Optimizing Backend API Performance

Presented by:

Garima Agarwal

Application Programmer V at Bank of America

Our Agenda Today

1. Why API Performance Matters
2. Common API Performance Bottlenecks
3. Pagination: Handling Large Datasets Efficiently
4. Asynchronous Logging: Enhancing Performance
5. Caching: Reducing Database Load
6. Payload Compression: Improving Network Efficiency
7. Connection Pooling: Optimizing Database Interactions
8. Key Takeaways

Why API Performance Matters?

1. **Better User Experiences**

   Ensures smooth and responsive applications.

2. **Scalability and Responsiveness**

   Handles increased traffic with efficiency.

3. **More Reliable**

   Results in dependable API performance.

4. **Fewer Resource Utilization**

   Consumes less server power for optimal performance.

5. **Business Impact**

   Drives revenue and boosts customer satisfaction.

Common API Performance Bottlenecks

Slow database queries

Poor indexing, unoptimized joins

High latency

In logging & processing

Large payloads

Uncompressed

Excessive connections

To database

Redundant requests

From client

Lack of caching

No caching mechanisms

Pagination: Handling Large Datasets Efficiently

Why Pagination?

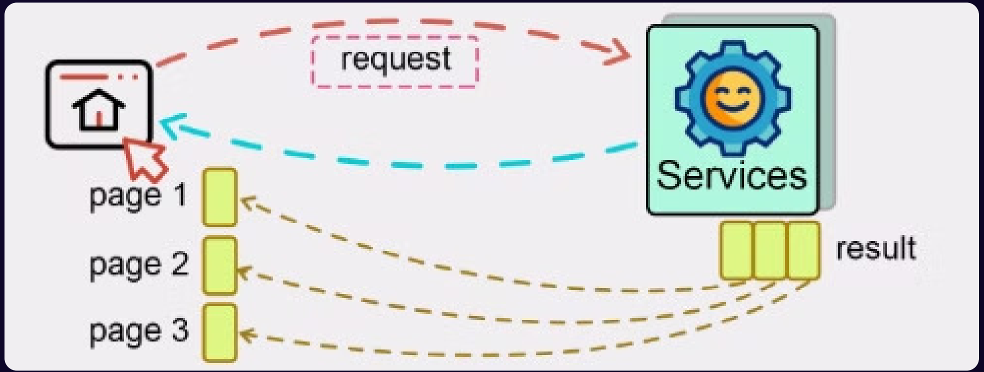
- Avoids large dataset retrievals
- Reducing response times

Techniques:

- Offset-based pagination
- Cursor-based pagination (real-time applications)
- Page-based pagination

Example (Spring Boot Code)

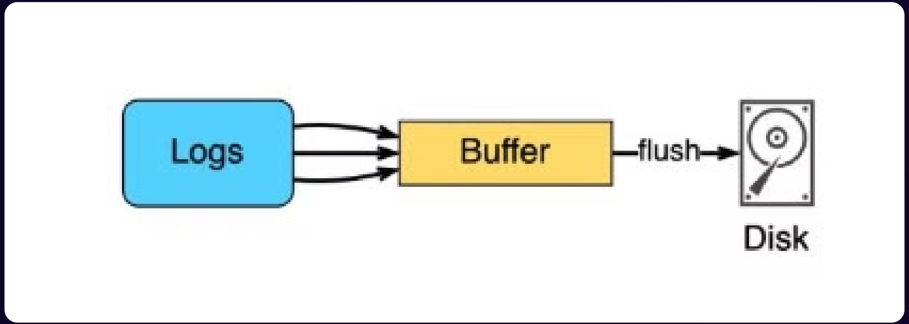
```
@GetMapping("/products")
public Page<Product> getAllProducts(Pageable pageable) {
  return productRepository.findAll(pageable);
}
```



