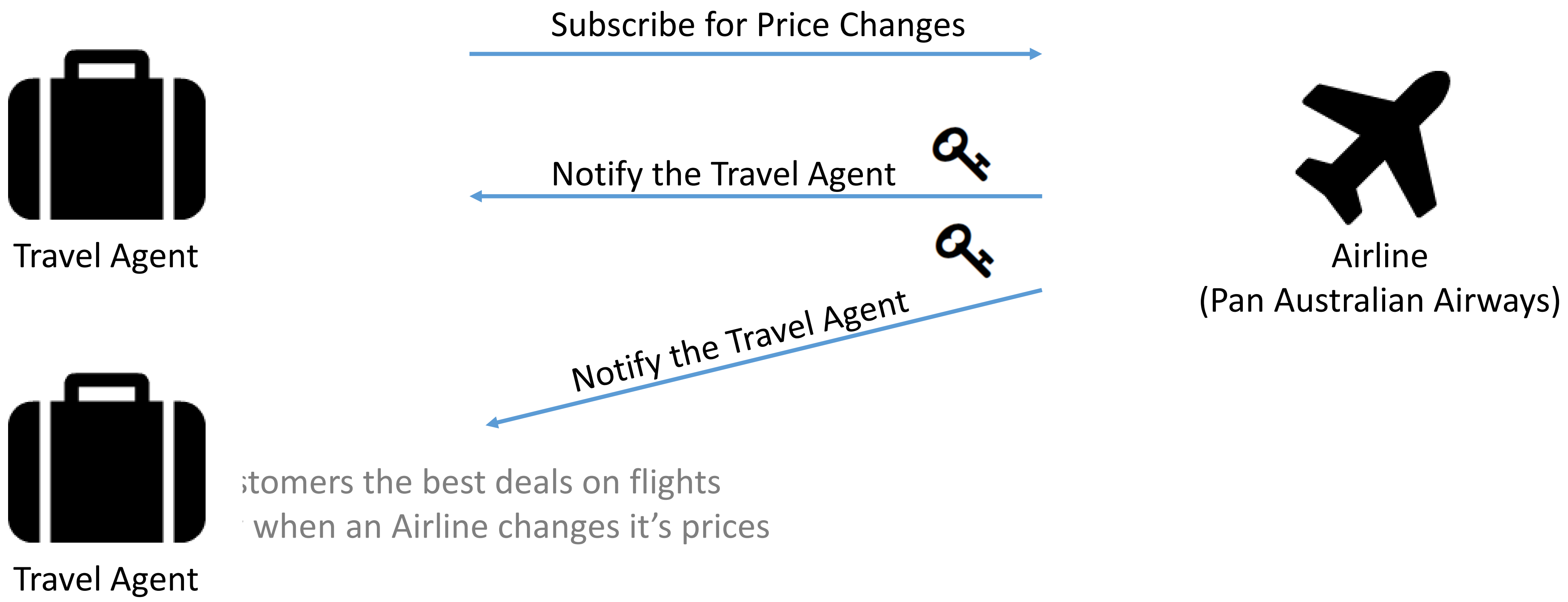
Webhooks

Solution build with .NET 5 & RabbitMQ

Demo

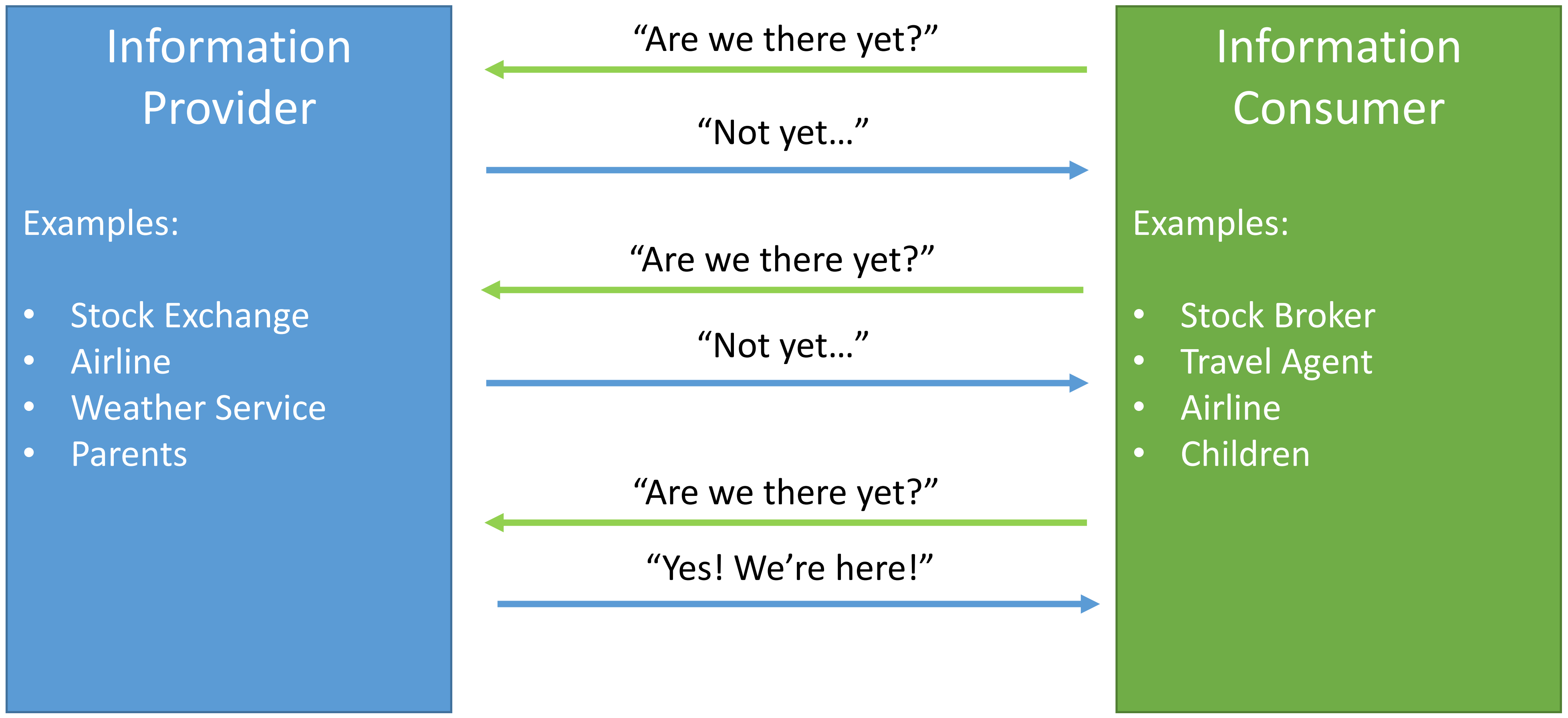
Demo Use Case / Context



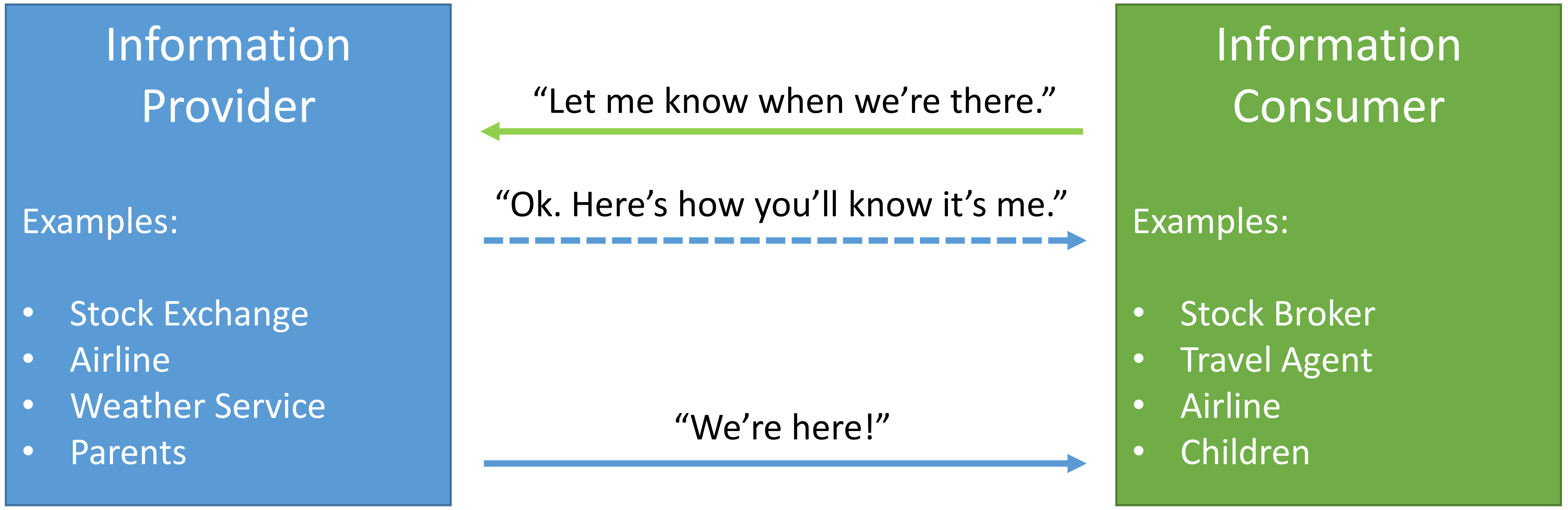
;tomers the best deals on flights
' when an Airline changes it's prices

What are Webhooks?

Alternative Pattern: Pull / Polling



Webhooks: Push / Notifications



In a technical context

- No real “standard”
- Information Consumer should provide a HTTP POST endpoint
- Registration of Webhooks can be any mechanism
  - HTTP POST Endpoint
  - SOAP
  - REST
  - Web Page (Form)

What we'll cover (& what you'll learn)

INTRODUCTION

- What we'll build (Demo)
- What are Webhooks?
- Course Structure
- Architecture Overview
- Ingredients / Tooling

ENVIRONMENT SET UP

- Solution Set Up
- RabbitMQ (Docker)
- SQL Server (Docker)
- VS Code Plugins

AIRLINE APPS Pt.1

- Webhook API
- Flight API
- Web Front End

TRAVEL AGENT APP

- Webhook POST Endpoint

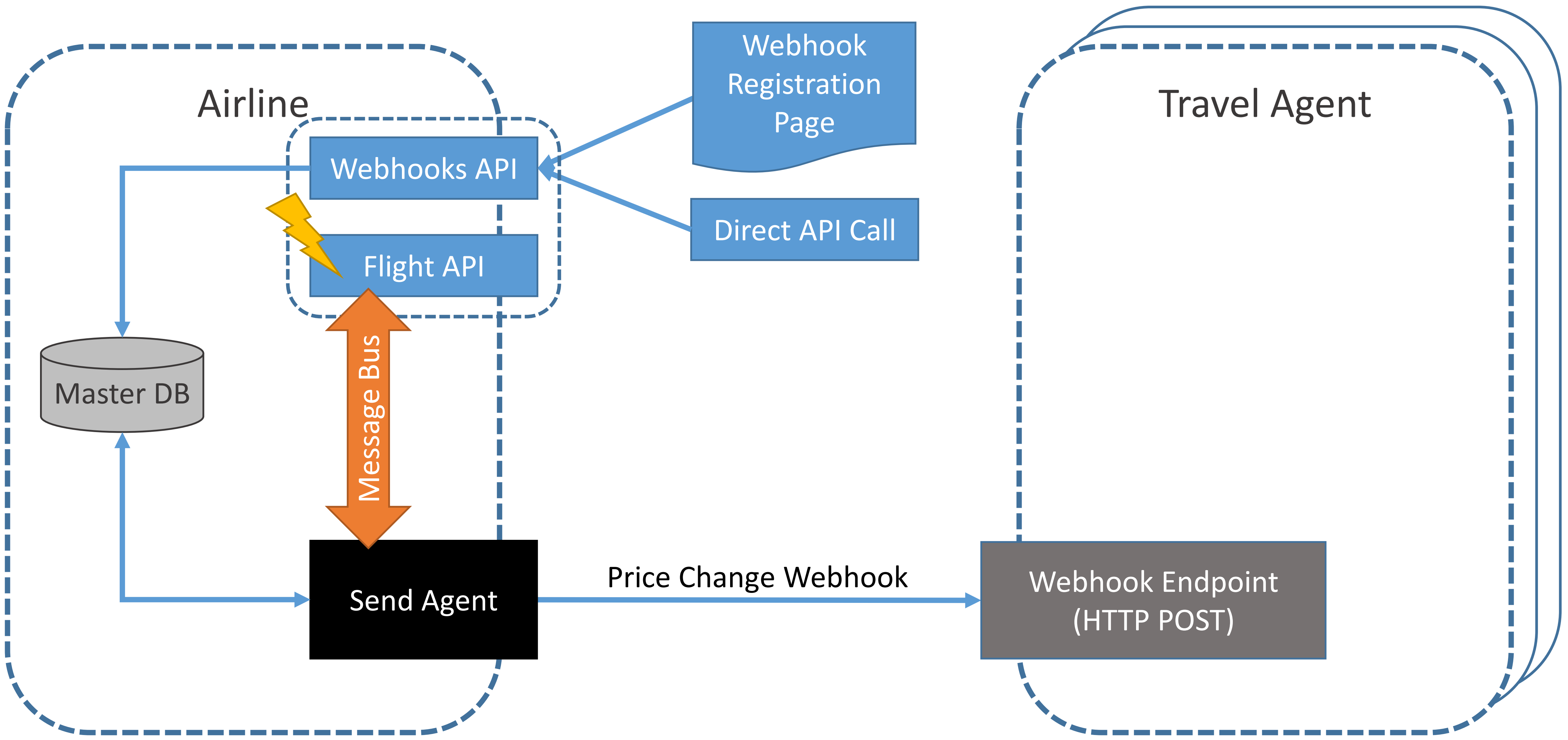
AIRLINE APPS Pt. 2

- Send Agent
- Set Up DI
- HttpClientFactory
- RabbitMQ (Pub & Sub)

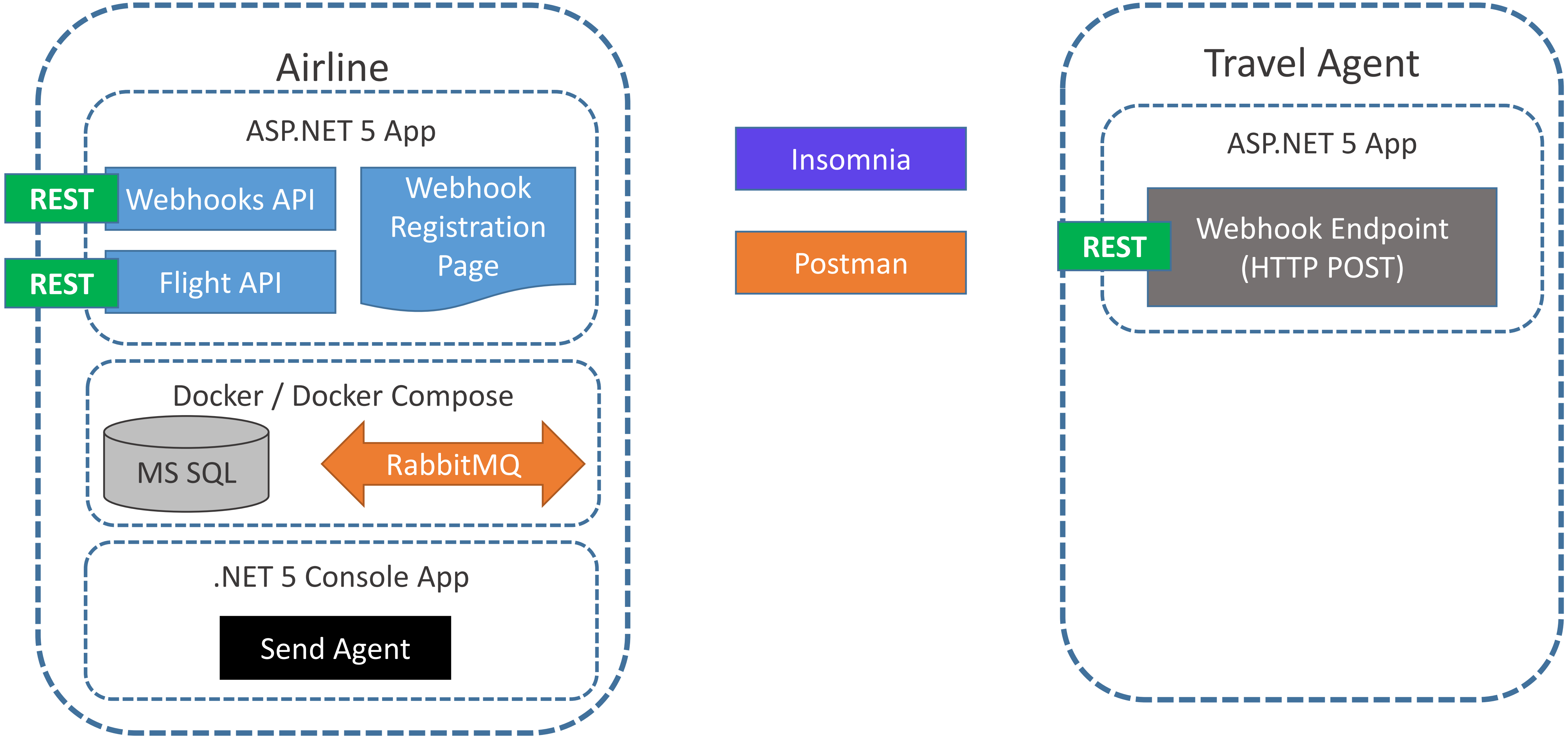
FINAL THOUGHTS

- End to End Testing
- Final Thoughts
- Credits

Solution Architecture



High-Level Application Architecture



Ingredients

- VS Code Text Editor (free)
- .NET Core 3.1 or .NET 5 SDK (free)
- Docker / Docker Compose (free)
  - RabbitMQ / SQL Server Express (free)
- API Client, e.g. Postman/ Insomnia / Curl (free)
- Web Browser (free)

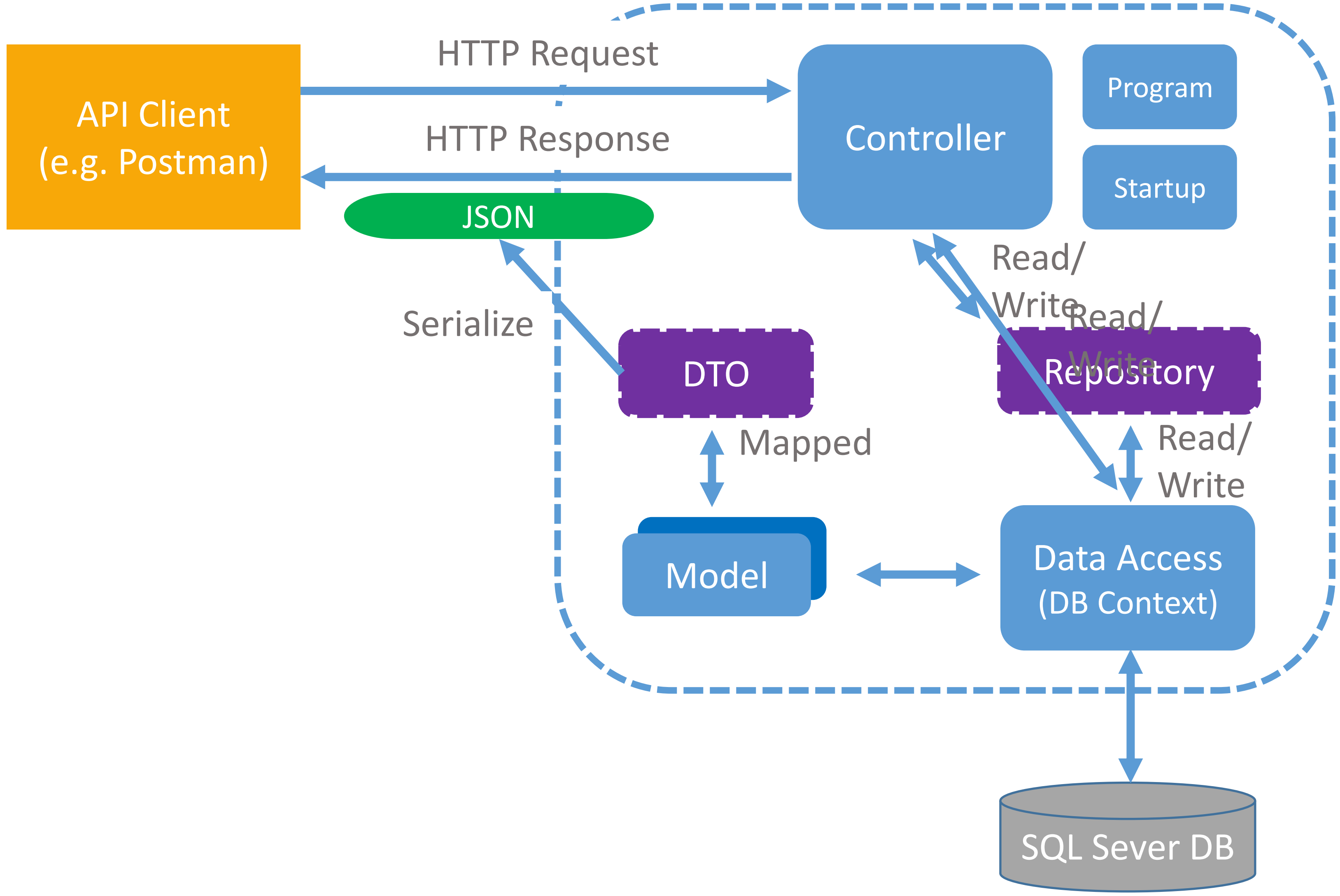
We cover everything step by step, but…

You should have an understanding of:

- ASP.NET (Core) REST APIs
- DB Context / Entity Framework
- Docker / Docker Compose

Links to my other videos in the description below

API Application Architecture



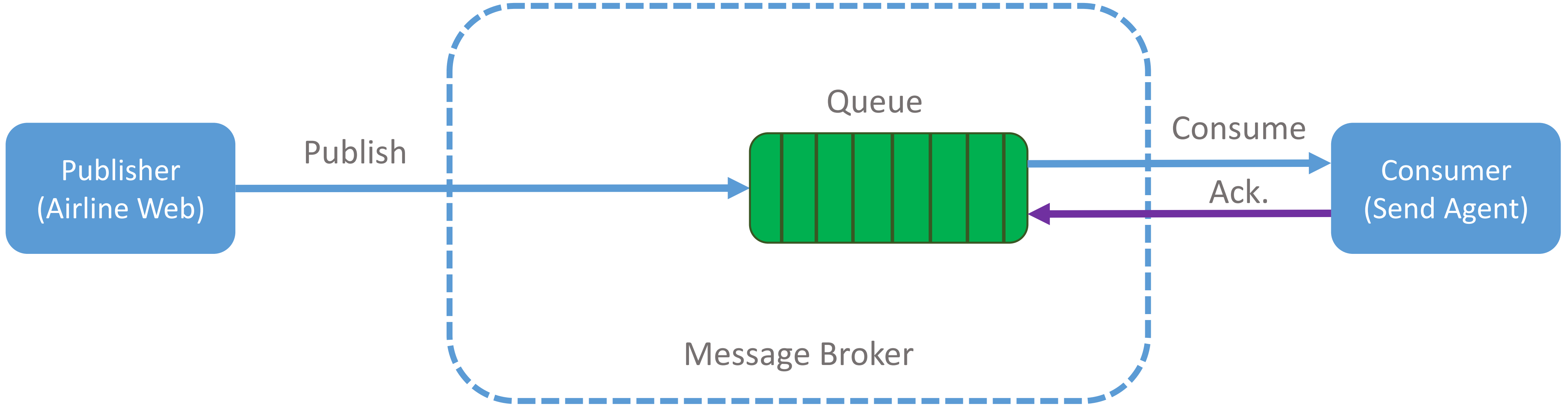
RabbitMQ

Overview

What is RabbitMQ

- A Message Broker – it accepts and forwards messages
- Messages are sent by Producers (or Publishers)
- Messages are received by Consumers
- Messages are stored on Queues (essentially a message buffer)
- Exchanges can be used to add “routing” functionality
- Uses Advanced Message Queuing Protocol (AMQP) & others
  - Messages sent as a Byte Array

RabbitMQ Direct Queue



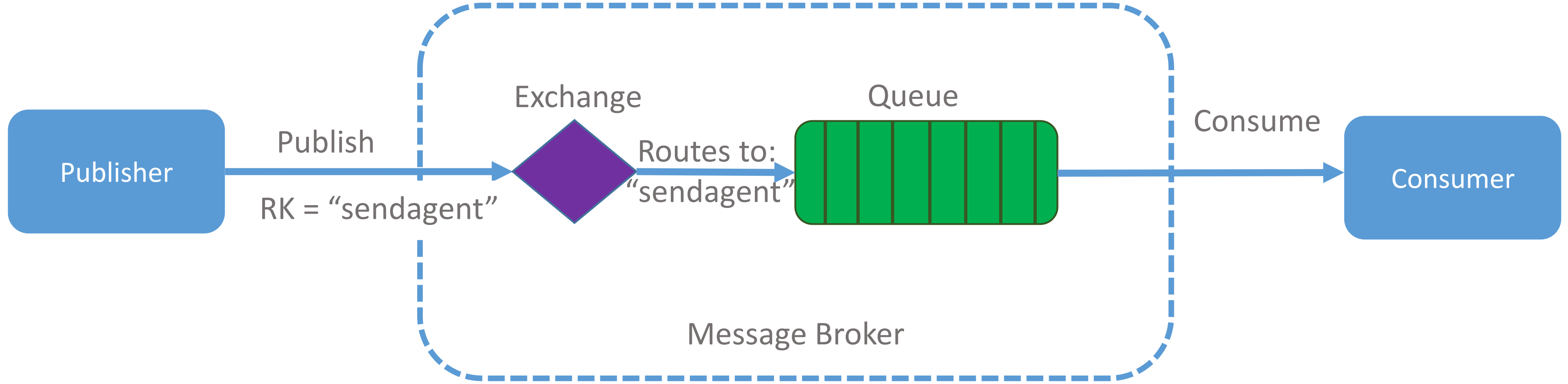
RabbitMQ

Exchanges

4 Types of Exchange

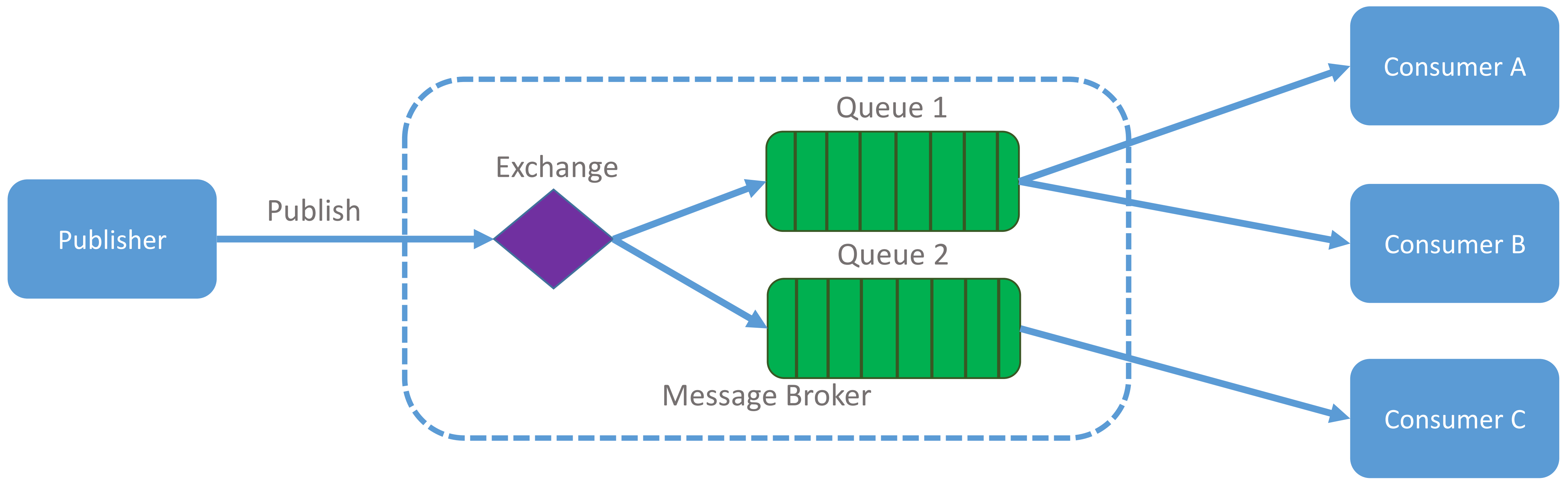
- Direct Exchange
- Fanout Exchange
- Topic Exchange
- Header Exchange

