BANKING SYSTEM PROJECT - DAY 6: DYNAMODB INTEGRATION

=====================================================

OVERVIEW

--------

Day 6 focuses on integrating Amazon DynamoDB as a NoSQL database for storing audit logs and event data. This implementation provides scalable, high-performance storage for time-series data with eventual consistency and automatic scaling capabilities.

DYNAMODB CONFIGURATION

----------------------

1. AWS SDK DEPENDENCIES

- software.amazon.awssdk:dynamodb:2.21.29

- software.amazon.awssdk:dynamodb-enhanced:2.21.29

- AWS SDK Core libraries

- AWS Credentials and Region configuration

2. APPLICATION PROPERTIES

aws.region=ap-south-1

aws.dynamodb.table-name=BankingAuditLogs

aws.dynamodb.local.enabled=true

aws.dynamodb.local.port=8003

aws.credentials.access-key=${AWS\_ACCESS\_KEY\_ID}

aws.credentials.secret-key=${AWS\_SECRET\_ACCESS\_KEY}

3. DYNAMODB CLIENT CONFIGURATION

@Configuration

public class DynamoDBConfig {

@Value("${aws.region}")

private String awsRegion;

@Value("${aws.dynamodb.local.enabled}")

private boolean localEnabled;

@Value("${aws.dynamodb.local.port}")

private int localPort;

@Bean

public DynamoDbClient dynamoDbClient() {

DynamoDbClientBuilder builder = DynamoDbClient.builder()

.region(Region.of(awsRegion));

if (localEnabled) {

builder.endpointOverride(URI.create("http://localhost:" + localPort));

}

return builder.build();

}

@Bean

public DynamoDbEnhancedClient dynamoDbEnhancedClient(DynamoDbClient dynamoDbClient) {

return DynamoDbEnhancedClient.builder()

.dynamoDbClient(dynamoDbClient)

.build();

}

}

DYNAMODB TABLE DESIGN

---------------------

1. AUDIT LOGS TABLE SCHEMA

Table Name: BankingAuditLogs

Primary Key:

- Partition Key: actionId (String) - Unique action identifier

- Sort Key: timestamp (String) - ISO 8601 timestamp for sorting

Attributes:

- userId (String) - User who performed the action

- actionType (String) - Type of action performed

- entityType (String) - Type of entity affected

- entityId (String) - ID of the affected entity

- actionDetails (String) - Detailed description of the action

- ipAddress (String) - IP address of the user

- userAgent (String) - User agent/browser information

- status (String) - SUCCESS or FAILURE

- errorMessage (String) - Error details if action failed

- createdDate (String) - Record creation timestamp

- ttl (Number) - Time-to-live for automatic cleanup

2. GLOBAL SECONDARY INDEXES

GSI-1: UserActionsIndex

- Partition Key: userId (String)

- Sort Key: timestamp (String)

- Purpose: Query audit logs by user and time range

GSI-2: ActionTypeIndex

- Partition Key: actionType (String)

- Sort Key: timestamp (String)

- Purpose: Query audit logs by action type and time range

GSI-3: EntityIndex

- Partition Key: entityType (String)

- Sort Key: timestamp (String)

- Purpose: Query audit logs by entity type and time range

3. TABLE CREATION SCRIPT

@Component

public class DynamoDBTableInitializer {

@Autowired

private DynamoDbClient dynamoDbClient;

@PostConstruct

public void createTables() {

createAuditLogsTable();

}

private void createAuditLogsTable() {

CreateTableRequest createTableRequest = CreateTableRequest.builder()

.tableName("BankingAuditLogs")

.keySchema(

KeySchemaElement.builder()

.attributeName("actionId")

.keyType(KeyType.HASH)

.build(),

KeySchemaElement.builder()

.attributeName("timestamp")

.keyType(KeyType.RANGE)

.build()

)

.attributeDefinitions(

AttributeDefinition.builder()

.attributeName("actionId")

.attributeType(ScalarAttributeType.S)

.build(),

AttributeDefinition.builder()

.attributeName("timestamp")

.attributeType(ScalarAttributeType.S)

