Event Manager Migration: Developer's Guide (Easy-Going Edition!) 🚀

Document Version: 1.0 (Now with more pizzazz!)
Date: May 26, 2025
Who's This For? Our Awesome Dev Team!

1. The Lowdown (aka Executive Summary)

Alright team, here's the scoop: We've got this older app called "Event Manager." Think of it as a busy traffic cop for data, directing and sometimes changing messages as they fly through our systems. The problem? It's a bit of an old-timer, a single unit that does *everything*. If it hiccups, *everything* related to it hiccups. Plus, it's not playing nice with our plans for a super cool, modern setup.

So, our mission, should you choose to accept it (you kinda have to!), is to help other apps break up with Event Manager. We want them to find new, better ways to talk to each other. This guide is your map to understanding the old beast, what shiny new world we're building, and how we're gonna get there. Your brains and coding skills are what will make this whole thing a success!

2. Our Big Quest & What We Want You To Know

The Main Goal: We're sending the Event Manager on a long, permanent vacation! We'll help all the apps that depend on it switch to other, more direct, or new helper apps. This will make our whole system stronger, easier to tweak, and ready for the future.

What We Need You, Our Dev Heroes, To Get:

- **Know Thy Enemy (Just Kidding!):** Get a good grip on how Event Manager works right now – its tech, how messages flow, the whole shebang.
- **See the Future:** Understand the new, improved setup we're aiming for and why it's way better.
- **Learn the Game Plan:** Get familiar with the different ways we'll be moving apps off Event Manager.
- **Your Super Role:** Figure out where you fit into this grand adventure and what awesome stuff you'll be doing.

3. The Way Things Are Now: Meet Event Manager 🧐

Event Manager is an older Java app that’s been the main message handler for a while.

3.1. What It Does (Its Day Job)

- **Message Makeovers (Event Transformation):** It changes incoming messages so they look just right for the apps waiting for them. This magic is mostly done by Java code made by a tool called **Contivo**.
- **Directing Traffic (Event Routing):** It figures out where messages need to go – to another waiting line (a queue) or to a web service. **Apache Camel** is the tool that handles this route planning.
- **One Big Funnel (Centralized Processing):** It sips messages from a single "in-tray" (an input queue) where lots of different apps drop off their messages.
- **Keeping Receipts (Persistence):** It saves a copy of every message that comes in and every message that goes out. Good for "who sent what when" questions.

3.2. The Tech It's Built With

- **Brains & Bones:** Java and the Spring Framework.
- **GPS & Roadmaps:** Apache Camel (for figuring out message paths).
- **Translation & Formatting:** Castor (for XML stuff) and Contivo (for creating the code that changes messages).
- **Message Post Office:** IBM MQ (this is where all the message queues live).
- **The Engine Room:** Apache Tomcat (the server that runs the app).
- **The Ground It Stands On:** Red Hat Enterprise Linux (RHEL).
- **The Filing Cabinet:** A database (we'll confirm which one, but it's where it keeps those message receipts).

3.3. How It All Works: A Bird's-Eye View

1. **Apps Send Stuff (Producers):** Various apps (we call them Producers) send their messages to **one main Event Manager Input MQ** (think of it as Event Manager's main mailbox).
2. **Event Manager Grabs the Mail:** The Event Manager app, running on its Tomcat server, keeps an eye on this mailbox and picks up new messages.
3. **Inside Event Manager's Brain:**
    - **Save a Copy (Inbound):** First thing, it saves the raw incoming message to its database.
    - **Decisions, Decisions:**
        - **"Just Pass It On" (Simple Pass-Through):** Sometimes, a message doesn't need changing. Event Manager just sends it straight to the right **Consumer's MQ** (the recipient's mailbox). The app waiting for it just picks it up there.
        - **"Change It, Then Send It" (Transform and Route):** For other messages:
            1. It uses those **Contivo-made Java bits** to give the message a

