# Propuesta técnica para Ingestas: Conexión a la API de Alliance Warehouse (AWH)

**Autor:** Puma (Big Data)

**Objetivo:** Dejar listo, claro y ejecutable **cómo** conectar vuestras apps/servicios a la **API de Alliance Warehouse (AWH)** para consultar mensajes SWIFT, incluyendo **red**, **certificados**, **cifrado**, **cabeceras**, **paginación** y **pruebas con SoapUI**.

## 1) Resumen ejecutivo (para los que van con prisas)

* **Red:** Abrir firewall **saliente** hacia los balanceadores de **TEST** y **PROD** (host/puerto más abajo). IP de cliente obligatoria en la solicitud.
* **Seguridad:** Autenticación **mutua TLS (mTLS)** con **certificado cliente** (DER, SHA256withRSA) y \*\*cifrados TLS\_ECDHE\_RSA\_WITH\_AES\_\*\_GCM\*\*.
* **Certificados:**
  1. Generar/usar **JKS** de aplicación + **CSR** y firmar con **CA corporativa**.
  2. **Exportar** el cert **público** y enviarlo a Swift para alta.
  3. **Importar** el **cert público AWH** en vuestro truststore.
* **API:** REST. Cabeceras mínimas: Version, Application. Soporta **paginación/scroll**. Respuesta en **JSON**.
* **Pruebas:** Recomendado **SoapUI ≥ 5.7** importando el **YAML/OpenAPI** de AWH.

## 2) Red y firewall

### Entornos y balanceadores

**TEST / DEV** - Balanceador principal: warehouse\_api\_test.swift.corp → **180.18.249.15:48400** - Balanceador alternativo (en despliegue): warehouseapitest.swift.corp → **180.18.249.23:443**

**PRODUCCIÓN** - Balanceador principal: warehouse\_api.swift.corp → **180.18.248.27:48400** - Balanceador alternativo (en despliegue): warehouseapi.swift.corp → **180.18.248.34:443**

**Acción Ingestas** - Solicitar reglas de firewall **salientes** desde vuestra **IP de aplicación** (o rango) a los **hosts/puertos** anteriores. - Confirmar **NAT**/salida compartida si aplica (para registrar la IP correcta).

## 3) Modelo de seguridad: mTLS y cifrados

* **Autenticación mutua TLS (mTLS)**: el servidor verifica vuestro **certificado cliente** y vosotros verificáis el **certificado servidor (AWH)**.
* **Formato y algoritmo del certificado cliente:**
  + Formato **DER**
  + Firma **SHA256withRSA**
* **Suites TLS soportadas (mínimo):**
  + TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
  + TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256

Nota: En entornos corporativos suele exigirse **SAN** (SubjectAltName) con el **CN/host** de la app consumidora; no lo olvidéis en el CSR.

## 4) Ciclo de vida de certificados (paso a paso)

### 4.1 Generar keystore y par de claves

keytool -genkeypair \  
 -keystore <ruta/cliente\_app.jks> \  
 -sigalg SHA256withRSA -keyalg RSA

Guardad la **password** del keystore en vuestro gestor de secretos.

### 4.2 Generar CSR

keytool -certreq \  
 -keystore <ruta/cliente\_app.jks> \  
 -file cliente\_app.csr

**Alternativa con OpenSSL** (forzando campos requeridos + SAN):

openssl req -new \  
 -subj "/C=ES/ST=ES/L=MADRID/O=VUESTRA\_ORG/OU=ES/CN=app.ejemplo.corp/emailAddress=devops@ejemplo.com" \  
 -addext "subjectAltName=DNS:app.ejemplo.corp" \  
 -newkey rsa:2048 -keyout cliente\_app.key -out cliente\_app.req

### 4.3 Firma por CA corporativa

* Subid el **CSR** al **catálogo corporativo** correspondiente (ServiceNow, etc.) y esperad el **CRT** firmado.