.build(),

AttributeDefinition.builder()

.attributeName("userId")

.attributeType(ScalarAttributeType.S)

.build(),

AttributeDefinition.builder()

.attributeName("actionType")

.attributeType(ScalarAttributeType.S)

.build(),

AttributeDefinition.builder()

.attributeName("entityType")

.attributeType(ScalarAttributeType.S)

.build()

)

.globalSecondaryIndexes(

GlobalSecondaryIndex.builder()

.indexName("UserActionsIndex")

.keySchema(

KeySchemaElement.builder()

.attributeName("userId")

.keyType(KeyType.HASH)

.build(),

KeySchemaElement.builder()

.attributeName("timestamp")

.keyType(KeyType.RANGE)

.build()

)

.projection(Projection.builder().projectionType(ProjectionType.ALL).build())

.provisionedThroughput(

ProvisionedThroughput.builder()

.readCapacityUnits(10L)

.writeCapacityUnits(10L)

.build()

)

.build()

)

.provisionedThroughput(

ProvisionedThroughput.builder()

.readCapacityUnits(20L)

.writeCapacityUnits(20L)

.build()

)

.build();

try {

dynamoDbClient.createTable(createTableRequest);

log.info("DynamoDB table 'BankingAuditLogs' created successfully");

} catch (ResourceInUseException e) {

log.info("DynamoDB table 'BankingAuditLogs' already exists");

}

}

}

DYNAMODB SERVICE IMPLEMENTATION

-------------------------------

1. DYNAMODB SERVICE CLASS

@Service

public class DynamoDBService {

private final DynamoDbEnhancedClient dynamoDbEnhancedClient;

private final DynamoDbTable<AuditLogItem> auditLogTable;

@Autowired

public DynamoDBService(DynamoDbEnhancedClient dynamoDbEnhancedClient) {

this.dynamoDbEnhancedClient = dynamoDbEnhancedClient;

this.auditLogTable = dynamoDbEnhancedClient.table("BankingAuditLogs",

TableSchema.fromBean(AuditLogItem.class));

}

}

2. AUDIT LOG ITEM MODEL

@DynamoDbBean

public class AuditLogItem {

private String actionId;

private String timestamp;

private String userId;

private String actionType;

private String entityType;

private String entityId;

private String actionDetails;

private String ipAddress;

private String userAgent;

private String status;

private String errorMessage;

private String createdDate;

private Long ttl;

@DynamoDbPartitionKey

public String getActionId() { return actionId; }

public void setActionId(String actionId) { this.actionId = actionId; }

@DynamoDbSortKey

public String getTimestamp() { return timestamp; }

public void setTimestamp(String timestamp) { this.timestamp = timestamp; }

@DynamoDbSecondaryPartitionKey(indexNames = "UserActionsIndex")

public String getUserId() { return userId; }

public void setUserId(String userId) { this.userId = userId; }

@DynamoDbSecondaryPartitionKey(indexNames = "ActionTypeIndex")

public String getActionType() { return actionType; }

public void setActionType(String actionType) { this.actionType = actionType; }

@DynamoDbSecondaryPartitionKey(indexNames = "EntityIndex")

public String getEntityType() { return entityType; }

public void setEntityType(String entityType) { this.entityType = entityType; }

// Other getters and setters

}

3. SAVE AUDIT LOG METHOD

public void saveAuditLog(AuditLog auditLog) {

try {

AuditLogItem auditLogItem = convertToAuditLogItem(auditLog);

auditLogTable.putItem(auditLogItem);

log.debug("Audit log saved to DynamoDB: {}", auditLog.getActionId());

} catch (Exception e) {

log.error("Failed to save audit log to DynamoDB: {}", e.getMessage());

throw new DynamoDBException("Failed to save audit log", e);

}

}

4. BATCH SAVE AUDIT LOGS

public void saveAuditLogsBatch(List<AuditLog> auditLogs) {

try {

List<AuditLogItem> auditLogItems = auditLogs.stream()

.map(this::convertToAuditLogItem)

.collect(Collectors.toList());