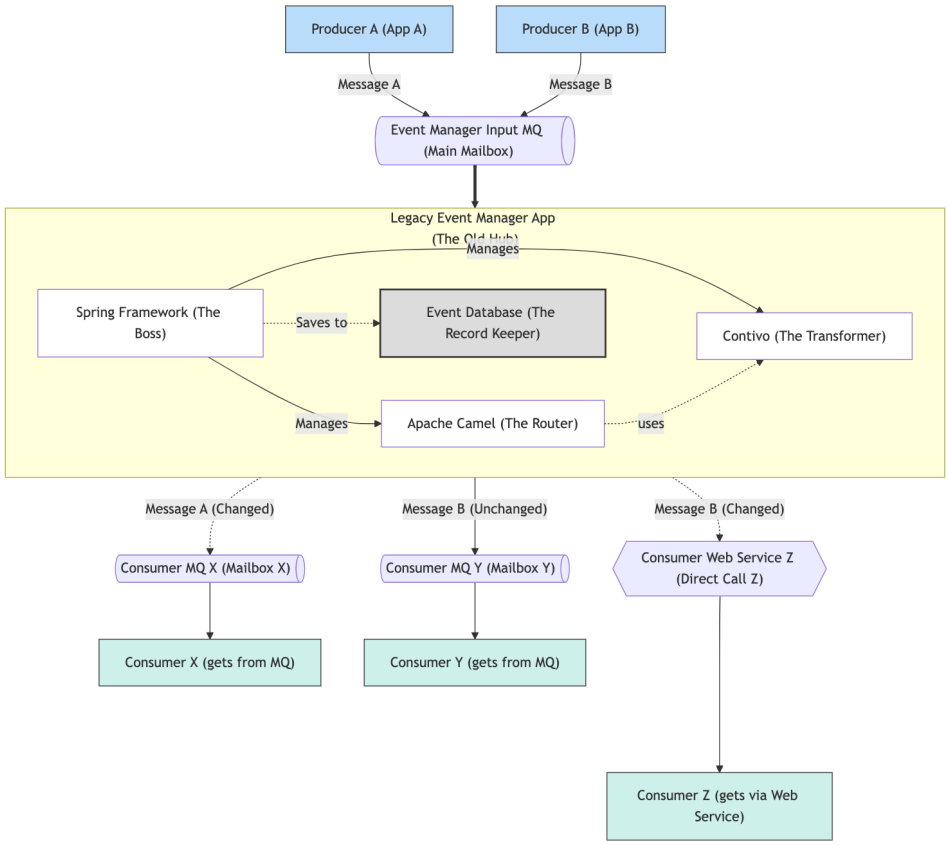
makeover.
2. Then, using its **Apache Camel roadmaps**, it sends the newly-changed message either to a **Consumer's MQ** or calls a **Consumer's Web Service** directly.

- **Save Another Copy (Outbound):** The message it's sending out (whether it was changed or not) also gets saved to the database.

4. **Apps Get Their Mail (Consumers):** The apps waiting for these messages (we call them Consumers) get them from their own MQs or when Event Manager calls their web service.

3.4. The Current Setup (Picture Time!)

Here's what it looks like now. Don't worry, you don't have to draw it, just get the gist!



Caption for the Diagram Above: Think of "Event Manager" as a big, central post office. Producers A and B drop all their mail into one big "Input MQ" box. The Event Manager sorts it, sometimes changes the address or repackages the mail (transforms it), and then sends it to specific "Consumer MQ" boxes or directly to a "Web Service" address. It also keeps a copy of everything in its "Event Database."

3.5. Where It All Lives (Infrastructure)

- **The Computers:** Red Hat Enterprise Linux servers.
- **The App's Home:** Apache Tomcat runs the Event Manager app.
- **The Message Pipes:** IBM MQ is used for all the message queues.

3.6. Why This Old Way is a Bit of a Headache ⚠️

- **One Hiccup, Big Problems (Single Point of Failure):** Because Event Manager is one big unit, if it has a bad day and stops working, *all* message traffic stops. Not good.
- **Doesn't Fit Our New Style:** We're aiming for a world of smaller, independent services (think microservices). Event Manager is like a giant mansion in a neighborhood of tiny houses – it just doesn't fit.
- **Changing Anything is a Big Deal (Deployment Rigidity):** Need to change how one type of message is handled? You have to update and restart the *entire* Event Manager. It's like repainting your whole house just to change the color of the mailbox. Slow and risky!
- **Getting Old and Cranky (Maintenance & Obsolescence):** Keeping old software bits (Java, Spring, etc.) up-to-date is tough and can leave security holes. Trying to upgrade them in this big app is like performing surgery on a dinosaur.
- **Can't Grow Smart (Scalability Issues):** If one message type gets super busy, you can't just give *that part* more power. You have to try and boost the whole Event Manager, which isn't efficient.
- **Too Much Stuff in One Place (Complexity):** Over years, tons of different rules for different messages have been crammed into Event Manager. It's now a bit of a maze to figure out, fix, or change.

