Contents

1 Asynchronous Document Generation Architecture 3

1.1 Overview . . . 4
1.2 Architecture . . . 5
1.2.1 Communication Options . . . 5
1.2.2 Direct Queue Communication . . . 5
1.2.3 HTTP Bulk Submission . . . 6
1.3 Key Components . . . 7
1.3.1 Message Models . . . 7
1.3.2 Queue Configuration . . . 8
1.3.3 Protocol Connection Details . . . 8
1.3.4 Wire-Level Message Format . . . 8
1.3.5 Retry Semantics and Redelivery Policy . . . 9
1.3.6 Message TTL and Expiration . . . 10
1.3.7 Idempotency Guarantees . . . 11
1.3.8 Worker Component . . . 12
1.3.9 Database Tracking . . . 12
1.3.10 Cleanup Scheduler . . . 13
1.3.11 HTTP Bulk Submission Service . . . 13
1.4 Configuration . . . 15
1.4.1 Application Properties . . . 15
1.4.2 Environment Variables (Docker/Kubernetes) . . . 15
1.5 Multi-Pod Deployment . . . 16
1.5.1 Horizontal Scalability . . . 16
1.5.2 Load Balancing . . . 16
1.5.3 Benefits . . . 16
1.5.4 Thread-Safe Template Caching . . . 16
1.5.5 Kubernetes Deployment Example . . . 17
1.5.6 Autoscaling . . . 18
1.6 Client Implementation . . . 20
1.6.1 Java Client Example . . . 20
1.6.2 Python Client Example . . . 21
1.6.3 HTTP Polling for Result Retrieval . . . 22
1.7 Performance Characteristics . . . 25
1.7.1 Throughput . . . 25
1.7.2 Latency . . . 25
1.7.3 Resource Usage . . . 25
1.8 Monitoring . . . 26
1.8.1 REST API Endpoints . . . 26
1.8.2 Database Metrics . . . 28

1.8.3 SQL Queries . . . 28
1.8.4 Logging . . . 29
1.9 Security Considerations . . . 30
1.9.1 Message Security . . . 30
1.9.2 Data Protection . . . 30
1.10 Limitations . . . 31
1.10.1 Size Limits . . . 31
1.10.2 Timeout . . . 31
1.10.3 Formats . . . 31
1.11 Troubleshooting . . . 32
1.11.1 Request Not Processed . . . 32
1.11.2 Slow Processing . . . 32
1.11.3 Messages in Dead Letter Queue . . . 32

Asynchronous Document Generation Architecture

Version: 1.6
Document Type: Service Contract Specification
Target Audience: External System Integrators
Classification: Public API Documentation
Last Updated: March 12, 2026

1.1 Overview

The Muban Document Generation Service supports asynchronous document generation using **Apache ActiveMQ** (or compatible solutions) as the message broker. This feature enables non-blocking document processing, horizontal scalability, and high-throughput document generation in distributed environments.

1.2 Architecture

1.2.1 Communication Options

The async mode supports multiple submission and retrieval methods:

Submission Methods:

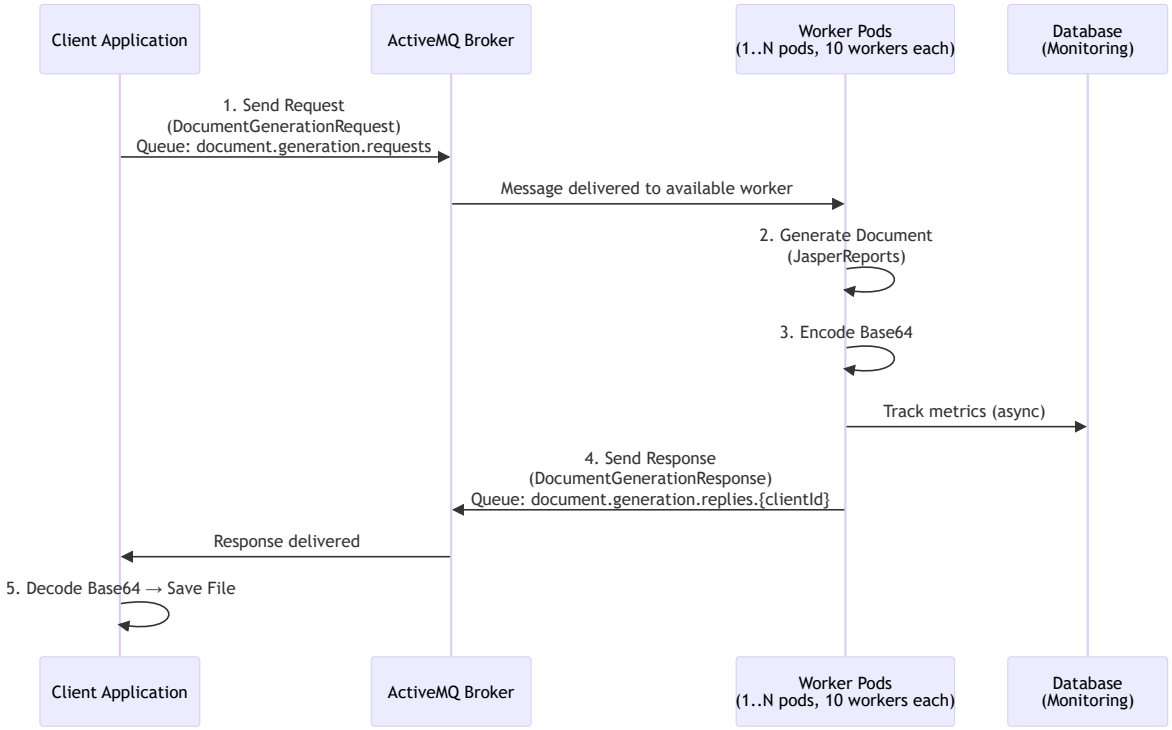
1. **Direct Queue Submission** - Clients send messages directly to ActiveMQ queues
2. **HTTP Bulk Submission** - Clients submit batches of requests via `POST /api/v1/async/bulk`

Result Retrieval Methods:

1. **JMS Queue Consumption** - Clients consume responses directly from reply queues (default)
2. **HTTP Polling** - Clients poll for results via `GET /api/v1/async/results/{requestId}`

