For designing Authorization Microservice we need to understand the problem statement first which are given in the problem statement file-

Section 1: Problem Understanding & Requirements

Multi-Application Support

- The service must be universal, catering to various applications (web, mobile, and admin panel).
- APIs should provide uniform functionality regardless of the platform.
- Centralize authentication and authorization, ensuring consistent and secure user management.

User Authentication

- Verify user identity with secure credentials (username, password, or tokens).
- Support SSO (Single Sign-On) across applications.
- Use token-based mechanisms like **OAuth 2.0** or **JWT** to manage authentication.

Role-Based Access Control (RBAC)

- Implement RBAC to assign users specific roles (e.g., admin, user, guest).
- Associate roles with permissions to define what actions (read, write, delete) they can perform on resources.
- Ensure flexibility to dynamically manage roles and permissions.

Token-Based Authorization

- Employ **JWT** tokens for stateless authorization.
- Include user information, roles, and permissions within the token payload.
- Validate tokens for each request, ensuring the user has access to the requested resource.

Auditing and Logging

- Log every authentication, authorization, and permission check for compliance and debugging.
- Include user details, requested actions, results, and timestamps in logs.

Scalability

- Design the service to handle high user volume and request loads by scaling horizontally.
- Use distributed caching, load balancing, and stateless architecture.

Key Design Considerations

To ensure the service is secure, scalable, and extensible, we have to consider the following:

1. Security

- **Encryption**: Use HTTPS for all communications to protect data in transit. Encrypt sensitive data like passwords and tokens in the database.
- **JWT Validation**: Verify tokens with signing algorithms to prevent tampering.
- **Rate Limiting**: Implement rate limits to prevent abuse of APIs.
- **Access Control**: Enforce strict permission checks for each API request.

2. Scalability

- **Horizontal Scaling**: Use stateless APIs so multiple service instances can handle requests concurrently.
- **Caching**: Store frequently accessed data (e.g., tokens, permissions) in Redis to minimize database load.
- **Database Optimization**: Use indexing and query optimization for faster role and permission lookups.

3. Extensibility

- **Dynamic Role Management**: Allow admin users to add or modify roles and permissions without code changes.
- **Plug-and-Play APIs**: Provide APIs that integrate seamlessly with new applications.
- **Protocol Flexibility**: Add support for OAuth 2.0, SAML, or custom protocols as needed.

4. Auditing

- Store detailed logs, including:
    - User ID and role
    - Resource accessed
    - Action performed (e.g., read, write)
    - Result (e.g., success, failure)
    - Timestamp and IP address

5. Performance

- **Load Balancer**: Distribute requests across multiple instances for better performance.
- **Asynchronous Processing**: Use queues for non-critical operations like logging to reduce API response time.

Technologies

Backend Technology

PHP with frameworks like **Laravel** or **Lumen** to build secure and scalable microservices.

Authentication Protocols

- **OAuth 2.0**: For token issuance, management, and validation.
- **JWT**: Stateless tokens containing encoded user data, roles, and permissions.

Database

- **MySQL**: To store users, roles, and permissions.
- **Redis**: For caching tokens and permissions.

Logging and Monitoring

- **Monolog**: For structured logging.
- **Elastic Stack (ELK)**: For centralized log storage and analysis.

API Standards

- **RESTful APIs**: For simplicity and integration.
- **Swagger/OpenAPI**: For documentation and testing.

Infrastructure

- **Docker**: Containerize the service for easy deployment.
- **Kubernetes**: Manage scalability and deployment.
- **NGINX**: Act as a reverse proxy and load balancer.

Detailed Workflow

1. Authentication Flow

1. A user logs in using their credentials.
2. The service validates credentials and issues a JWT token.
3. The token includes:
    - User ID
    - Roles
    - Permissions
    - Expiration time

2. Authorization Flow

1. A user requests a resource, attaching the JWT token in the header.
2. The service validates the token, extracts roles and permissions, and checks if the user is authorized.
3. If valid, the service grants access; otherwise, it denies the request.

3. Role Management Flow

1. Admins create roles and assign permissions via API endpoints.
2. Users are assigned roles dynamically.
3. Updates are cached to reflect changes instantly.