27 August 2024 14:59

User Management Service Documentation

High-Level Architecture Overview

1.1 Architecture Components

- **Microservices**: User Management Service, Audit Service, Role Service should be separate microservices, allowing independent scaling, deployment, and management.
  - **User Service**: Handles user management.
  - **Role Service**: Handles role management.
- **Spring Boot**: Both microservices will be built using Spring Boot, a framework that simplifies microservice development.
- **Database**:
  - **Amazon RDS** (Relational Database Service) or **Amazon Aurora**: A highly available, scalable, and fully managed relational database service.
- **Service Discovery**:
  - **Eureka/Consul**: For service discovery, enabling microservices to locate each other dynamically.
- **API Gateway**:
  - **Spring Cloud Gateway**: For routing requests to the appropriate microservice and handling cross-cutting concerns like authentication and rate limiting.
- **Docker**: Containerizes each microservice, ensuring consistent deployment across different environments.
- **Kubernetes (K8s)**:
  - Orchestrates the deployment, scaling, and management of Docker containers.
  - **EKS (Elastic Kubernetes Service)**: A managed Kubernetes service provided by AWS.
- **Load Balancer**:
  - **Elastic Load Balancer (ELB)**: Distributes incoming application traffic across multiple targets, such as EC2 instances, ensuring high availability.
- **CI/CD Pipeline**:
  - **Jenkins/GitLab CI**: Automates building, testing, and deploying the microservices.
  - **Amazon CodePipeline**: An AWS CI/CD service that integrates with CodeBuild, CodeDeploy, and CodeCommit.
- **Monitoring and Logging**:
  - **Prometheus and Grafana**: For monitoring and alerting.
  - **ELK Stack (Elasticsearch, Logstash, Kibana)** or **AWS CloudWatch**: For centralized logging and analysis.
- **AWS S3**: For storing static files, logs, and backups.
- **AWS IAM**: Manages access control and permissions.

2. Running the API in Production

2.1 High Availability and Resiliency

- **Multi-AZ Deployment**: Deploy services across multiple AWS Availability Zones to ensure high availability. In case one AZ fails, the application continues running in another.
- **Auto Scaling**: Use Kubernetes Horizontal Pod Autoscaler (HPA) to automatically scale microservices based on CPU/memory usage.
- **Database Replication**: Use Amazon RDS Multi-AZ deployments with read replicas to ensure high availability and quick recovery.
- **Load Balancing**: Distribute traffic using AWS ELB to route requests to healthy instances only.
- **CI/CD**: Automated deployment pipelines ensure that the latest stable versions of the microservices are deployed with minimal downtime.
- **Rolling Updates**: Kubernetes performs rolling updates to minimize downtime and ensure that the application remains available during upgrades.
- **Circuit Breaker Pattern**: Implement with Spring Cloud Circuit Breaker to prevent cascading failures in case of service unavailability.