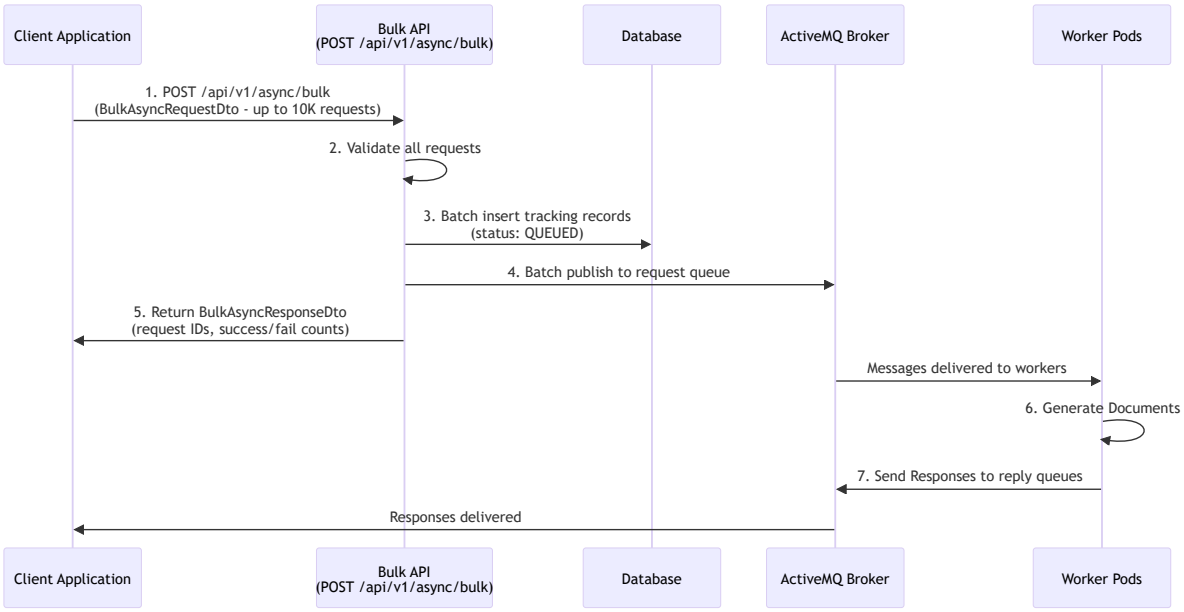
Both retrieval methods can be used simultaneously. HTTP polling provides a synchronous API alternative for clients that cannot maintain persistent JMS connections.

1.2.2 Direct Queue Communication



1.2.3 HTTP Bulk Submission

For clients that prefer HTTP over direct JMS connections:



1.3 Key Components

1.3.1 Message Models

1.3.1.1 DocumentGenerationRequest

Request message sent to the request queue by clients.

```
{
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "correlationId": "client-correlation-123",
  "templateId": "template-uuid",
  "format": "PDF",
  "parameters": [
    {"name": "title", "value": "Monthly Report"},
    {"name": "date", "value": "2025-12-03"}
  ],
  "data": {
    "items": [{"id": 1, "name": "Sample Item"}]
  },
  "documentLocale": "en",
  "ignorePagination": false,
  "pdfExportOptions": {
    "pdfaConformance": "PDF/A-1b"
  },
  "htmlExportOptions": null,
  "txtExportOptions": null,
  "filename": "monthly_report",
  "replyQueue": "document.generation.replies.client-abc-123",
  "userId": "user@example.com",
  "priority": 5,
  "timestamp": "2025-12-03T10:30:00",
  "timeoutSeconds": 300
}
```

1.3.1.2 DocumentGenerationResponse

Response message sent to the client's reply queue.

Success Response:

```
{
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "correlationId": "client-correlation-123",
  "status": "COMPLETED",
  "documentBase64": "JVBERi0xLjQKJeLjz9MKMy...", // Base64-encoded document
  "filename": "monthly_report.pdf",
  "contentType": "application/pdf",
  "fileSize": 245678,              // Original size (bytes)
  "base64Size": 327570,            // Encoded size (bytes)
  "processingTimeMs": 2340,
  "timestamp": "2025-12-03T10:30:45"
}
```

Failure Response:

```
{
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
```

```
  "correlationId": "client-correlation-123",
  "status": "FAILED",
  "error": "Template not found",
  "errorCode": "ResourceNotFoundException",
  "stackTrace": null,  // Optional - only if document.async.include-stack-traces=true
  "processingTimeMs": 150,
  "timestamp": "2025-12-03T10:30:45"
}
```

1.3.2 Queue Configuration

Queue Name	Purpose	Type
`document.generation.requests`	Incoming generation requests	Shared queue (all pods)
`document.generation.replies.{clientId}`	Client-specific reply queue	Per-client queue
`document.generation.dlq`	Dead letter queue for failed messages	Error queue

1.3.3 Protocol Connection Details

ActiveMQ supports multiple protocols. Choose based on your client technology:

Protocol	Port	Use Case	Client Libraries
OpenWire	61616	Java/JMS clients (recommended)	ActiveMQ Classic, Spring JMS
STOMP	61613	Python, Ruby, JavaScript, Go	stomp.py, stompest, stompit
AMQP	5672	Cross-platform, .NET, Node.js	RabbitMQ clients, rhea, amqplib
MQTT	1883	IoT, lightweight clients	Eclipse Paho, MQTT.js
WebSocket	61614	Browser-based clients	STOMP over WebSocket

Connection URLs by Protocol:

```
# OpenWire (Java/Spring) - DEFAULT
tcp://activemq-host:61616

# OpenWire with failover
failover:(tcp://broker1:61616,tcp://broker2:61616)?randomize=false

# STOMP (Python, Ruby)
stomp://activemq-host:61613

# AMQP 1.0
amqp://activemq-host:5672

# WebSocket (Browser)
ws://activemq-host:61614/stomp
```

1.3.4 Wire-Level Message Format

Messages are serialized as **JSON text messages** with the following JMS properties:

1.3.4.1 Request Message Properties

JMS Property	Type	Description	Example
`_type`	String	Fully qualified class name for deserialization	`...model.DocumentGenerationRequest*`
`JMSCorrelationID`	String	Client correlation ID (copied from request body)	`order-12345`
`JMSPriority`	Integer	Message priority (0-9, default 4)	`5`
`JMSTimestamp`	Long	Message creation timestamp (epoch ms)	`1737907200000`
`JMSMessageID`	String	Broker-assigned unique ID	`ID:broker-123...`

*Full package: `me.muban.documentservice.messaging.model`

1.3.4.2 Response Message Properties

JMS Property	Type	Description	Example
`_type`	String	Response class name	`...model.DocumentGenerationResponse*`
`JMSCorrelationID`	String	Copied from request for matching	`order-12345`
`status`	String	Processing status (custom property)	`COMPLETED`, `FAILED`
`requestId`	String	Request UUID (custom property)	`550e8400-...`