4. The Shiny Future: Breaking Free! 🌟

Our main aim is to let the Event Manager retire gracefully. We'll do this by letting apps talk more directly or through new, smaller helper services.

4.1. Our New Rules of the Road

- **Spread Things Out:** No more one giant app controlling everything.
- **Simple Connections:** Apps should connect directly if it makes sense, or through simple, dedicated helpers.
- **Keep Consumers Happy:** We'll try super hard not to make the apps *receiving* messages change how they work (like listening to a new MQ or expecting a different web call).
- **Don't Reinvent the Wheel:** If some of that old Contivo message-changing code is still good, we'll try to reuse it.

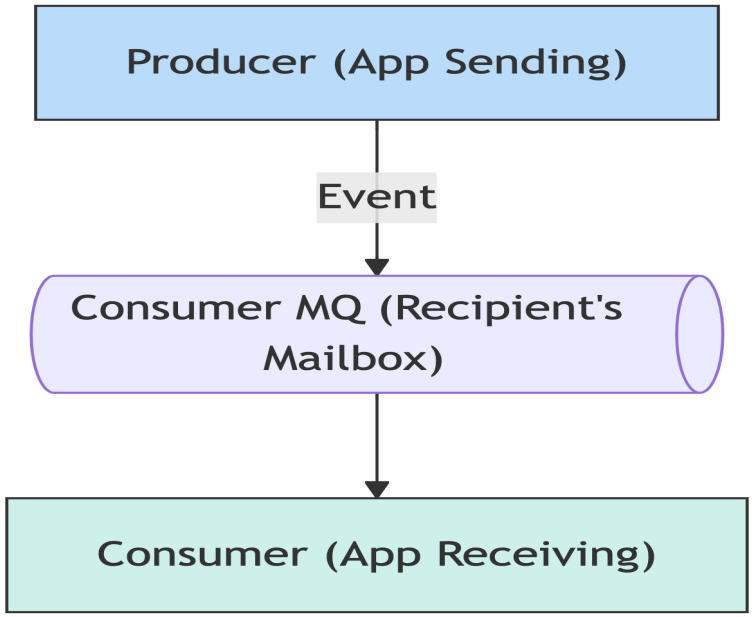
- **Hello, Modern Tech!:** We'll use newer tools like Kafka where it's a good fit and helps us build for the future.
- **Change and Grow Easily:** Each message path or its helper apps should be changeable and scalable on its own.

4.2. How We'll Get There: Our Migration Recipes

We've got a few different recipes, depending on what the Event Manager is doing for each message flow.

Recipe 1: The "No Change, Just Send" Flows (Direct MQ Routing)

- **When to Use:** When Event Manager just passes a message from its input MQ to a consumer's MQ without changing it at all.
- **The Fix:** The app sending the message (Producer) will just send it straight to the Consumer's MQ. Event Manager gets cut out of the loop. Easy peasy!
- **Picture It:**

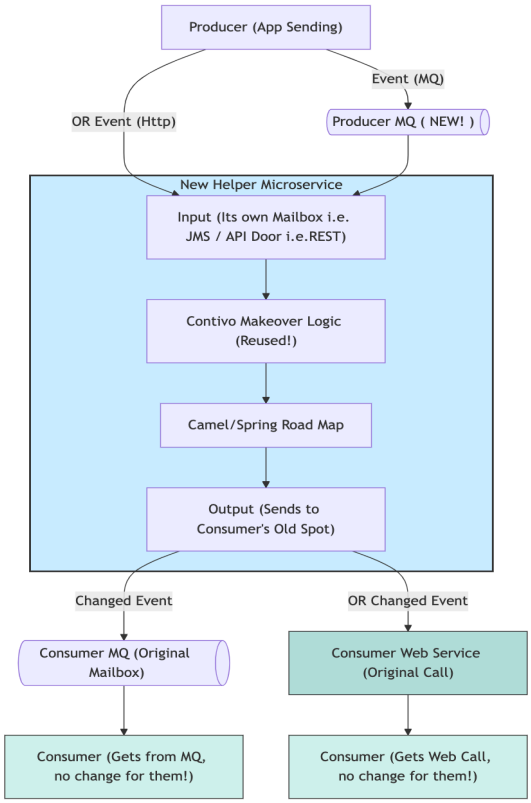


- **Why It's Cool:**
  - The Consumer app doesn't even notice a change!
  - Super simple to do.
  - One less stop for the message, so it might even be faster.
- **Your Job, Should You Choose to Code It:**
  - Find these "pass-through" message flows.
  - Help the Producer app teams change their settings to send to the right Consumer MQ.

- Test it all works.

Recipe 2: The "Change It, Then Send It" Flows (New Little Helper App)

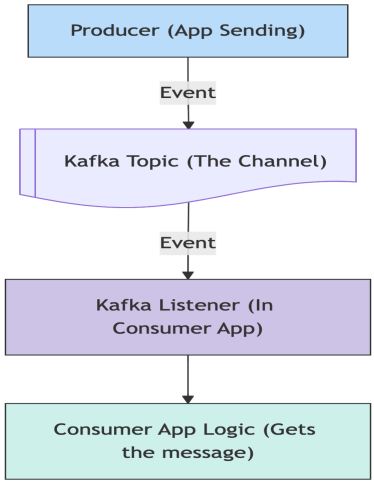
- **When to Use:** When Event Manager changes a message (using Contivo) and *then* sends it to a Consumer's MQ or web service.
- **The Fix:** We'll build a brand new, small, dedicated app (let's call it an "Intermediate App" or "Helper App") just for this one message flow (or a small group of similar ones).
  - **Tech for the Helper App:** Probably a Spring Boot app (great for small services).
  - **What It'll Do:**
    1. Get messages from the Producer (either from a new, dedicated MQ for this helper, or the Producer will call a REST API on this helper).
    2. It will reuse the *exact same Contivo-made Java code* that Event Manager used to change the message. (We'll package this code up like a library).
    3. It will use Apache Camel (or maybe something simpler in Spring) to send the changed message to the *original* Consumer's MQ or call the *original* Consumer's web service.
- **Picture It:**



- **Why It's Cool:**
  - Again, the Consumer app doesn't have to change a thing!
  - We reuse the Contivo code we know already works.
  - Each helper app is small and independent. If one has an issue, others are fine. They can also grow (scale) on their own.
  - It's clear who owns which bit of logic.
- **Your Job, Should You Choose to Code It:**
  - Build these new Spring Boot helper apps.
  - Plug in the old Contivo code.
  - Set up the Camel/Spring routes.
  - Create the "front door" for these apps (MQ listener or REST API).
  - Test, test, test that the message makeovers and sending work perfectly.

Recipe 3: The "Let's Use Kafka!" Flows (For the Modern Folks)

- **When to Use:** If an app sending messages (Producer) already uses Kafka (a cool, modern message streaming system) or wants to, AND the app receiving messages (Consumer) can also use Kafka (or we can help it).
- **The Fix:**
  1. The Producer sends its messages to a specific "channel" in Kafka (called a Kafka topic).
  2. The Consumer app gets a new part (or we build one for it) that listens to this Kafka channel.
  3. If the message needs changing, this can happen:
     - The Producer changes it *before* sending it to Kafka.
     - The Consumer changes it *after* getting it from Kafka.
     - Or, for fancy stuff, a special Kafka Streams app could do it in the middle.
- **Picture It:**



- **Pros (The Good Stuff):**
  - Super modern and can handle tons of messages. This is where the cool kids are heading.
  - Kafka is built to be fast and not lose messages.
- **Cons (The Tricky Bits):**
  - Both the sending and receiving apps need to change.
  - Might need to learn some new Kafka skills.
- **Your Job, Should You Choose to Code It:**
  - Help build the Kafka listeners for Consumer apps.
  - Maybe help Producer teams set up their Kafka sending part.
  - Add any message-changing logic needed in this new Kafka flow.