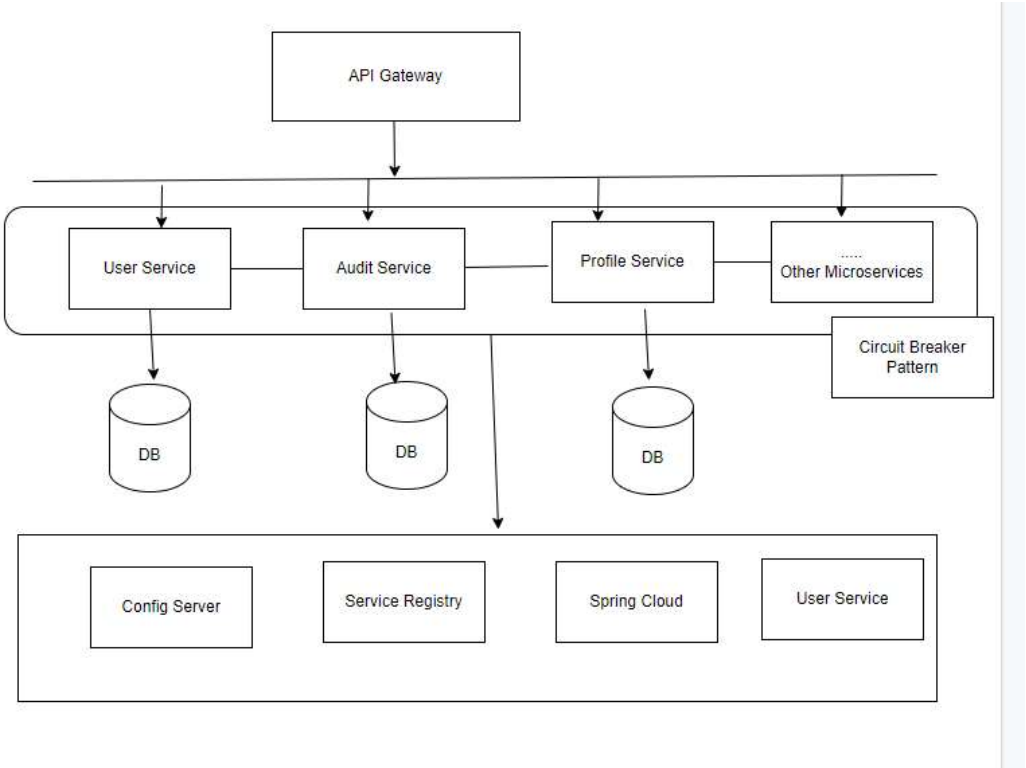
2.2 Security

- **Authentication and Authorization**: Use OAuth2/JWT for securing APIs. Leverage Spring Security to enforce roles and permissions.
- **Encryption**: Data at rest (using AWS KMS for encryption) and in transit (using HTTPS) should be encrypted.
- **Network Security**: Use AWS Security Groups and Network ACLs to secure network traffic.

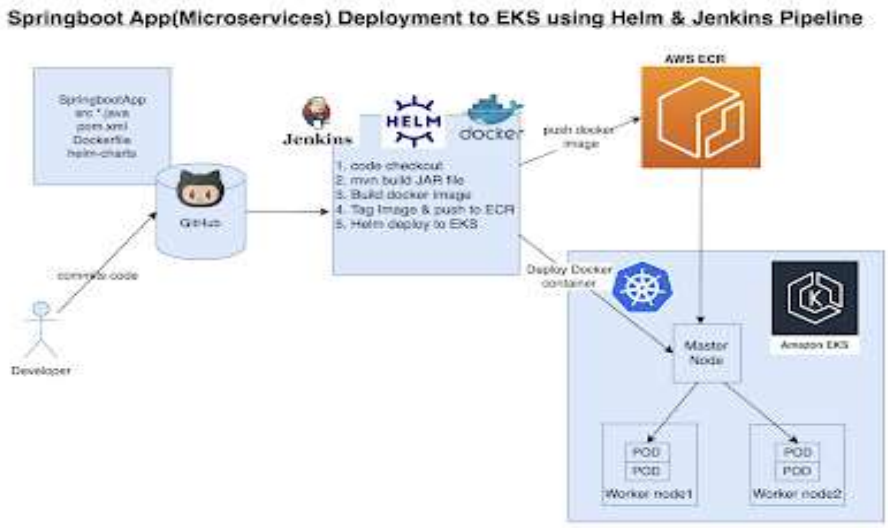
2.3 Observability

- **Monitoring**: Use Prometheus to monitor metrics like response times, error rates, CPU, and memory usage. Set up alerts for thresholds.
- **Centralized Logging**: Use AWS CloudWatch or the ELK stack to aggregate logs from all microservices for easier debugging and analysis.
- **Tracing**: Implement distributed tracing using Spring Cloud Sleuth and Zipkin to track requests across microservices.

High Level Diagram



Deployment Diagram Using EKS for High Availability and Resiliency



User Management Endpoint Detail

Endpoint	HTTP Method	Request Body	Response	Response Status

/api/users/create	POST	{
		"username": "newUser2",
		"firstName": "John",
		"lastName":"singh",
		"email": "newUser2@example.com",
		"createdBy": "ADMIN",
		"updatedBy": "ADMIN",
		"roles": ["USER"]
		}
			{
			"id": 2,
			"username": "newUser2",
			"firstName": "John",
			"lastName": "singh",
			"email": "newUser2
			@example.com",
			"roles": [
			"USER"
			],
			"approved": false
			}
				201 Created

/{id}/assign-roles	POST	["ADMIN","USER"]	{ "id": 1, "username": "newUser", "firstName": "pankaj",	200 OK

			"lastName": "singh", "email": "newuser@example.com", "roles": ["ADMIN", "USER"], "approved": true }	
/api/users/1/approve/ApprovedBy/newUser	POST	None	UserDto (same as above)	200 OK

/{id}	PUT	{
		"username": "newUser2",
		"firstName": "John",
		"lastName":"Spence",
		"email": "newUser2@example.com",
		"createdBy": "ADMIN",
		"updatedBy": "ADMIN",
		"roles": ["USER"]
		}
			{
			"id": 2,
			"username": "newUser2",
			"firstName": "John",
			"lastName": "Spence",
			"email": "newUser2
			@example.com",
			"roles": [
			"ADMIN",
			"USER"
			],
			"approved": true
			}
				200 OK
				404 Not Found