WriteBatch.Builder<AuditLogItem> batchBuilder = WriteBatch.builder(AuditLogItem.class)

.mappedTableResource(auditLogTable);

auditLogItems.forEach(batchBuilder::addPutItem);

BatchWriteItemEnhancedRequest batchRequest = BatchWriteItemEnhancedRequest.builder()

.writeBatches(batchBuilder.build())

.build();

dynamoDbEnhancedClient.batchWriteItem(batchRequest);

log.info("Batch saved {} audit logs to DynamoDB", auditLogs.size());

} catch (Exception e) {

log.error("Failed to batch save audit logs to DynamoDB: {}", e.getMessage());

throw new DynamoDBException("Failed to batch save audit logs", e);

}

}

QUERY OPERATIONS

----------------

1. QUERY BY USER ID

public List<AuditLogItem> getAuditLogsByUser(String userId, LocalDateTime fromDate, LocalDateTime toDate) {

try {

QueryConditional queryConditional = QueryConditional

.keyEqualTo(Key.builder().partitionValue(userId).build());

QueryEnhancedRequest.Builder queryBuilder = QueryEnhancedRequest.builder()

.queryConditional(queryConditional)

.indexName("UserActionsIndex");

if (fromDate != null && toDate != null) {

queryBuilder.filterExpression(

Expression.builder()

.expression("#timestamp BETWEEN :fromDate AND :toDate")

.putExpressionName("#timestamp", "timestamp")

.putExpressionValue(":fromDate", AttributeValue.builder().s(fromDate.toString()).build())

.putExpressionValue(":toDate", AttributeValue.builder().s(toDate.toString()).build())

.build()

);

}

PageIterable<AuditLogItem> pages = auditLogTable.index("UserActionsIndex")

.query(queryBuilder.build());

return pages.items().stream().collect(Collectors.toList());

} catch (Exception e) {

log.error("Failed to query audit logs by user: {}", e.getMessage());

throw new DynamoDBException("Failed to query audit logs by user", e);

}

}

2. QUERY BY ACTION TYPE

public List<AuditLogItem> getAuditLogsByActionType(String actionType, LocalDateTime fromDate, LocalDateTime toDate) {

try {

QueryConditional queryConditional = QueryConditional

.keyEqualTo(Key.builder().partitionValue(actionType).build());

QueryEnhancedRequest.Builder queryBuilder = QueryEnhancedRequest.builder()

.queryConditional(queryConditional)

.indexName("ActionTypeIndex");

if (fromDate != null && toDate != null) {

queryBuilder.filterExpression(

Expression.builder()

.expression("#timestamp BETWEEN :fromDate AND :toDate")

.putExpressionName("#timestamp", "timestamp")

.putExpressionValue(":fromDate", AttributeValue.builder().s(fromDate.toString()).build())

.putExpressionValue(":toDate", AttributeValue.builder().s(toDate.toString()).build())

.build()

);

}

PageIterable<AuditLogItem> pages = auditLogTable.index("ActionTypeIndex")

.query(queryBuilder.build());

return pages.items().stream().collect(Collectors.toList());

} catch (Exception e) {

log.error("Failed to query audit logs by action type: {}", e.getMessage());

throw new DynamoDBException("Failed to query audit logs by action type", e);

}

}

3. QUERY BY ENTITY TYPE

public List<AuditLogItem> getAuditLogsByEntityType(String entityType, LocalDateTime fromDate, LocalDateTime toDate) {

try {

QueryConditional queryConditional = QueryConditional

.keyEqualTo(Key.builder().partitionValue(entityType).build());

QueryEnhancedRequest.Builder queryBuilder = QueryEnhancedRequest.builder()

.queryConditional(queryConditional)

.indexName("EntityIndex");

if (fromDate != null && toDate != null) {

queryBuilder.filterExpression(

Expression.builder()

.expression("#timestamp BETWEEN :fromDate AND :toDate")

.putExpressionName("#timestamp", "timestamp")

.putExpressionValue(":fromDate", AttributeValue.builder().s(fromDate.toString()).build())

