# ADR-002: استفاده از الگوی Outbox/Inbox برای تضمین تحویل پیام‌ها

## وضعیت (Status)

Accepted

## زمینه و مشکل (Context)

در معماری میکروسرویس‌های ما، سرویس‌ها برای تبادل داده به صف پیام (Message Queue) متکی هستند.   
الزامات بیزینسی به وضوح بیان می‌کنند که:  
  
1. هیچ پیامی نباید از دست برود – تضمین شود که پیام‌ها همیشه روی صف ذخیره می‌شوند (at least once delivery).  
2. پیام نباید دوباره پردازش شود – گیرنده فقط یک‌بار پیام را دریافت کند (exactly once processing).  
  
در صورتی که مستقیماً پیام را به صف ارسال کنیم، مشکلات زیر پیش می‌آید:  
- خطر از دست‌رفتن پیام در صورت crash بین تراکنش دیتابیس و ارسال به صف.  
- امکان پردازش دوباره پیام‌ها توسط مصرف‌کننده (consumer) در شرایط خطا یا retry.

## تصمیم (Decision)

ما از الگوی Outbox/Inbox استفاده می‌کنیم:  
  
- Outbox Pattern:  
 - هر پیام به‌جای ارسال مستقیم، در جدولی به نام Outbox در همان تراکنش دیتابیس بیزینسی ذخیره می‌شود.  
 - یک پردازشگر (Outbox Processor) پیام‌ها را از جدول Outbox می‌خواند و در صف پیام منتشر می‌کند.  
 - این کار تضمین می‌کند که پیام هیچ‌وقت بین عملیات دیتابیس و ارسال به صف از دست نمی‌رود.  
  
- Inbox Pattern:  
 - در سمت مصرف‌کننده، همه پیام‌های دریافتی قبل از پردازش در جدولی به نام Inbox ذخیره می‌شوند.  
 - اگر پیام تکراری دریافت شود (به‌خاطر retry یا خطا)، با بررسی کلید پیام در Inbox متوجه می‌شویم که قبلاً پردازش شده و دوباره اعمال نمی‌شود.  
 - این کار تضمین می‌کند که هر پیام فقط یک‌بار پردازش شود.

## پیامدها (Consequences)

مزایا:  
- تضمین عدم از دست رفتن پیام‌ها (reliable delivery).  
- تضمین پردازش یکتا (idempotency و exactly-once).  
- هماهنگی با الگوی Transactional Outbox → تراکنش دیتابیس و تولید پیام همیشه اتمیک هستند.  
- امکان مانیتورینگ و replay پیام‌ها از جدول Outbox در شرایط خطا.  
  
معایب / چالش‌ها:  
- نیاز به ایجاد و نگهداری جدول‌های Outbox و Inbox در دیتابیس.  
- افزایش سربار ذخیره‌سازی و مدیریت پیام‌ها در دیتابیس.  
- نیاز به مکانیزم پاک‌سازی (cleanup) برای جلوگیری از رشد بی‌نهایت جداول Outbox/Inbox.

## گزینه‌های بررسی‌شده (Alternatives / Considered Options)

1. ارسال مستقیم به صف (بدون Outbox/Inbox):  
 - سادگی بیشتر، اما خطر از دست رفتن پیام در صورت crash.  
 - عدم تضمین پردازش یکتا توسط مصرف‌کننده.  
 - رد شد.  
  
2. تکیه بر قابلیت‌های Queue (مثل Kafka Exactly Once Semantics):  
 - Kafka و برخی سیستم‌ها مکانیزم‌هایی برای Exactly-Once دارند.  
 - ولی وابستگی به تکنولوژی خاص ایجاد می‌کند و همیشه در محیط‌های عملیاتی (مانند RabbitMQ یا Azure Service Bus) در دسترس نیست.  
 - رد شد به دلیل نیاز به vendor independence.

## ارجاعات (References)

- Transactional Outbox Pattern – microservices.io  
- Inbox Pattern – microservices.io  
- نتایج تست داخلی تیم (Doc-POC-2025-01)