### 4.4 Importar el certificado **firmado** en vuestro JKS

keytool -importcert -keystore cliente\_app.jks -file cliente\_app.crt -alias cliente\_app

### 4.5 Exportar el **cert público** para Swift

keytool -exportcert -keystore cliente\_app.jks -file cliente\_app.cer

* Enviad **cliente\_app.cer** al equipo Swift para su importación en AWH.

### 4.6 Importar el **certificado público de AWH** en vuestro **truststore**

keytool -importcert \  
 -keystore cliente\_app.jks \  
 -file awh\_api.cer \  
 -alias awh

### 4.7 Verificación

keytool -list -v -keystore cliente\_app.jks

Debéis ver **al menos dos entradas**: vuestro alias (cliente) y awh (servidor).

## 5) Configuración de pruebas con SoapUI (opcional pero recomendado)

1. **Crear** un proyecto vacío.
2. **Importar** el **YAML/OpenAPI** de AWH (nativo en SoapUI ≥ 5.7; en versiones previas, instalar *Swagger Connector*).
3. En la **Request**, configurar el **endpoint** del entorno, p. ej.:
   * https://180.18.253.164:48400
4. Añadir cabeceras **obligatorias** en la pestaña **Headers**:
   * Version: 'x.y.z' (versión de vuestra app)
   * Application: 'nombre\_app\_en\_AWH'
5. En **Preferences → SSL Settings**, seleccionar vuestro **keystore (JKS)** y su **password**.
6. Probar la llamada (verificar en **SSL Info** que se usa vuestro cert) y revisar **Representations** en caso de error.

## 6) Consumir la API: parámetros, paginación y respuesta

* La API es **REST** y devuelve **JSON**.
* **Parámetros**: múltiples filtros combinables (vacíos → sin efecto). Ejemplo típico: own\_bic, fechas, tipo de mensaje, etc.
* **Paginación / scroll**: el servicio acepta **criterios de paginación** (limit, etc.) y **scroll** para grandes volúmenes.
* **Cabeceras mínimas**: Version, Application.
* **Campo total\_count**: número total de mensajes que cumplen el filtro.

**Ejemplo de llamada (TEST):**

GET https://180.18.253.164:48400/v1/messages?own\_bic=BSCHESM0&limit=2  
Headers:  
 Version: '1.0.0'  
 Application: 'mi\_app\_awh'  
 Content-Type: application/json

**Respuesta (fragmento):**

{  
 "total\_count": 2,  
 "results": [  
 {  
 "own\_bic": "BSCHESM0GTS",  
 "message\_type": "fin.103.2021",  
 "uetr": "254e7132-08b1-45c0-a66f-ab181aaf65e7",  
 "status": "Sent to network",  
 "value\_date": ["2022-05-10"],  
 "amount": [{"value": 123, "currency": "BRL"}],  
 "ordering\_customer\_account": ["ES2100490001552116240241"]  
 // ...  
 }  
 ]  
}

## 7) BICs autorizados

* La app se asocia a un **perfil AWH** con los **BICs** que el cliente puede visualizar.
* **Acción Ingestas:** Confirmad con **negocio/seguridad** el **listado de BICs** a habilitar en el alta de la aplicación.

## 8) Gestión de errores y diagnóstico

* Revisar **estado HTTP** y pestaña **Representations** en SoapUI para diagnósticos.
* Validar:
  + Reglas **FW** y **resolución DNS/IP**.
  + **mTLS**: keystore correcto, alias correcto, cadena CA completa.
  + **Cabeceras** Version y Application presentes y con valor.
  + **Filtros** (parámetros vacíos no aplican; combinaciones muy restrictivas → 0 resultados).

