6. Design a Content Delivery Network (CDN)

Step 1: Outline Use Cases and Constraints

Use Cases

We'll scope the problem to handle only the following use cases:

1. **Content Caching**: Store and serve static and dynamic content efficiently.
2. **Load Balancing**: Distribute traffic among multiple edge servers to optimize performance.
3. **Geo-Distributed Content Delivery**: Serve content from locations closer to users.
4. **Cache Expiration and Invalidation**: Ensure content remains fresh and updated.
5. **Security and DDoS Protection**: Implement security features like TLS, access control, and DDoS mitigation.
6. **Logging and Analytics**: Track user requests, cache hits/misses, and performance metrics.
7. **High Availability**: Ensure 99.99% uptime with failover mechanisms.

Out of Scope

- Custom user authentication systems.
- Complete logging and monitoring dashboards.
- Managing real-time dynamic content.

Constraints and Assumptions

- Global user base with traffic distributed unevenly.
- 100 million requests per day (~1,157 requests per second).
- Edge nodes located in multiple geographic regions.
- Requests should be served in under 50ms.
- Content stored primarily as static assets (~10 PB total storage).

- Cache hit ratio should be 90%.

Step 2: Create a High-Level Design

Components

1. Client (User Request) – Browser or application requesting content.
2. DNS and Load Balancer – Directs requests to the nearest edge server.
3. Edge Servers (PoPs - Points of Presence) – Serve cached content to users.
4. Origin Servers – The original source of content when a cache miss occurs.
5. Cache Storage Layer – Stores frequently accessed content using LRU policies.
6. Security Layer – Handles TLS termination, DDoS protection, and rate limiting.
7. Analytics & Monitoring – Logs request details, cache efficiency, and network performance.

Step 3: Design Core Components

Use Case: Handling Content Requests

- User requests content.
- DNS resolves to the nearest edge server.
- Edge server checks cache:
- If cached: Serve directly.
- If not cached: Fetch from origin, store in cache, and serve.

Cache Management

- LRU (Least Recently Used) eviction policy to remove old content.
- Time-to-Live (TTL) settings for content freshness.
- Cache invalidation using a pull model (stale-while-revalidate).

Load Balancing

- Distribute load across PoPs using GeoDNS.
- Use Round-robin and weighted load balancing to avoid overloading a single PoP.

Security & DDoS Protection

- Implement rate limiting at the edge.
- Use WAF (Web Application Firewall) for filtering malicious traffic.
- Implement TLS termination at edge locations.

Data Model

Field	Type	Description
URL Hash	String	Hash of requested URL
Content Path	String	Path to cached content
Expiration Time	Timestamp	Time when cache expires
Last Accessed	Timestamp	Last time content was served
Size	Integer	Size of Content in bytes

API Endpoints

Request Content

GET /content/{resource_id}

Invalidate Cache

DELETE /cache/{resource_id}

Cache Lookup Algorithm (Python)

```
def get_content(resource_id):
    if cache.exists(resource_id):
        return cache.get(resource_id)
    else:
        content = fetch_from_origin(resource_id)
        cache.set(resource_id, content, ttl=3600)
        return content
```

1. Dependencies (Maven)

You'll need these dependencies in your pom.xml:

```
<dependencies>
    <!-- Spring Boot Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- Spring Boot Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- PostgreSQL Driver -->
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
    </dependency>

    <!-- AWS SDK for S3 -->
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>s3</artifactId>
        <version>2.17.103</version>
    </dependency>

    <!-- Lombok for Reducing Boilerplate Code -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

2. Short Link Generation (MD5 + Base62 Encoding)

This method generates a unique short link using MD5 hash and Base62 encoding.

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

public class URLShortener {
```

```
    private static final String BASE62 =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

    public static String generateShortLink(String input) {
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            byte[] digest = md.digest(input.getBytes());
            BigInteger bigInt = new BigInteger(1, digest);
            String base62Hash = encodeBase62(bigInt);
            return base62Hash.substring(0, 7); // Take first 7 chars
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Error generating hash", e);
        }
    }

    private static String encodeBase62(BigInteger number) {
        StringBuilder sb = new StringBuilder();
        while (number.compareTo(BigInteger.ZERO) > 0) {
            sb.append(BASE62.charAt(number.mod(BigInteger.valueOf(62)).intValue()));
            number = number.divide(BigInteger.valueOf(62));
        }
        return sb.reverse().toString();
    }
}
```

3. Content Storage (AWS S3)

This service uploads and retrieves content from Amazon S3.

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.*;

import java.nio.charset.StandardCharsets;

@Service
public class StorageService {

    private final S3Client s3Client;

    @Value("${aws.s3.bucket}")
    private String bucketName;

    public StorageService() {
        this.s3Client = S3Client.builder()
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();
```

```
    }

    public String uploadContent(String shortLink, String content) {
        PutObjectRequest putRequest = PutObjectRequest.builder()
                .bucket(bucketName)
                .key(shortLink)
                .build();

        s3Client.putObject(putRequest, RequestBody.fromString(content,
StandardCharsets.UTF_8));

        return shortLink;
    }

    public String getContent(String shortLink) {
        GetObjectRequest getRequest = GetObjectRequest.builder()
                .bucket(bucketName)
                .key(shortLink)
                .build();

        return s3Client.getObjectAsBytes(getRequest).asUtf8String();
    }
}
```

4. Database Entity (JPA)

We use Spring Data JPA to store the metadata of each paste.