Asynchronous Logging: Enhancing Performance

Problem with Synchronous Logging

- Blocks execution for each log write
- Increases response time



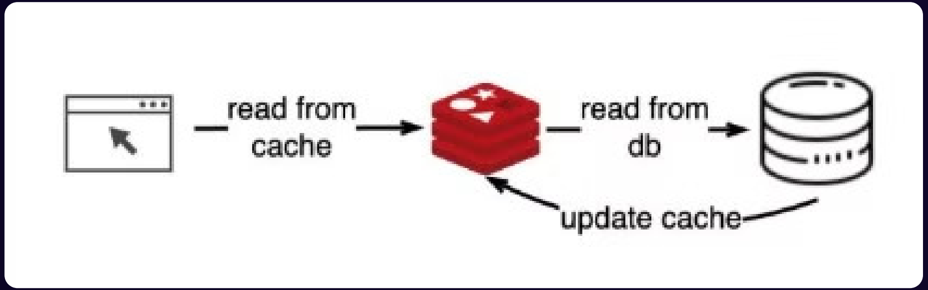
Solution: Use Async Logging

- Logs are written in memory and flushed periodically

Caching: Reducing Database Load

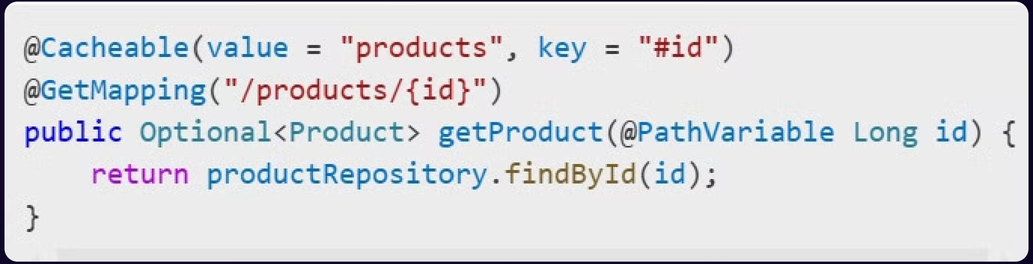
Before Caching

- Each request hits the database
- Slower response times
- Higher database load



After Caching (Redis Example)

- Reduced database hits & speeds up responses
- In-memory data retrieval



```java
@Cacheable(value = "products", key = "#id")
@GetMapping("/products/{id}")
public Optional<Product> getProduct(@PathVariable Long id) {
    return productRepository.findById(id);
}
```

Made with Gamma

Payload Compression: Improving Network Efficiency

Why Compress Payloads?

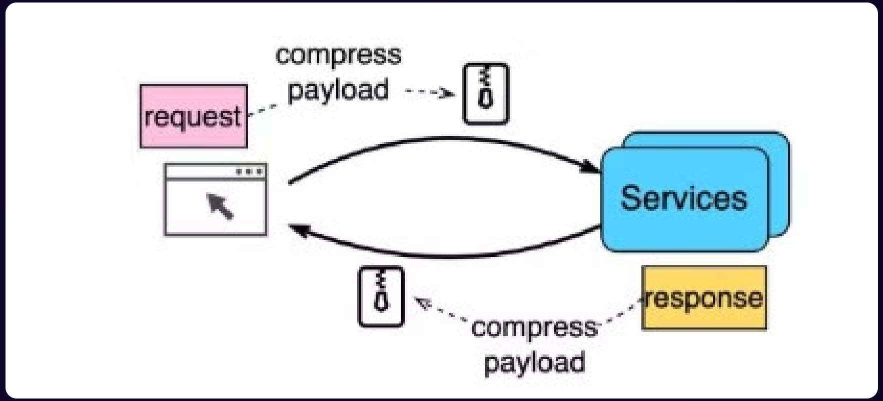
- Reduces bandwidth usage
- Faster response times
- Improved user experience

Compression Techniques

- Gzip
- Brotli
- HTTP/2 multiplexing

Configuration (Spring Boot)

```
server:
  compression:
    enabled: true
    mime-types:
application/json,application/xml,text/html,text/xml,text/plain
    min-response-size: 1024
```



Made with Gamma

Connection Pooling: Optimizing Database Interactions

Why Connection Pooling?

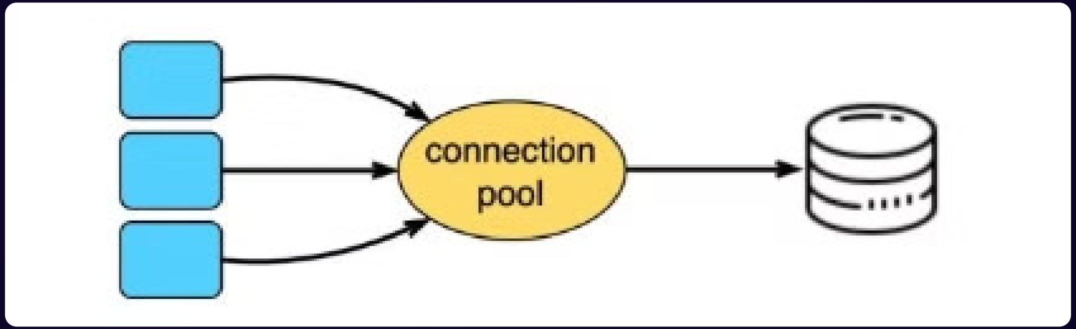
Reduces overhead of opening/closing database connections.

Best Practices

- Use HikariCP for performance.
- Tune pool size based on traffic.

Example (HikariCP Config):

```
spring:
  datasource:
    hikari:
      maximum-pool-size: 10
      minimum-idle: 5
      idle-timeout: 30000
```



Key Takeaways

1. **Monitor & Profile APIs**

   Use tools like Prometheus and Grafana to identify bottlenecks.

2. **Implement Caching & Compression**

   Reduce bandwidth usage and improve response times.

3. **Optimize Database Queries & Connection Pooling**

   Enhance database interaction efficiency.

4. **Leverage Async Processing**

   Handle heavy tasks such as logging.

5. **Test & Scale Proactively**

   Use testing to prepare APIs for peak loads.

Thank You

LinkedIn: https://www.linkedin.com/in/garima24agarwal/