up command.
2. Docker CLI → Sends the request to the Docker daemon.
3. Docker Daemon:
   * Creates a container from an image.
   * Configures networking, storage, and isolation for the container.
4. Application:
   * Executes inside the container as defined in the Dockerfile.
   * Responds to incoming requests via exposed ports.

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated