Pseudo code of Core logic (C#-.Net)

```
public async Task HandleMessageAsync(string message, ulong deliveryTag)
{
  // Deserialize the incoming event
  var evt = JsonSerializer.Deserialize<EventMessage>(message);
  string eventId = evt.EventId;

  // Begin database transaction for atomic claim and process
  await using var transaction = await _db.Database.BeginTransactionAsync();

  try
  {
    // 1 Deduplication check
    bool alreadyProcessed = await _db.Events
      .AnyAsync(e => e.EventId == eventId && e.IsCompleted);

    if (alreadyProcessed)
    {
      _logger.LogInformation($"Duplicate event {eventId}, skipping.");
      _channel.BasicAck(deliveryTag, multiple: false);
      return;
    }

    // 2 Attempt to claim the event
    var record = new EventRecord
    {
      EventId = eventId,
      Payload = evt.Payload,
      IsCompleted = false,
      ProcessedAt = DateTime.UtcNow,
            LastAttemptedAt = DateTime.UtcNow
        };

        _db.Events.Add(record);
        await _db.SaveChangesAsync();

        // 3 Process the event (business logic)
        await ProcessEvent(evt);

        // 4 Mark the event as completed
        record.IsCompleted = true;
        record.LastAttemptedAt = DateTime.UtcNow;
        await _db.SaveChangesAsync();

        // 5 Commit transaction and acknowledge
        await transaction.CommitAsync();
        _channel.BasicAck(deliveryTag, multiple: false);
        _logger.LogInformation($"Event {eventId} processed successfully.");
    }
    catch (DbUpdateException dbEx) when (dbEx.InnerException?.Message.Contains("duplicate") ==
true)
    {
        // Handles concurrent insert race condition (duplicate EventId)
        _logger.LogWarning($"Duplicate event {eventId} detected at DB level.");
        await transaction.RollbackAsync();
        _channel.BasicAck(deliveryTag, multiple: false);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, $"Failed to process event {eventId}");
        // Rollback on failure and requeue the message for retry
        await transaction.RollbackAsync();
        _channel.BasicNack(deliveryTag, multiple: false, requeue: true);
    }
}
```