*Full package: `me.muban.documentservice.messaging.model`

1.3.4.3 Example Raw Message (OpenWire/STOMP)

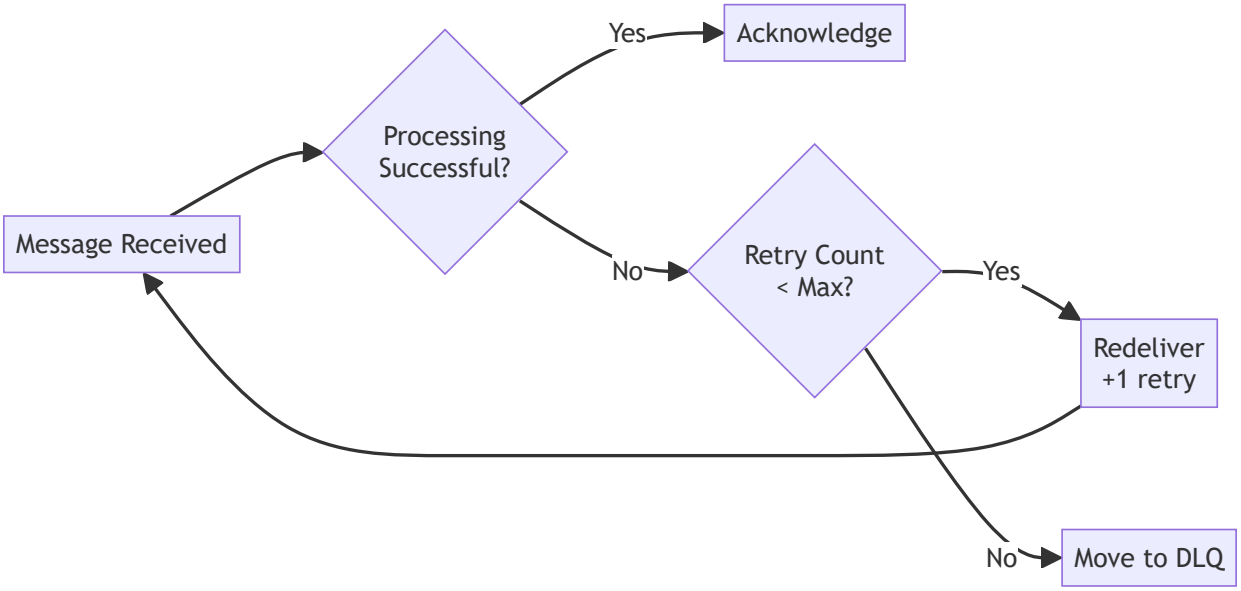
```
MESSAGE
destination:/queue/document.generation.replies.client-abc
content-type:application/json
correlation-id:order-12345
status:COMPLETED
requestId:550e8400-e29b-41d4-a716-446655440000

{"requestId":"550e8400-e29b-41d4-a716-446655440000","correlationId":"order-
↪ 12345","status":"COMPLETED","documentBase64":"JVBERi0x...","filename":"report.pdf","contentType":"application/pd
↪ 01-26T10:30:45"}
```

1.3.5 Retry Semantics and Redelivery Policy

The system uses **broker-managed redelivery** with the following behavior:

Parameter	Value	Description
`max-retries`	3 (configurable)	Maximum redelivery attempts before DLQ
Redelivery delay	Immediate	No delay between retries (broker default)
Backoff multiplier	None	Fixed delay, no exponential backoff
DLQ destination	`document.generation.dlq`	Dead letter queue for exhausted retries

Retry Flow:



What Triggers a Retry:

- Unhandled exception during processing
- Database connection failure (transient)
- Template extraction failure (corrupted ZIP)
- JasperReports compilation error

What Does NOT Trigger a Retry (Immediate Failure):

- Template not found (404) — fails immediately, no retry
- Invalid format — validation error, no retry
- Size limit exceeded — fails immediately, no retry

Tracking Retries:

The `retry_count` field in the database tracks delivery attempts. Check via monitoring API:

```
GET /api/v1/async/requests/{requestId}
```

1.3.6 Message TTL and Expiration

Message Type	Default TTL	Configurable	Behavior on Expiry
Request messages	None (infinite)	No	Processed eventually
Response messages	None (infinite)	No	Client must consume
DLQ messages	Cleanup retention	Yes	Deleted by scheduler

Important: Messages do **not** expire in queues. If a client disconnects without consuming responses, messages remain in the reply queue until:

1. Client reconnects and consumes them
2. Administrator manually purges the queue
3. Broker restart (if non-persistent)

Recommendation: Clients should always consume all responses from their reply queue, even after reconnection.

Database Record Retention:

Status	Retention	Cleanup
COMPLETED	7 days (configurable)	Automatic via scheduler
FAILED	7 days (configurable)	Automatic via scheduler
PROCESSING	Never deleted	Manual intervention if stuck
QUEUED	Never deleted	Will be processed eventually

Configure retention via:

```
document.async.cleanup.retention-hours=168  # 7 days
document.async.cleanup.cron=0 0 3 * * ?     # Daily at 3 AM
```

1.3.7 Idempotency Guarantees

Current Behavior (At-Least-Once Delivery):

The system does **NOT** enforce idempotency by `requestId`. If you send the same `requestId` twice:

Scenario	Behavior
Same `requestId` sent twice	Both processed, two responses sent
Database tracking	Second request **overwrites** first record
Reply queue	Client receives **two** responses

Why No Deduplication:

- Performance: Checking every requestId adds latency
- Simplicity: Stateless worker design
- Client responsibility: Clients can track their own request IDs

Client-Side Idempotency Recommendations:

1. **Generate unique `requestId`** per logical request
2. **Track sent requests** in your own database
3. **Deduplicate responses** by `requestId` on receive
4. **Use `correlationId`** for business-level tracking (order ID, invoice number)

Example Deduplication Pattern:

```python
processed_requests = set()

def on_message(response):
    request_id = response['requestId']
    if request_id in processed_requests:
        log.warning(f"Duplicate response ignored: {request_id}")
        return
    processed_requests.add(request_id)
    handle_response(response)
```

Future Consideration:

Server-side deduplication using Redis or database lookup may be added in a future version if demand exists.