RabbitMQ Direct Exchange



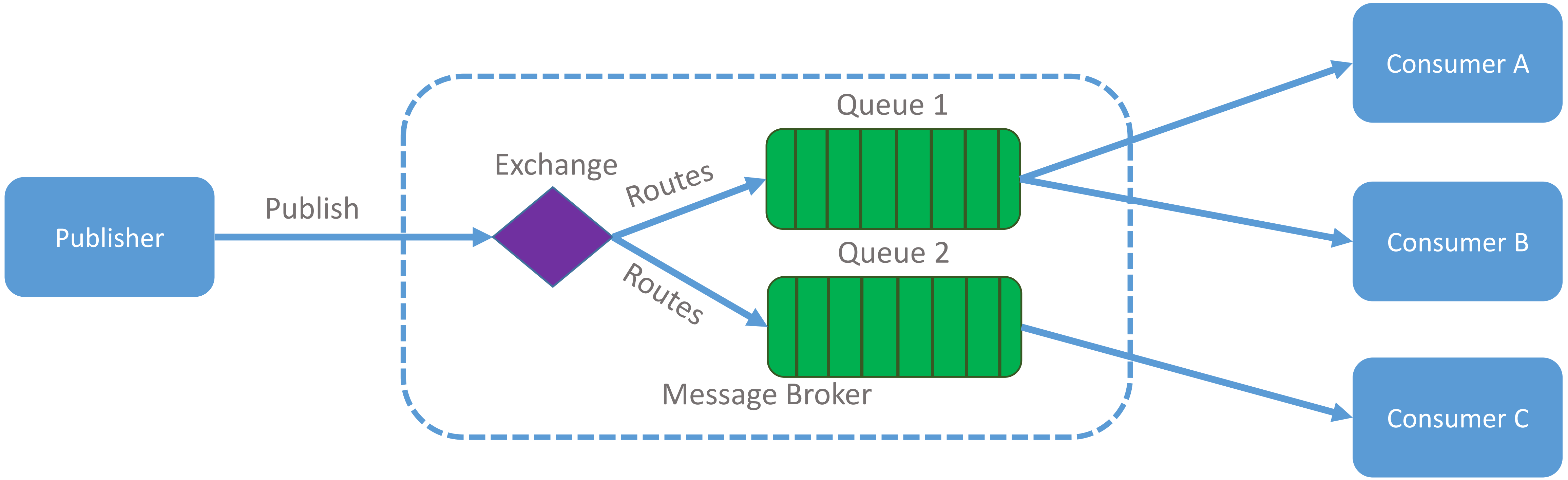
- Delivers Messages to queues based on a routing key
- Ideal for “direct” or unicast messaging

RabbitMQ Fanout Exchange



- Delivers Messages to all Queues that are bound to the exchange
- It ignores the routing key
- Ideal for broadcast messages

RabbitMQ Topic Exchange



- Routes messages to 1 or more queues based on the routing key (and patterns)
- Used for Multicast messaging
- Implements various Publisher / Subscriber Patterns