4.3. Why This Future Will Be So Much Better ✅

- **No More Single Crash Point:** If one small helper app has an issue, the rest of the system keeps humming. Much safer!
- **Quicker Changes, Faster Releases:** Need to tweak one flow? Just update its small app. No more re-deploying a giant beast.
- **Easier to Look After:** Small, focused apps are simpler to understand, fix, and upgrade.
- **Fresher Tech:** We get to use newer Java, Spring Boot, and maybe Kafka. Yay, new toys!
- **Grow Just What You Need:** If one message flow gets super busy, just give its helper app more resources. Smart!
- **Everyone Knows Their Job:** It'll be much clearer which app is responsible for what.
- **Fits the Big Picture:** This all lines up with our company's grand plan for how apps should be built.

5. Your Part in This Epic Tale (Roles, Responsibilities, and How We'll Do It)

You, our Java developers, are the stars of this show! Here’s the kind of awesome stuff you'll be doing:

- **Detective Work & Blueprints (Analysis & Design):**
  - Dig into how Event Manager currently handles messages – look at its Camel routes and Contivo setups.
  - Help design the new little helper apps (Recipe 2).
  - Help decide which recipe is best for each old message flow.
- **Building Cool Stuff (Development):**
  - Tweak Producer apps so they can send messages directly via MQ (Recipe 1).

    - Build those new Spring Boot helper apps, adding in the old Contivo code and new Camel/Spring routes (Recipe 2).
    - Create Kafka listeners for Consumer apps and maybe help Producer teams with their Kafka sending bits (Recipe 3).
- **Making Sure It Works (Testing):**
    - Write good unit tests for all your new code.
    - Do integration tests to see if messages flow correctly from start to finish.
    - Help out when the business folks and Consumer app teams do their own testing (UAT).
- **Launching It (Deployment & Cutover):**
    - Get your code ready to go live.
    - Be there during the launch when we switch over from the old way to the new way.
- **Leaving Good Notes (Documentation):**
    - Write down how your new services work and any changes you made.
- **Teamwork Makes the Dream Work (Collaboration):**
    - Chat and work with other devs, architects, testers, and the ops team.
    - Talk to the Producer and Consumer app teams to make sure everyone's on the same page for changes and testing.

Our General Game Plan:

1. **List and Rank:** We'll make a big list of all the message flows Event Manager handles. Then we'll decide which ones to tackle first based on how important they are, how tricky they look, and any risks.
2. **One Bite at a Time:** We'll move these flows over bit by bit, not all at once in a scary "big bang."
3. **Test Drive:** We'll probably pick a couple of typical flows to try out our recipes on first. This helps us smooth out any wrinkles in our plan.
4. **Build, Test, Repeat:** For new helper apps, we'll use agile methods – building a bit, testing it, getting feedback, and then building the next bit.
5. **Keep Talking:** We'll have regular team chats, updates, and share what we're learning.

6. Your Superhero Toolkit (Tech & Skills)

Here’s the tech you'll be using or maybe learning a bit more about:

- **Java & Spring Power:** You're already good here! Java, Spring Boot, Spring Framework (all its useful bits like Core, MVC, Data, Security), and Spring Integration.
- **Apache Camel Know-How:** Understanding the old Camel routes is key. For new

helper apps, Camel or Spring Integration will be your friends.
- **Contivo (The Old Transformer):** We want to reuse the *Java classes* Contivo made, not make new Contivo stuff. Knowing a bit about how it worked helps.
- **IBM MQ Smarts:** You'll need to know about MQ, how it works, and how to use it from Java (JMS).
- **RESTful API Skills:** Designing and building APIs with Spring Boot.
- **Apache Kafka (for Recipe 3):** Getting to know Kafka (topics, consumers, producers) and how to use it with Java.
- **Building & Shipping Tools:** Maven/Gradle, Jenkins, Git – the usual suspects.
- **Testing Tools:** JUnit, Mockito, Spring Test.
- **Containers (Maybe Later?):** Docker, Kubernetes – as OpenShift Cloud Platform (OCP) is the future.

7. What's Next? Let's Do This!

1. **Read This (Duh!):** Go through this guide. Hopefully, it's clearer now!
2. **Peek at the Old Code:** Start looking at the Event Manager code. Focus on its Camel routes and how it uses those Contivo-made classes.
3. **Got Questions?** Jot them down. No silly questions! Hahaha, just kidding ;)
4. **Team Huddle!** We'll have a project kick-off meeting soon to chat about all this, give out first tasks, and answer your questions.

This is a big, important project, and it's going to make our systems way better. Your skills and enthusiasm are what will make it happen. We're excited to get started with you! Let's make some magic (and write some great code)!