1.3.8 Worker Component

`DocumentGenerationWorker` - JMS listener that processes requests:

```
@JmsListener(destination = "${document.async.request-queue}")
@Transactional
public void processGenerationRequest(DocumentGenerationRequest request) {
    // 1. Generate document using JasperReportService
    // 2. Convert to base64
    // 3. Validate size limits
    // 4. Send response to reply queue
    // 5. Track in database (monitoring only)
}
```

Features:

- Transactional processing (JMS + Database)
- Automatic retry on failure
- Size validation (max 10 MB document, 14 MB base64)
- Configurable timeout
- Error handling with detailed messages

1.3.9 Database Tracking

`async_document_requests` table tracks requests for **operational monitoring only** (NOT used by clients):

```
CREATE TABLE async_document_requests (
    request_id UUID PRIMARY KEY,
    correlation_id VARCHAR(255),
    template_id UUID NOT NULL,
    format VARCHAR(10) NOT NULL,
    user_id VARCHAR(255) NOT NULL,
    status VARCHAR(20) NOT NULL,  -- QUEUED, PROCESSING, COMPLETED, FAILED, TIMEOUT
    reply_queue VARCHAR(255),
    filename VARCHAR(255),
    file_size BIGINT,
    base64_size BIGINT,
    processing_time_ms BIGINT,
    error_message TEXT,
    created_at TIMESTAMP NOT NULL,
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    retry_count INTEGER DEFAULT 0,
    priority INTEGER DEFAULT 5
);
```

Purpose:

- Operational metrics and monitoring
- Performance analysis
- Audit trail
- Troubleshooting

NOT used for:

- Client status checks (clients only use queues)
- Document storage
- Request/response routing

1.3.10 Cleanup Scheduler

`AsyncRequestCleanupScheduler` - Scheduled task for database maintenance:

```
@Scheduled(cron = "${document.async.cleanup.cron:0 0 3 * * ?}")
public void cleanupOldRequests() {
    // Delete records older than retention period (default: 7 days)
    // Log statistics (queued, processing, completed, failed)
    // Calculate average processing time
}
```

Note: This is separate from `OutputFileCleanupService` which cleans temporary files from the filesystem.

1.3.11 HTTP Bulk Submission Service

`AsyncBulkSubmissionService` - Service for bulk HTTP submissions:

For clients that prefer HTTP over direct JMS connections, the bulk submission endpoint provides an efficient way to submit up to 10,000 requests in a single HTTP call.

Endpoint: `POST /api/v1/async/bulk`

Request Format:

```
{
  "batchCorrelationId": "batch-2025-12-30-001",
  "requests": [
    {
      "requestId": "550e8400-e29b-41d4-a716-446655440000",
      "correlationId": "order-12345",
      "templateId": "123e4567-e89b-12d3-a456-426614174000",
      "format": "PDF",
      "parameters": [
        {"name": "reportTitle", "value": "Monthly Report"}
      ],
      "data": {"items": [{"id": 1}]},
      "documentLocale": "pl_PL",
      "pdfExportOptions": {"pdfaConformance": "PDF/A-1b"},
      "htmlExportOptions": null,
      "txtExportOptions": null,
      "filename": "monthly_report",
      "replyQueue": "document.generation.replies.client-abc",
      "priority": 5,
      "timeoutSeconds": 300
    }
  ]
}
```

Response Format:

```
{
  "batchCorrelationId": "batch-2025-12-30-001",
  "submittedAt": "2025-12-30T10:00:00Z",
  "totalRequests": 1000,
  "successCount": 998,
  "failedCount": 2,
  "queueDepthBefore": 150,
  "estimatedTotalProcessingMs": 300000,
  "queuedRequests": [
    {
      "requestId": "550e8400-e29b-41d4-a716-446655440000",
      "correlationId": "order-12345",
      "index": 0,
      "templateId": "123e4567-e89b-12d3-a456-426614174000",
      "format": "PDF"
    }
  ],
  "failedRequests": [
    {
      "index": 5,
      "correlationId": "order-12350",
      "errorCode": "VALIDATION_ERROR",
      "errorMessage": "Template not found"
    }
  ]
}
```


Key Features:

- **Batch size**: Up to 10,000 requests per call
- **Batch database insert**: Uses `saveAll()` for efficient DB writes
- **Batch JMS publish**: Rapid sequential JMS publishing
- **Individual validation**: Each request validated independently
- **Partial success**: Valid requests are queued even if some fail validation
- **Same response delivery**: Responses delivered to JMS reply queues (same as direct queue submission)

When to use HTTP Bulk vs Direct Queue:

Aspect	HTTP Bulk	Direct Queue
Client complexity	Simpler (just HTTP)	Requires JMS client
Throughput	Good for batches	Best for streaming
Connection management	Stateless HTTP	Persistent JMS connection
Request grouping	Natural batching	Manual batching
Validation feedback	Immediate in response	Async via reply queue

Correlation ID Handling:

- `batchCorrelationId`: Client can provide, or UUID is auto-generated
- Per-request `correlationId`: Client can provide, or derived as `{batchCorrelationId}-{index}`
- `replyQueue`: Client must specify (no default magic)

1.4 Configuration

1.4.1 Application Properties

```
# Enable/disable async processing
document.async.enabled=false

# ActiveMQ connection
document.async.broker-url=tcp://localhost:61616
document.async.username=admin
document.async.password=admin

# Queue names
document.async.request-queue=document.generation.requests
document.async.reply-queue-prefix=document.generation.replies.
document.async.dead-letter-queue=document.generation.dlq

# Processing configuration
document.async.worker-threads=10
document.async.max-retries=3
document.async.processing-timeout-seconds=300
document.async.max-document-size-mb=10
document.async.max-base64-size-mb=14
document.async.max-queue-size=1000
document.async.include-stack-traces=false

# Cleanup configuration
document.async.cleanup.enabled=true
document.async.cleanup.retention-hours=168  # 7 days
document.async.cleanup.cron=0 0 3 * * ?     # 3 AM daily
```

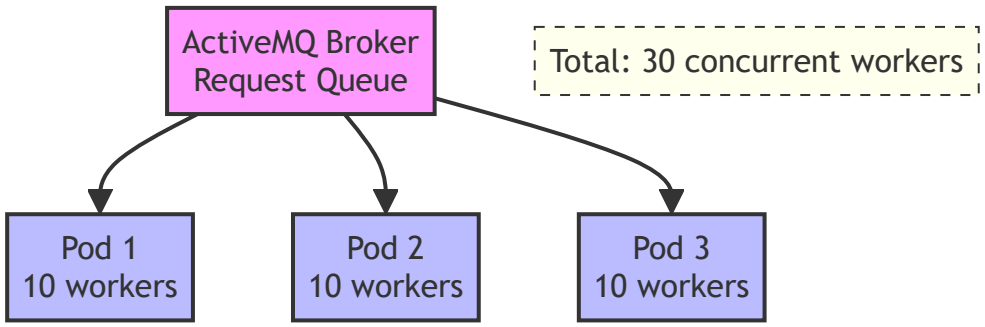
1.4.2 Environment Variables (Docker/Kubernetes)

```
ASYNC_ENABLED=true
ACTIVEMQ_BROKER_URL=tcp://activemq-service:61616
ACTIVEMQ_USERNAME=admin
ACTIVEMQ_PASSWORD=admin
ASYNC_WORKER_THREADS=10
ASYNC_MAX_DOC_SIZE=10
ASYNC_RETENTION_HOURS=168
```

1.5 Multi-Pod Deployment

1.5.1 Horizontal Scalability

The async mode is designed for **horizontal scalability** with multiple pods:



1.5.2 Load Balancing

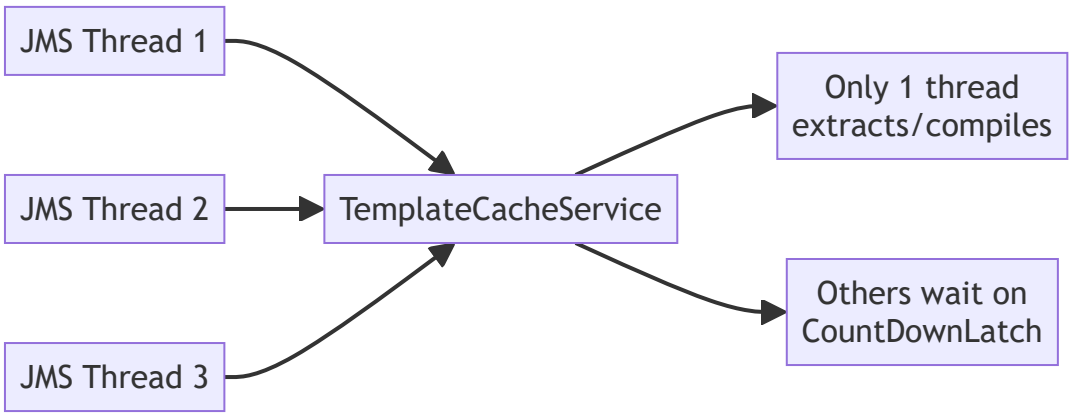
- **Automatic distribution**: ActiveMQ distributes messages to available consumers
- **Competing consumers**: Workers from all pods compete for messages
- **First-come, first-served**: No pod affinity required
- **Exactly-once processing**: Each request processed by exactly one worker

1.5.3 Benefits

- **High Availability** - If one pod dies, others continue processing
- **Load Distribution** - Work automatically spread across pods
- **Easy Scaling** - Add/remove pods without code changes
- **Fault Tolerance** - Failed messages automatically retried
- **Stateless Workers** - Any pod can process any request

1.5.4 Thread-Safe Template Caching

When multiple JMS listener threads within a single pod receive requests for the **same template** simultaneously (cold cache scenario), the `TemplateCacheService` ensures **only one thread** performs template preparation:

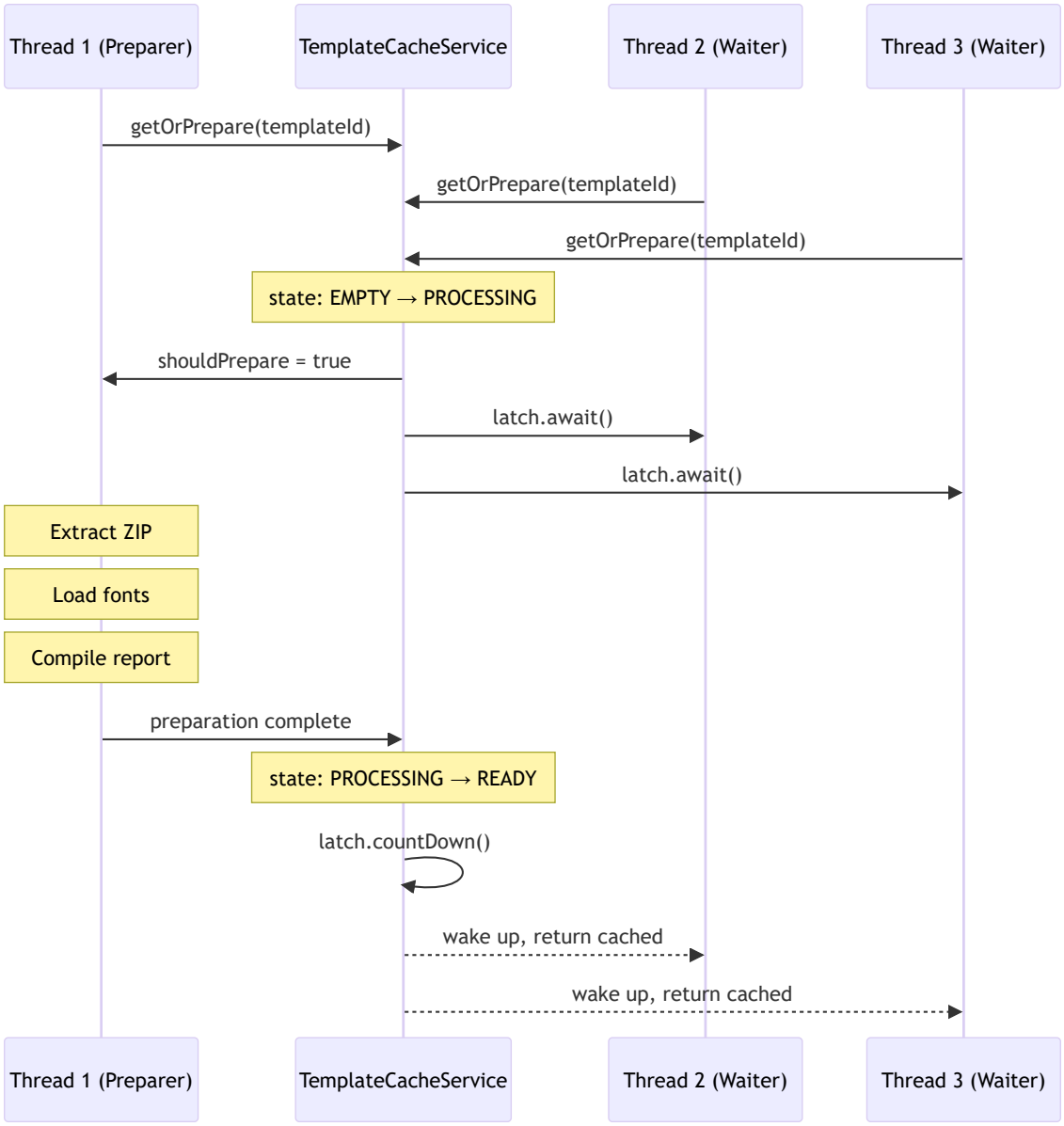


Why this matters for JMS:

- JMS messages arrive with **zero network latency** (vs HTTP with network jitter)

- Multiple listener threads (`spring.jms.listener.max-concurrency=5`) receive messages simultaneously
- Without protection, all threads would try to prepare the same template concurrently

State Machine Flow:



1.5.5 Kubernetes Deployment Example

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: document-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: document-service
  template:
```

```
    metadata:
      labels:
        app: document-service
    spec:
      containers:
      - name: document-service
        image: document-service:1.5.0
        env:
        - name: ASYNC_ENABLED
          value: "true"
        - name: ACTIVEMQ_BROKER_URL
          value: "tcp://activemq-service:61616"
        - name: ASYNC_WORKER_THREADS
          value: "10"
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
---
apiVersion: v1
kind: Service
metadata:
  name: activemq-service
spec:
  selector:
    app: activemq
  ports:
  - port: 61616
    targetPort: 61616
```

1.5.6 Autoscaling

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: document-service-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: document-service
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
```

```
    target:
      type: Utilization
      averageUtilization: 80
```

1.6 Client Implementation

1.6.1 Java Client Example

```java
// 1. Connect to ActiveMQ
ConnectionFactory factory = new ActiveMQConnectionFactory("tcp://localhost:61616");
Connection connection = factory.createConnection("admin", "admin");
connection.start();
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

// 2. Create reply queue
String replyQueueName = "document.generation.replies." + UUID.randomUUID();
Destination replyQueue = session.createQueue(replyQueueName);
MessageConsumer consumer = session.createConsumer(replyQueue);

// 3. Build request
DocumentGenerationRequest request = DocumentGenerationRequest.builder()
    .requestId(UUID.randomUUID())
    .correlationId("my-correlation-123")
    .templateId(UUID.fromString("550e8400-e29b-41d4-a716-446655440000"))
    .format("PDF")
    .parameters(parameters)
    .data(jsonData)
    .filename("report")
    .replyQueue(replyQueueName)
    .userId("user@example.com")
    .priority(5)
    .timestamp(LocalDateTime.now())
    .build();

// 4. Send request
Destination requestQueue = session.createQueue("document.generation.requests");
MessageProducer producer = session.createProducer(requestQueue);
ObjectMessage message = session.createObjectMessage(request);
message.setJMSCorrelationID(request.getCorrelationId());
producer.send(message);

// 5. Wait for response
Message responseMessage = consumer.receive(300000); // 5 minutes timeout
DocumentGenerationResponse response =
    (DocumentGenerationResponse) ((ObjectMessage) responseMessage).getObject();

// 6. Process response
if (response.getStatus() == MessageStatus.COMPLETED) {
    byte[] documentBytes = Base64.getDecoder().decode(response.getDocumentBase64());
    Files.write(Paths.get(response.getFilename()), documentBytes);
    System.out.println("Document generated: " + response.getFilename());
} else {
    System.err.println("Generation failed: " + response.getError());
}

// 7. Cleanup
consumer.close();
producer.close();
session.close();
connection.close();
```

1.6.2 Python Client Example

```python
import stomp
import json
import base64
import uuid
from datetime import datetime

# 1. Connect to ActiveMQ via STOMP protocol
# Note: STOMP uses port 61613, not OpenWire port 61616
conn = stomp.Connection([('localhost', 61613)])
conn.connect('admin', 'admin', wait=True)

# 2. Create reply queue
reply_queue = f"document.generation.replies.{uuid.uuid4()}"

# 3. Build request
request = {
    "requestId": str(uuid.uuid4()),
    "correlationId": "my-correlation-123",
    "templateId": "550e8400-e29b-41d4-a716-446655440000",
    "format": "PDF",
    "parameters": [
        {"name": "title", "value": "Report"},
        {"name": "date", "value": "2025-12-03"}
    ],
    "data": {"items": [{"id": 1, "name": "Item 1"}]},
    "filename": "report",
    "replyQueue": reply_queue,
    "userId": "user@example.com",
    "priority": 5,
    "timestamp": datetime.now().isoformat()
}

# 4. Subscribe to reply queue
class ResponseListener(stomp.ConnectionListener):
    def on_message(self, frame):
        response = json.loads(frame.body)
        if response['status'] == 'COMPLETED':
            doc_bytes = base64.b64decode(response['documentBase64'])
            with open(response['filename'], 'wb') as f:
                f.write(doc_bytes)
            print(f"Document saved: {response['filename']}")
        else:
            print(f"Error: {response['error']}")

conn.set_listener('', ResponseListener())
conn.subscribe(destination=reply_queue, id=1, ack='auto')

# 5. Send request
conn.send(body=json.dumps(request),
          destination='document.generation.requests',
          headers={'correlation-id': request['correlationId']})

# 6. Wait for response (with timeout)
time.sleep(300)  # 5 minutes
conn.disconnect()
```

