Sequence Diagram

External Synchronization Protocol and Vault (Sequences & Lifecycles)

The comprehensive technical document for architectural design and code structure

Engineering & Operations Team (WSS)

Technical Application Profile

Project : NotificationPublisher Ecosystem

Paths (Endpoints) : Email (Single/Bulk), SMS, WhatsApp, FCM

Data protection (Secrets) : HashiCorp Vault KV v2

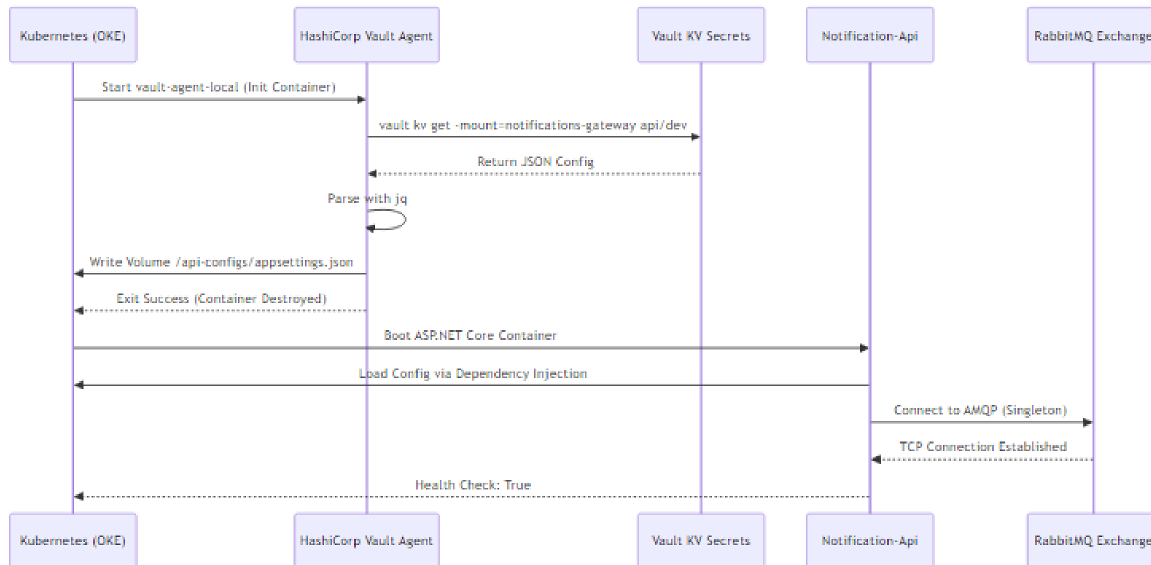
Implementation stage : Production - OKE Cluster

Core Tech : ASP.NET Core 8.0, RabbitMQ

---

1. Vault Bootstrap & Injection sequence

The sequence explains how applications get their secrets encrypted before they work:



Vault injection sequence in a Kubernetes environment

Detailed sequence steps

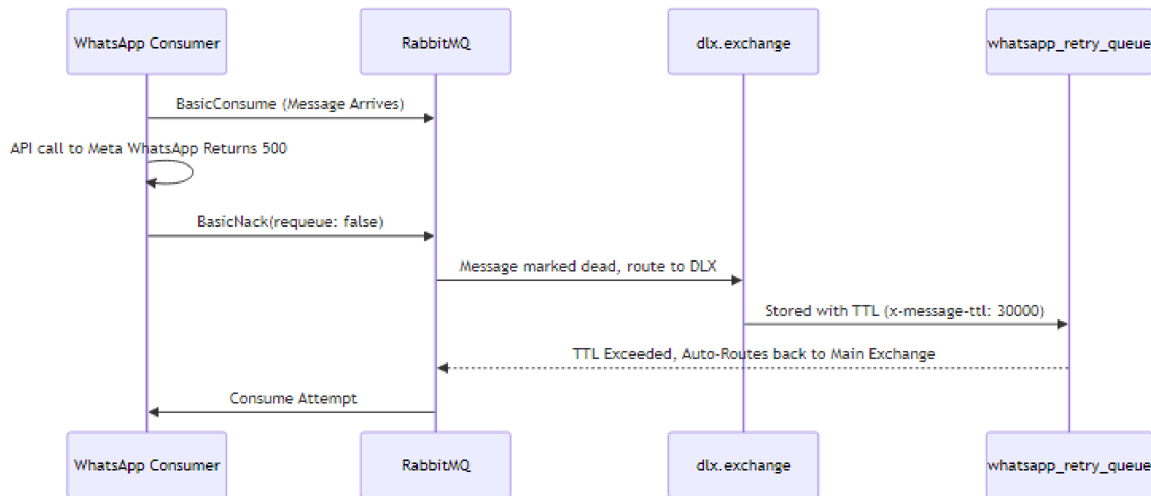
the details	Procedure	actor	Step
Init Containers starts first	Start Pod	Kubernetes	1
Authentication via role_id and secret_id	AppRole Auth	Vault Agent	2
vault kv get -mount=notifications-gateway	KV Read	Vault Agent	3
Extract JSON and write appsettings	Template Render	Vault Agent	4
The file is placed in emptyDir (RAM)	Volume Write	Vault Agent	5
Agent exits with code 0	Exit	Vault Agent	6
The container rides the same size	Start	.NET Container	7
ASP.NET detects JSON automatically	IConfiguration	.NET Container	8

Security features

- Temporary secrets: files exist only in memory (emptyDir medium: Memory). They are never written to disk
- Pod-limited access: Each Pod gets a unique set of credentials
- Rotation Support: Renewing a Vault lease can update Secrets without restarting the Pod

2. Dead Letter Exchange (DLX) life cycle

When a consumer fails to deliver to a third party provider:



DLX sequence: Failed -> Nack -> Retry

DLX course details

Result	Procedure	actor	Step
Attempting to deliver to an external provider	Message processing	Consumer	1
HTTP 5xx, timeout, or network error	to fail	External Provider	2
Reject message with requeue=false	BasicNack	Consumer	3
The message goes to Dead Letter Exchange	DLX Routing	RabbitMQ Broker	4
The message is waiting for the duration of the attempt	TTL Wait	DLX	5
Forwarding the message to the original queue	Re-Routing	DLX	6
New delivery attempt	Retry	Consumer	7

3. BasicQos initialization and balanced distribution

```
channel.BasicQos(
    prefetchSize: 0,    // No limit on byte size
    prefetchCount: 15,  // Max 15 in-flight per consumer
    global: false       // Per-channel, not global
);
```

- prefetchSize = 0: No limit on the total byte size of unconfirmed messages
- prefetchCount = 15: Each consumer fetches a maximum of 15 messages. The medium stops transmitting until ACK
- global = false: The limit applies to each channel (each consumer) and not to all consumers

- With 3 consumer instances: a maximum of 45 messages in processing at the same time
- No Hunger: The broker alternates distributions among consumers who have available slots

4. Publisher Confirm Sequence

the details	Procedure	actor	Step
Activate publisher confirmations on the channe	ConfirmSelect()	API	1
Send message with DeliveryMode = 2	BasicPublish()	API	2
Write to disk /var/lib/rabbitmq/mnesia/	Disk Write	RabbitMQ	3
Send basic.ack to the publisher	Confirm	RabbitMQ	4
Block until confirmation or timeout	WaitForConfirms(5s)	API	5
Upon confirmation -> updateStatus	DB Update	API (OK)	6a
On timeout -> PushFail response code	Return Error	API (Fail)	6b

This Two-Phase Commit pattern ensures that the database state accurately reflects whether RabbitMQ has accepted responsibility for the message.