## 9) Checklist de entrega a Producción

* Reglas de **firewall** abiertas para **TEST** y **PROD** (hosts/puertos).
* **JKS** generado, **CSR** firmado, **cert cliente** importado.
* **Cert público del cliente** enviado a Swift y **dado de alta** en AWH.
* **Cert público de AWH** importado en truststore.
* **Monitorización** de errores de red/TLS y métricas de consumo API (latencia, 4xx/5xx, timeouts).
* **Rotación** de certificados calendarizada (antes de caducidad).

## 10) Glosario técnico (rápido pero útil)

* **AWH (Alliance Warehouse):** Repositorio/servicio de mensajería SWIFT consumible vía API.
* **BIC:** Código identificador de entidad en red SWIFT.
* **CSR:** *Certificate Signing Request*; petición que contiene vuestra clave pública y DN/SAN para firmar por CA.
* **DER:** Formato binario habitual para certificados X.509.
* **JKS:** *Java KeyStore*; almacén de claves/certificados para Java.
* **Keystore / Truststore:** El primero guarda **clave privada+cert cliente**; el segundo, **certificados de confianza** (servidores/CA).
* **mTLS:** Autenticación mutua en TLS: cliente y servidor presentan certificado.
* **SAN:** *Subject Alternative Name*; extensión X.509 necesaria para hostnames modernos.
* **Suites TLS GCM:** Cifrados simétricos con autenticación (AES-GCM) y clave negociada con ECDHE.
* **SoapUI:** Herramienta de pruebas para APIs (soporta OpenAPI/YAML a partir de 5.7).
* **Version / Application (headers):** Metadatos obligatorios para identificar versión y nombre de la app consumidora en AWH.
* **Scroll/Paginación:** Mecanismos para leer grandes sets de resultados de forma incremental.

## 11) Anexos

### 11.1 Comandos de referencia

# Generar par de claves  
keytool -genkeypair -keystore cliente\_app.jks -sigalg SHA256withRSA -keyalg RSA  
  
# CSR  
keytool -certreq -keystore cliente\_app.jks -file cliente\_app.csr  
  
# Importar cert firmado  
keytool -importcert -keystore cliente\_app.jks -file cliente\_app.crt -alias cliente\_app  
  
# Exportar público  
keytool -exportcert -keystore cliente\_app.jks -file cliente\_app.cer  
  
# Importar cert público AWH  
keytool -importcert -keystore cliente\_app.jks -file awh\_api.cer -alias awh  
  
# Listar contenidos del JKS  
keytool -list -v -keystore cliente\_app.jks

### 11.2 Plantilla de request (curl)

curl -v \  
 --cert-type P12 --cert <opcional\_si\_usas\_p12> \  
 --key <opcional\_si\_usas\_key> \  
 --cacert awh\_api.cer \  
 "https://180.18.253.164:48400/v1/messages?own\_bic=BSCHESM0&limit=2" \  
 -H "Version: '1.0.0'" \  
 -H "Application: 'mi\_app\_awh'"

Si trabajáis con Java/OkHttp/Apache HTTP Client, basta con cargar el **JKS** en el SSLContext y añadir las **headers**.

## 12) Decisiones y dependencias

* **Dependencia Seguridad/PKI:** firma del CSR y alta del cert en AWH.
* **Dependencia Redes:** apertura de FW y resolución.
* **Decisión de despliegue:** uso de puerto **48400** vs **443** según balanceador y políticas.

## 13) FAQ express

* **¿Puedo reutilizar un JKS existente?** Sí, si cumple **DER + SHA256withRSA** y cadena CA válida; si no, generad uno nuevo.
* **¿Falla el mTLS en TEST pero no en PROD?** Revisad que el **cert de AWH (TEST)** está también en el truststore y que usáis el **endpoint** correcto.
* **¿Sin resultados pero 200 OK?** Puede que los filtros sean demasiado restrictivos; comprobad **total\_count** y probad sin date.

*Hecho con cariño para Ingestas. Si algo falla, culpad al DNS (es broma… salvo cuando no lo es).*