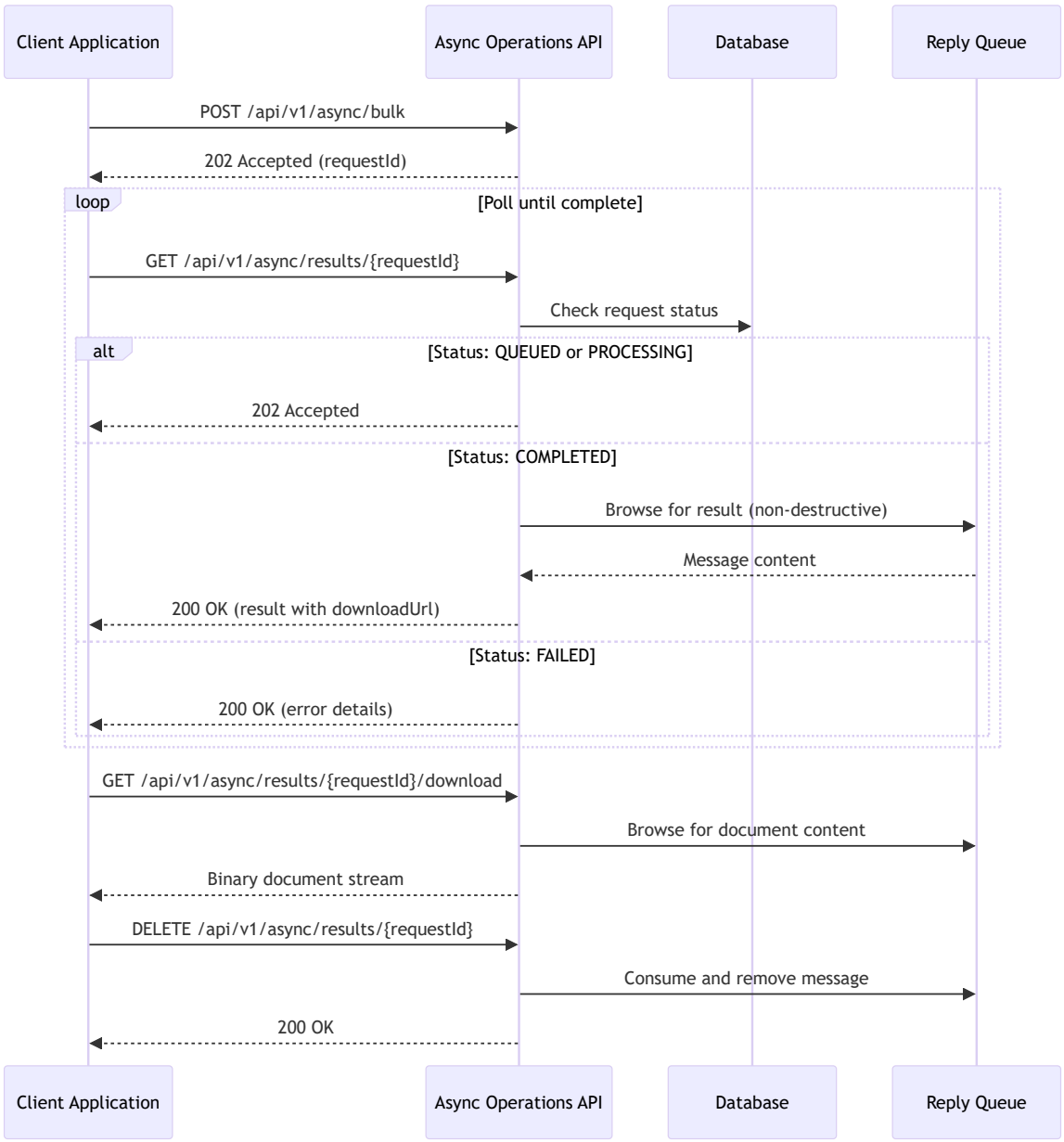
1.6.3 HTTP Polling for Result Retrieval

As an alternative to JMS queue consumption, clients may retrieve async results via HTTP polling. This approach is suitable for web applications and clients that prefer synchronous HTTP over persistent message queue connections.

1.6.3.1 Endpoints

Method	Endpoint	Description
GET	`/api/v1/async/results/{requestId}`	Check status and retrieve result metadata
GET	`/api/v1/async/results/{requestId}/download`	Download completed document as binary
DELETE	`/api/v1/async/results/{requestId}`	Acknowledge and remove result from queue

1.6.3.2 HTTP Polling Flow



1.6.3.3 Response Status Codes

HTTP Status	Meaning
200 OK	Request completed (COMPLETED or FAILED status)
202 Accepted	Request still processing (QUEUED or PROCESSING)
404 Not Found	Request ID not found or not owned by user
403 Forbidden	Request belongs to another user

1.6.3.4 Example: Polling with cURL

```
# 1. Submit async request
REQUEST_ID=$(curl -s -X POST "https://api.example.com/api/v1/async/bulk" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"requests": [{"templateId": "...", "format": "PDF"}]}' \
  | jq -r '.data.queuedRequests[0].requestId')

# 2. Poll for completion
while true; do
  RESPONSE=$(curl -s -w "\n%{http_code}" \
    "https://api.example.com/api/v1/async/results/$REQUEST_ID" \
    -H "Authorization: Bearer $TOKEN")

  HTTP_CODE=$(echo "$RESPONSE" | tail -1)

  if [ "$HTTP_CODE" = "200" ]; then
    echo "Complete"
    break
  elif [ "$HTTP_CODE" = "202" ]; then
    echo "Still processing..."
    sleep 2
  else
    echo "Error: $HTTP_CODE"
    exit 1
  fi
done

# 3. Download document
curl -o document.pdf \
  "https://api.example.com/api/v1/async/results/$REQUEST_ID/download" \
  -H "Authorization: Bearer $TOKEN"

# 4. Acknowledge (remove from queue)
curl -X DELETE \
  "https://api.example.com/api/v1/async/results/$REQUEST_ID" \
  -H "Authorization: Bearer $TOKEN"
```

1.6.3.5 Important Considerations

1. **Non-destructive Browse**: The GET endpoints use JMS QueueBrowser, leaving messages available for JMS clients.
2. **Destructive Acknowledge**: The DELETE operation permanently removes the message from the queue. JMS clients will no longer receive it.

3. **Dual-Mode Compatibility**: Both HTTP polling and JMS consumption can operate simultaneously. Coordinate between consumers to avoid duplicate processing.
4. **Queue Retention**: Unacknowledged results remain in the queue until TTL expiration or manual cleanup. Clients should acknowledge results after successful download.
5. **Authentication**: All endpoints require authentication. Users can only access their own async requests.
6. **Polling Interval**: Recommended polling interval is 1-5 seconds. Avoid aggressive polling to reduce server load.

1.7 Performance Characteristics

1.7.1 Throughput

Configuration	Throughput (docs/min)
1 pod, 10 workers	~100-150
3 pods, 30 workers	~300-450
5 pods, 50 workers	~500-750
10 pods, 100 workers	~1000-1500

Note: Actual throughput depends on document complexity, template size, and hardware.

1.7.2 Latency

Document Size	Avg Processing Time
Simple PDF (< 100 KB)	1-3 seconds
Medium PDF (100-500 KB)	3-10 seconds
Complex PDF (1-5 MB)	10-30 seconds
Large PDF (5-10 MB)	30-60 seconds

1.7.3 Resource Usage

Component	Memory (per pod)	CPU (per pod)
Base application	256 MB	0.2 cores
Worker threads (10)	+256 MB	+0.5 cores
Document processing	+512 MB peak	+0.3 cores peak
Recommended	**1 GB limit**	**1 core limit**

1.8 Monitoring

1.8.1 REST API Endpoints

The async monitoring module provides REST API endpoints for operational monitoring. These endpoints are only available when `document.async.enabled=true`.