/{id}/removeBy/{Removed-By}	DELETE	None	None	204 No Content

/api/users/getAllUsers	GET	None	[
			{
			"id": 1,
			"username":
			"newUser",
			"firstName":
			"pankaj",
			"lastName": "singh",
			"email":
			"newuser@example
			.com",
			"roles": [
			"ADMIN",
			"USER"
			],
			"approved": true
			}
			]
				200 OK

api/users/newUser/profile	GET	None	{
			"id": 1,
			"username": "newUser",
			"firstName": "pankaj",
			"lastName": "singh",
			"email":
			"newuser@example.com",
			"roles": [
			"ADMIN",
			"USER"
			],
			"approved": false
			}
				200 OK

Role Management Endpoint Details

Endpoint	HTTP Method	Request Body	Response	Response Status

/api/roles/create-multiple	POST	[
		{"name": "ADMIN"},
		{"name": "USER"}
		]
			[
			{
			"id": 1,
			"name": "ADMIN"
			},
			{
			"id": 2,
			"name": "USER"
			}
			]
				201 Created

Open OPI Documentation

```
openapi: 3.0.0
info:
  title: User Management API
  description: API for managing users, including creating, updating, and deleting users, as well as assigning roles and viewing profiles.
  version: 1.0.0
servers:
```

```
- url: /api/users
  description: User Management API server

paths:
  /create:
    post:
      summary: Create a new user
      description: Create a new user with the provided details.
      tags:
        - Users
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CreateUserDto'
      responses:
        '201':
          description: User created successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/UserDto'
        '403':
          description: Access denied

  /{id}/assign-roles:
    post:
      summary: Assign roles to a user
      description: Assign specified roles to a user by user ID.
      tags:
        - Users
      parameters:
        - name: id
          in: path
          required: true
          description: ID of the user
          schema:
            type: integer
            format: int64
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
      responses:
        '200':
          description: Roles assigned successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/UserDto'
        '403':
          description: Access denied

  /{id}/approve/ApprovedBy/{ApprovedBy}:
    post:
      summary: Approve a user
      description: Approve a user by user ID and specify who approved.
      tags:
        - Users
      parameters:
        - name: id
          in: path
          required: true
          description: ID of the user
          schema:
            type: integer
            format: int64
        - name: ApprovedBy
          in: path
          required: true
          description: Username of the person approving the user
          schema:
            type: string
```

```
    responses:
      '200':
        description: User approved successfully
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/UserDto'
      '403':
        description: Access denied

  /{id}:
    put:
      summary: Update a user
      description: Update the details of an existing user by user ID.
      tags:
        - Users
      parameters:
        - name: id
          in: path
          required: true
          description: ID of the user
          schema:
            type: integer
            format: int64
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CreateUserDto'
      responses:
        '200':
          description: User updated successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/UserDto'
        '403':
          description: Access denied
        '404':
          description: User not found

  /{id}/removeBy/{Removed-By}:
    delete:
      summary: Remove a user
      description: Remove a user by user ID and specify who removed the user.
      tags:
        - Users
      parameters:
        - name: id
          in: path
          required: true
          description: ID of the user
          schema:
            type: integer
            format: int64
        - name: Removed-By
          in: path
          required: true
          description: Username of the person removing the user
          schema:
            type: string
      responses:
        '204':
          description: User removed successfully
        '403':
          description: Access denied

  /getAllUsers:
    get:
      summary: List all users
      description: Retrieve a list of all users.
      tags:
        - Users
      responses:
        '200':
          description: Users retrieved successfully
          content:
```

```
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/UserDto'
      '403':
        description: Access denied

  /{Username}/profile:
    get:
      summary: View user profile
      description: Retrieve the profile of a user by their username.
      tags:
        - Users
      parameters:
        - name: Username
          in: path
          required: true
          description: Username of the user
          schema:
            type: string
      responses:
        '200':
          description: User profile retrieved successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/UserDto'
        '403':
          description: Access denied

components:
  schemas:
    CreateUserDto:
      type: object
      properties:
        username:
          type: string
        firstName:
          type: string
        lastName:
          type: string
        email:
          type: string
          format: email
        roles:
          type: array
          items:
            type: string
        createdBy:
          type: string
        updatedBy:
          type: string
      required:
        - username
        - firstName
        - lastName
        - email

    UserDto:
      type: object
      properties:
        id:
          type: integer
          format: int64
        username:
          type: string
        firstName:
          type: string
        lastName:
          type: string
        email:
          type: string
          format: email
        roles:
          type: array
          items:
            type: string
```

```
  approved:
    type: boolean
required:
  - id
  - username
  - firstName
  - lastName
  - email
```