**📜 Manifesto Asgard.RabbitMQ**

**🎯 Obiettivo**

**Fornire un’implementazione concreta di IEventPublisher e IEventSubscriber basata su RabbitMQ, conforme ai contratti di Asgard.Abstraction, che:**

* **pubblichi e consumi eventi tipizzati tramite CloudEvent**
* **gestisca in modo trasparente retry, delay e DLQ**
* **configuri la topologia (exchange, queue, binding) all’avvio**
* **mantenga separazione totale tra logica applicativa e semantica broker-specifica**

**📦 Requisiti funzionali**

**Il pacchetto Asgard.RabbitMQ deve supportare i seguenti scenari operativi:**

**🔹 Pubblicazione eventi**

* **CloudEvent come formato obbligatorio per i messaggi pubblicati.**
* **Supporto alla pubblicazione:**
  + **con o senza routing key**
  + **su exchange di tipo direct, topic o fanout**
* **L’utente può inviare eventi attraverso un IEventPublisher in modo agnostico, ma il publisher RabbitMQ deve mappare correttamente routingKey e exchange type in base alla configurazione.**

**🔹 Sottoscrizione eventi**

* **Possibilità di consumare da una coda:**
  + **con o senza routing key**
  + **con gestione multipla di routing key sulla stessa coda (es. #, \*.event.\*, ecc.)**
* **Il subscriber usa ICloudEventDispatcher per invocare in modo dinamico gli ICloudEventHandler<T>, anche in scenari multi-routingKey.**

**🔹 Gestione dei retry**

* **I retry sono configurabili per:**
  + **routing key singola**
  + **routing key multiple**
  + **assenza di routing key**
* **I messaggi falliti vengono inviati a una retry queue con TTL e reinoltrati dopo un delay.**
* **Al superamento del numero massimo di tentativi, il messaggio viene inviato a una DLQ.**
* **Il numero di tentativi è tracciato in CloudEvent.Extensions["x-retry-count"].**

**🔹 Configurazione topologica**

* **L’intera topologia (exchange, queue, routing key, policy retry/DLQ) è definita in appsettings.json, tramite:**
  + **RabbitMqOptions per connessione, exchange e tipo**
  + **RabbitMQConfiguration (o simili) per ogni subscription specifica**
* **La configurazione è:**
  + **singola, se la subscription gestisce una sola routing key**
  + **multipla, se gestisce più routing key**
  + **neutrale, se lavora senza routing key**

**🧱 Componenti principali**

| **Componente** | **Ruolo** |
| --- | --- |
| **RabbitPublisher** | **Pubblica eventi su RabbitMQ secondo le convenzioni definite** |
| **RabbitEventSubscriber** | **Sottoscrive eventi e li inoltra a ICloudEventDispatcher** |
| **RabbitMqTopologyInitializer** | **Crea e configura la topologia completa (exchange, queue, DLQ, retry)** |
| **RabbitMqEventStream** | **Loop asincrono che legge i messaggi da RabbitMQ** |
| **RabbitMqEnvelope** | **Rappresenta il messaggio raw RabbitMQ + metadati CloudEvent** |
| **RabbitMqConsumerContext** | **Informazioni contestuali interne alla ricezione di un messaggio** |

**🧩 Garanzie implementative**

| **Garanzia** | **Descrizione** |
| --- | --- |
| **🔄 Mappatura completa del CloudEvent** | **Converte ogni CloudEvent in messaggio RabbitMQ con headers, body e properties coerenti** |
| **♻️ Gestione integrata di retry e DLQ** | **Supporto nativo per x-retry-count, retry queue con TTL e DLQ finale, secondo configurazione** |
| **🧼 Nessun leakage di Rabbit verso l’esterno** | **Tutta la semantica broker-specific resta incapsulata nel pacchetto** |
| **🧪 Serializzazione centralizzata** | **Usa sempre ICloudEventSerializer per codifica e decodifica dei payload** |
| **🏗️ Topologia autodichiarativa** | **Tutte le code, exchange e binding sono configurati esplicitamente all’avvio** |
| **🧠 Estensioni nel CloudEvent.Extensions** | **I metadati di messaggio (retry count, errori, origini) sono isolati nel dizionario Extensions** |
| **⚙️ Completamente configurabile via DI** | **Tutti i parametri sono impostabili tramite RabbitMqOptions e RabbitMQConfiguration** |

**⚒️ Uso atteso**

* **Il publisher viene registrato tramite AddRabbitMq(...) e pubblica eventi via IEventPublisher.**
* **Il subscriber attiva code RabbitMQ e inoltra i messaggi a uno o più ICloudEventHandler<T>.**
* **Retry e DLQ sono trasparenti: l'utente non ha bisogno di gestirli manualmente.**
* **Le queue vengono dichiarate automaticamente in startup tramite RabbitMqTopologyInitializer.**

**📦 Estendibilità**

* **La logica di serializzazione può essere sovrascritta registrando un custom ICloudEventSerializer.**
* **È possibile creare versioni custom di RabbitPublisher e RabbitEventSubscriber mantenendo la stessa interfaccia.**
* **Il pacchetto è compatibile con test di integrazione via Testcontainers.NET.**