.putExpressionValue(":toDate", AttributeValue.builder().s(toDate.toString()).build())

.build()

);

}

PageIterable<AuditLogItem> pages = auditLogTable.index("EntityIndex")

.query(queryBuilder.build());

return pages.items().stream().collect(Collectors.toList());

} catch (Exception e) {

log.error("Failed to query audit logs by entity type: {}", e.getMessage());

throw new DynamoDBException("Failed to query audit logs by entity type", e);

}

}

SCAN OPERATIONS

---------------

1. SCAN WITH FILTERS

public List<AuditLogItem> scanAuditLogsWithFilters(Map<String, String> filters) {

try {

ScanEnhancedRequest.Builder scanBuilder = ScanEnhancedRequest.builder();

if (!filters.isEmpty()) {

Expression.Builder expressionBuilder = Expression.builder();

StringBuilder expression = new StringBuilder();

Map<String, String> expressionNames = new HashMap<>();

Map<String, AttributeValue> expressionValues = new HashMap<>();

int index = 0;

for (Map.Entry<String, String> filter : filters.entrySet()) {

if (index > 0) {

expression.append(" AND ");

}

String attributeName = "#attr" + index;

String valueName = ":val" + index;

expression.append(attributeName).append(" = ").append(valueName);

expressionNames.put(attributeName, filter.getKey());

expressionValues.put(valueName, AttributeValue.builder().s(filter.getValue()).build());

index++;

}

expressionBuilder

.expression(expression.toString())

.expressionNames(expressionNames)

.expressionValues(expressionValues);

scanBuilder.filterExpression(expressionBuilder.build());

}

PageIterable<AuditLogItem> pages = auditLogTable.scan(scanBuilder.build());

return pages.items().stream().collect(Collectors.toList());

} catch (Exception e) {

log.error("Failed to scan audit logs with filters: {}", e.getMessage());

throw new DynamoDBException("Failed to scan audit logs with filters", e);

}

}

2. PARALLEL SCAN

public List<AuditLogItem> parallelScanAuditLogs(int segments) {

try {

List<CompletableFuture<List<AuditLogItem>>> futures = new ArrayList<>();

for (int segment = 0; segment < segments; segment++) {

final int currentSegment = segment;

CompletableFuture<List<AuditLogItem>> future = CompletableFuture.supplyAsync(() -> {

ScanEnhancedRequest scanRequest = ScanEnhancedRequest.builder()

.segment(currentSegment)

.totalSegments(segments)

.build();

PageIterable<AuditLogItem> pages = auditLogTable.scan(scanRequest);

return pages.items().stream().collect(Collectors.toList());

});

futures.add(future);

}

List<AuditLogItem> allItems = new ArrayList<>();

for (CompletableFuture<List<AuditLogItem>> future : futures) {

allItems.addAll(future.get());

}

return allItems;

} catch (Exception e) {

log.error("Failed to parallel scan audit logs: {}", e.getMessage());

throw new DynamoDBException("Failed to parallel scan audit logs", e);

}

}

EVENTUAL CONSISTENCY HANDLING

-----------------------------

1. CONSISTENCY CONFIGURATION

public DynamoDbClient createDynamoDbClientWithConsistency() {

return DynamoDbClient.builder()

.region(Region.of(awsRegion))

.overrideConfiguration(

ClientOverrideConfiguration.builder()

.retryPolicy(

RetryPolicy.builder()

.numRetries(3)

.backoffStrategy(BackoffStrategy.defaultStrategy())

.build()

)

.build()

)

.build();

}

2. CONSISTENT READS

public AuditLogItem getAuditLogConsistent(String actionId, String timestamp) {

try {

GetItemEnhancedRequest getRequest = GetItemEnhancedRequest.builder()

.key(Key.builder()

.partitionValue(actionId)

.sortValue(timestamp)

.build())

.consistentRead(true)

.build();

return auditLogTable.getItem(getRequest);

} catch (Exception e) {

log.error("Failed to get audit log with consistent read: {}", e.getMessage());

throw new DynamoDBException("Failed to get audit log with consistent read", e);