```
import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDateTime;

@Entity
@Table(name = "pastes")
@Getter @Setter @NoArgsConstructor @AllArgsConstructor
public class Paste {

    @Id
    private String shortLink;  // Primary Key

    @Column(nullable = false)
    private String pastePath;  // S3 Key

    @Column(nullable = false)
    private LocalDateTime createdAt;

    private Integer expirationMinutes;  // Optional expiration
}
```

5. REST API Controller

Handles creating and retrieving pastes.

```
import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.Autowired;

import java.time.LocalDateTime;
import java.util.Optional;

@RestController
@RequestMapping("/api/v1/paste")
public class PasteController {

    @Autowired
    private PasteRepository pasteRepository;

    @Autowired
    private StorageService storageService;

    @PostMapping
    public Paste createPaste(@RequestBody PasteRequest request) {
        String shortLink = URLShortener.generateShortLink(request.getContent() +
System.currentTimeMillis());
        storageService.uploadContent(shortLink, request.getContent());

        Paste paste = new Paste(shortLink, shortLink, LocalDateTime.now(),
request.getExpirationMinutes());
        return pasteRepository.save(paste);
    }

    @GetMapping("/{shortLink}")
    public String getPaste(@PathVariable String shortLink) {
        Optional<Paste> paste = pasteRepository.findById(shortLink);
        return paste.map(p -> storageService.getContent(p.getPastePath())).orElse("Paste not
found");
    }
}
```

6. Tracking Analytics

A simple Redis-based analytics tracker for storing page visits.

```
import org.springframework.stereotype.Service;
import redis.clients.jedis.Jedis;

@Service
public class AnalyticsService {

    private final Jedis jedis;

    public AnalyticsService() {
        this.jedis = new Jedis("localhost", 6379);
    }

    public void trackVisit(String shortLink) {
        jedis.incr("visits:" + shortLink);
    }

    public int getVisitCount(String shortLink) {
        String count = jedis.get("visits:" + shortLink);
        return count != null ? Integer.parseInt(count) : 0;
    }
}
```

7. Deleting Expired Content

A cron job that runs periodically to remove expired pastes.

```
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import java.time.LocalDateTime;
import java.util.List;

@Component
public class ExpiredPasteCleanupJob {

    private final PasteRepository pasteRepository;
    private final StorageService storageService;

    public ExpiredPasteCleanupJob(PasteRepository pasteRepository, StorageService
storageService) {
        this.pasteRepository = pasteRepository;
```

```
        this.storageService = storageService;
    }

    @Scheduled(fixedRate = 3600000)  // Run every hour
    public void cleanExpiredPastes() {
        List<Paste> expiredPastes = pasteRepository.findExpired(LocalDateTime.now());
        expiredPastes.forEach(paste -> {
            storageService.deleteContent(paste.getPastePath());
            pasteRepository.delete(paste);
        });
    }
}
```

8. Repository Interface

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import java.time.LocalDateTime;
import java.util.List;

public interface PasteRepository extends JpaRepository<Paste, String> {

    @Query("SELECT p FROM Paste p WHERE p.expirationMinutes IS NOT NULL AND p.createdAt +
INTERVAL '1 MINUTE' * p.expirationMinutes < ?1")
    List<Paste> findExpired(LocalDateTime now);
}
```

Final Notes

- The system supports short link generation, content storage (AWS S3), retrieval, analytics tracking (Redis), and automatic deletion of expired pastes.
- It is designed using Spring Boot, JPA, Redis, and AWS S3.
- The API allows users to store, retrieve, and track content analytics efficiently.

Step 4: Scale the Design

- **Handling Bottlenecks**

  1. **Scaling Edge Servers**: Use auto-scaling groups for handling peak loads.

  2. **Database Scaling**:

     - Read-heavy workload → Use Read Replicas.

     - High write throughput → Implement Sharding.

  3. **Reducing Latency**:

     - Use Anycast DNS to route users to the nearest PoP.

     - HTTP/3 and QUIC for faster transport.

  4. **Ensuring High Availability**:

     - Use multi-region deployment.

     - Implement failover mechanisms.

- **Integrate a Caching Layer** - Use CloudFront, Nginx, or Redis for caching frequently accessed content
- **Optimize Content Delivery** - Implement gzip compression, HTTP/2, and edge computing
- **Load Balancing** - Use NGINX, HAProxy, or AWS ALB to distribute traffic efficiently.
- **Multi-Region Content Replication** - Store copies of content across multiple locations for faster delivery.
- **Security & Access Control** - Use JWT authentication, Signed URLs, and rate limiting.

By following these steps iteratively, we ensure the CDN remains performant, secure, and scalable while handling global traffic efficiently.