System Design Dat 6: API Gateway Design

🔷 What Is an API Gateway?

An **API Gateway** is a **central entry point** for client requests in a **microservices architecture**. It acts as a reverse proxy that routes requests to backend services and handles cross-cutting concerns like:

- Authentication
- Rate Limiting
- Routing
- Caching
- Logging
- TLS Termination

🕰️ Evolution of Architecture

1. **2000s – Monoliths**
    - Clients talked to a single application.
    - Simple and easy to deploy, but not scalable.
2. **2010–2012 – Microservices Era**
    - Applications were split into many services.
    - Clients either needed to know each service's URL or use a proxy service—both messy solutions.
3. **2013–2014 – API Gateway Emerges**
    - Introduced a thin centralized layer to handle all requests.
    - Clients only need to know **one endpoint**.
    - The gateway routes the request to the right microservice.

🧱 Why Use an API Gateway?

Instead of duplicating the same code in every microservice, the API Gateway handles **common infrastructure responsibilities**:

🌐 Routing

- Routes incoming requests to the correct backend service using path-based or host-based rules.

> Java Context: Spring Cloud Gateway provides configurable routing using YAML or Java DSL.
>
> **Nginx/Ingress:** Nginx or Kubernetes Ingress controllers can act as layer-7 routers.

🔒 Authentication & Authorization

- Validates tokens (e.g., OAuth2 JWT) and restricts access based on roles/policies.

> Java: Spring Security filters can be applied at the gateway level.
>
> **Nginx:** Can integrate with external OAuth2 proxy or Lua scripts.
>
> **Ingress:** Can use annotations and plugins like OIDC for auth.

🚦 Rate Limiting

- Protects services from abuse and ensures fair usage.

> Java: Spring Cloud Gateway supports Redis-based rate limiting via RequestRateLimiter filter.
>
> **Nginx:** Supports rate limiting via built-in modules (`limit_req`).
>
> **Ingress:** Ingress-NGINX supports rate limiting with annotations.

🔄 Request/Response Transformation

- Converts protocols (e.g., gRPC → REST) or rewrites headers and paths.

> Useful for interoperability and versioning strategies.

🔐 TLS Termination

- Offloads HTTPS decryption from microservices.

> Done at API Gateway, Nginx, or Ingress controller.

📜 Other Features

- TLS Termination
- Logging & Monitoring
- Response Transformation (e.g., gRPC → REST)
- Caching of frequent responses

⚙️ Core Responsibilities of an API Gateway

1. **Request Validation**
    - Ensures requests have valid structure, headers, tokens.
2. **Middleware Execution**
    - Handles authentication, authorization, rate limiting, logging, etc.
3. **Routing to Services**
    - Uses a configuration to map URL paths to backend services.
4. **Response Transformation**
    - Converts protocol-specific responses (like gRPC) to RESTful JSON if needed.

🛠️ API Gateway Implementations

☁️ Managed Services

- **AWS API Gateway**

- **Azure API Management**
- **Google Cloud API Gateway**

🧩 Open Source & Self-Managed

- **Spring Cloud Gateway** (Java-native, ideal for Spring Boot ecosystems)
- **Kong**, **Tyk**, **Express Gateway**
- **Nginx** (high-performance reverse proxy with custom configuration)
- **Kubernetes Ingress Controller** (like NGINX Ingress or Istio Gateway)

🚪 NGINX and Kubernetes Ingress Controllers

🔧 NGINX

- Acts as a high-performance reverse proxy and can function as an API gateway.
- Supports:
  - Rate limiting
  - SSL termination
  - Header rewriting
  - Load balancing
  - Auth integration via Lua scripts or external tools

```
nginx
CopyEdit
location /api/ {
  proxy_pass http://backend_service;
  limit_req zone=api_zone burst=10 nodelay;
  auth_request /auth;
}
```

☸️ Kubernetes Ingress + NGINX

- Ingress is the K8s-native way to define API gateway rules.

- Works with Ingress Controllers like **NGINX**, **Istio**, or **Traefik**.
- Allows declarative config via annotations:

```
yaml
CopyEdit
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/limit-connections: "1"
    nginx.ingress.kubernetes.io/auth-url: "https://auth.myapp.com"
spec:
  rules:
    - http:
        paths:
          - path: /api/users
            pathType: Prefix
            backend:
              service:
                name: user-service
                port:
                  number: 80
```

🧠 System Design Interview Tip

> In system design interviews:

- Include an **API Gateway** by default.
- Mention it handles:
  - Routing
  - Auth
  - Rate limiting
  - Response transformation

- Say **"this is implemented via Spring Cloud Gateway or NGINX/Ingress"**, and **move on quickly**.

Spending too much time on the gateway is discouraged—it’s considered infrastructure hygiene.

👨‍💻 Java + Spring Example

```
yaml
CopyEdit
# application.yml for Spring Cloud Gateway
spring:
  cloud:
    gateway:
      default-filters:
        - AddRequestHeader=X-Request-Source, Gateway
        - RequestRateLimiter=replenishRate=10,burstCapacity=20
      routes:
        - id: user-service
          uri: http://localhost:8081
          predicates:
            - Path=/users/**
          filters:
            - JwtAuthenticationFilter
```

With Spring Security:

```
java
CopyEdit
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
      .authorizeRequests()
      .antMatchers("/users/**").hasRole("USER")
      .anyRequest().authenticated()
      .and()
      .oauth2ResourceServer()
      .jwt();
  }
}
```