}

}

3. EVENTUAL CONSISTENCY HANDLING

public List<AuditLogItem> getAuditLogsEventuallyConsistent(String userId) {

try {

QueryConditional queryConditional = QueryConditional

.keyEqualTo(Key.builder().partitionValue(userId).build());

QueryEnhancedRequest queryRequest = QueryEnhancedRequest.builder()

.queryConditional(queryConditional)

.indexName("UserActionsIndex")

.consistentRead(false) // Eventual consistency

.build();

PageIterable<AuditLogItem> pages = auditLogTable.index("UserActionsIndex")

.query(queryRequest);

return pages.items().stream().collect(Collectors.toList());

} catch (Exception e) {

log.error("Failed to get audit logs with eventual consistency: {}", e.getMessage());

throw new DynamoDBException("Failed to get audit logs with eventual consistency", e);

}

}

PERFORMANCE OPTIMIZATION

------------------------

1. BATCH OPERATIONS

public void batchWriteAuditLogs(List<AuditLogItem> auditLogItems) {

try {

int batchSize = 25; // DynamoDB batch write limit

List<List<AuditLogItem>> batches = partitionList(auditLogItems, batchSize);

for (List<AuditLogItem> batch : batches) {

WriteBatch.Builder<AuditLogItem> batchBuilder = WriteBatch.builder(AuditLogItem.class)

.mappedTableResource(auditLogTable);

batch.forEach(batchBuilder::addPutItem);

BatchWriteItemEnhancedRequest batchRequest = BatchWriteItemEnhancedRequest.builder()

.writeBatches(batchBuilder.build())

.build();

dynamoDbEnhancedClient.batchWriteItem(batchRequest);

}

} catch (Exception e) {

log.error("Failed to batch write audit logs: {}", e.getMessage());

throw new DynamoDBException("Failed to batch write audit logs", e);

}

}

2. CACHING STRATEGY

@Service

public class DynamoDBCacheService {

private final CacheManager cacheManager;

@Cacheable(value = "auditLogs", key = "#userId + '\_' + #fromDate + '\_' + #toDate")

public List<AuditLogItem> getCachedAuditLogsByUser(String userId, LocalDateTime fromDate, LocalDateTime toDate) {

return dynamoDBService.getAuditLogsByUser(userId, fromDate, toDate);

}

@CacheEvict(value = "auditLogs", allEntries = true)

public void evictAuditLogCache() {

// Cache eviction logic

}

}

3. CONNECTION POOLING

@Configuration

public class DynamoDBConnectionConfig {

@Bean

public DynamoDbClient dynamoDbClientWithConnectionPool() {

return DynamoDbClient.builder()

.region(Region.of(awsRegion))

.httpClient(

UrlConnectionHttpClient.builder()

.connectionTimeout(Duration.ofSeconds(10))

.socketTimeout(Duration.ofSeconds(30))

.build()

)

.build();

}

}

MONITORING AND METRICS

----------------------

1. CLOUDWATCH METRICS

@Component

public class DynamoDBMetrics {

private final CloudWatchClient cloudWatchClient;

public void putMetric(String metricName, double value, String unit) {

PutMetricDataRequest request = PutMetricDataRequest.builder()

.namespace("BankingSystem/DynamoDB")

.metricData(

MetricDatum.builder()

.metricName(metricName)

.value(value)

.unit(unit)

.timestamp(Instant.now())

.build()

)

.build();

cloudWatchClient.putMetricData(request);

}

}

2. PERFORMANCE MONITORING

public void monitorDynamoDBPerformance() {

// Monitor read capacity units

putMetric("ReadCapacityUnits", getReadCapacityUnits(), "Count");

// Monitor write capacity units

putMetric("WriteCapacityUnits", getWriteCapacityUnits(), "Count");

// Monitor throttled requests

putMetric("ThrottledRequests", getThrottledRequests(), "Count");

// Monitor successful requests

putMetric("SuccessfulRequests", getSuccessfulRequests(), "Count");

}

This DynamoDB integration provides scalable, high-performance NoSQL storage for audit logs with eventual consistency and automatic scaling capabilities.