Base URL: `/api/v1/async`

Endpoint	Method	Description
`/metrics`	GET	Dashboard metrics (counts, processing times, throughput)
`/requests`	GET	List async requests with filtering and pagination
`/requests/{requestId}`	GET	Get details of a specific request
`/workers`	GET	JMS worker status and configuration
`/errors`	GET	Recent failed requests with pagination
`/health`	GET	System health check (database, ActiveMQ, workers)

1.8.1.1 List Async Requests

```
GET /api/v1/async/requests?status=COMPLETED&userId=user1&templateId=UUID&since=2025-01-
↪ 01T00:00:00&page=1&size=20
```

Query Parameters:

Parameter	Type	Default	Description
`status`	String	-	Filter by status: `QUEUED`, `PROCESSING`, `COMPLETED`, `FAILED`, `TIMEOUT`
`userId`	String	-	Filter by user ID
`templateId`	UUID	-	Filter by template ID
`since`	DateTime	-	Filter requests created after this timestamp (ISO 8601)
`page`	Integer	1	Page number (1-based)
`size`	Integer	20	Items per page (max: 100)

Response:


```
{
  "meta": {
    "timestamp": "2025-12-30T10:00:00Z",
    "correlationId": "abc-123",
    "requestId": "req-456",
    "status": "success"
  },
  "data": {
    "totalItems": 150,
    "totalPages": 8,
    "currentPage": 1,
    "itemsPerPage": 20,
    "items": [
      {
        "requestId": "550e8400-e29b-41d4-a716-446655440000",
```

```
        "correlationId": "client-123",
        "templateId": "template-uuid",
        "format": "PDF",
        "userId": "user@example.com",
        "status": "COMPLETED",
        "filename": "report.pdf",
        "fileSize": 245678,
        "processingTimeMs": 2340,
        "createdAt": "2025-12-30T09:55:00Z",
        "completedAt": "2025-12-30T09:55:02Z"
      }
    ]
  }
}
```

1.8.1.2 List Recent Errors

```
GET /api/v1/async/errors?since=2025-12-29T00:00:00&page=1&size=20
```

Query Parameters:

Parameter	Type	Default	Description
since	DateTime	24h ago	Filter errors after this timestamp (ISO 8601)
page	Integer	1	Page number (1-based)
size	Integer	20	Items per page (max: 100)

Response:

```
{
  "meta": {
    "timestamp": "2025-12-30T10:00:00Z",
    "correlationId": "abc-123",
    "requestId": "req-456",
    "status": "success"
  },
  "data": {
    "totalItems": 5,
    "totalPages": 1,
    "currentPage": 1,
    "itemsPerPage": 20,
    "items": [
      {
        "requestId": "550e8400-e29b-41d4-a716-446655440001",
        "correlationId": "client-456",
        "templateId": "template-uuid",
        "format": "PDF",
        "userId": "user@example.com",
        "status": "FAILED",
        "errorMessage": "Template not found",
        "createdAt": "2025-12-30T09:50:00Z",
        "completedAt": "2025-12-30T09:50:01Z"
      }
    ]
  }
```

```
}
```

1.8.1.3 Get Metrics Dashboard

```
GET /api/v1/async/metrics
```

Response:

```
{
  "meta": { ... },
  "data": {
    "requestCounts": {
      "queued": 5,
      "processing": 2,
      "completed": 1500,
      "failed": 12,
      "timeout": 1
    },
    "averageProcessingTimeMs": 2340.5,
    "throughputPerMinute": 45.2
  }
}
```

1.8.2 Database Metrics

The system tracks the following metrics in the database:

- **Queue depth**: Number of pending requests
- **Processing time**: Average time per document
- **Success rate**: Completed vs failed requests
- **Error types**: Categorized failure reasons
- **Throughput**: Documents generated per time period

1.8.3 SQL Queries

```
-- Current queue status
SELECT status, COUNT(*)
FROM async_document_requests
GROUP BY status;

-- Average processing time (last 24 hours)
SELECT AVG(processing_time_ms) as avg_ms
FROM async_document_requests
WHERE status = 'COMPLETED'
AND completed_at > NOW() - INTERVAL '24 hours';

-- Error analysis
SELECT error_message, COUNT(*)
FROM async_document_requests
WHERE status = 'FAILED'
GROUP BY error_message
ORDER BY COUNT(*) DESC;

-- Throughput (last hour)
SELECT DATE_TRUNC('minute', completed_at) as minute,
```

```
       COUNT(*) as documents
FROM async_document_requests
WHERE status = 'COMPLETED'
AND completed_at > NOW() - INTERVAL '1 hour'
GROUP BY minute
ORDER BY minute;
```

1.8.4 Logging

Workers log the following:

- Request received with ID and format
- Processing start and completion
- File sizes (original and base64)
- Processing time
- Errors with stack traces (if enabled)

Example logs:

```
INFO  Processing async request 550e8400-e29b... - template abc12345-... in PDF format for user
 ↪  john@example.com
INFO  Document generated successfully: original=234KB, base64=312KB, time=2340ms
INFO  Reply sent to queue 'document.generation.replies.client-123' for request 550e8400-e29b... -
 ↪  status: COMPLETED
```

1.9 Security Considerations

1.9.1 Message Security

1. **Trusted Packages**: Only specific packages can be deserialized from JMS messages

```
factory.setTrustAllPackages(false);
factory.setTrustedPackages(Arrays.asList(
    "me.muban.documentservice.messaging.model",
    "java.util",
    "java.lang"
));
```

2. **Authentication**: ActiveMQ requires username/password authentication
3. **Queue Isolation**: Each client uses a unique reply queue

1.9.2 Data Protection

1. **Document Transmission**: Base64 encoding (not encryption) - consider TLS for ActiveMQ
2. **Sensitive Data**: Avoid including sensitive data in monitoring database
3. **Error Messages**: Stack traces disabled by default in production

1.10 Limitations

1.10.1 Size Limits

- **Maximum document size**: 10 MB (configurable)
- **Maximum base64 size**: 14 MB (configurable)
- **ActiveMQ message size**: ~64 MB default (configure if needed)

1.10.2 Timeout

- **Default processing timeout**: 5 minutes (300 seconds)
- **Configurable per request**: `timeoutSeconds` field

1.10.3 Formats

All synchronous formats are supported:

- PDF, XLSX, DOCX, RTF, HTML, TXT

1.11 Troubleshooting

1.11.1 Request Not Processed

Check:

1. Is `document.async.enabled=true`?
2. Is ActiveMQ broker running and accessible?
3. Check ActiveMQ queue depth - is queue full?
4. Check worker logs for errors
5. Check database for request status

1.11.2 Slow Processing

Check:

1. Worker thread count (`document.async.worker-threads`)
2. Database performance (tracking overhead)
3. Template complexity
4. Number of running pods
5. ActiveMQ broker performance

1.11.3 Messages in Dead Letter Queue

Check:

1. Error logs for exception details
2. Message retry count (max retries exceeded?)
3. Document size exceeding limits
4. Template not found or corrupted
5. Database connection issues