a **realistic, secure SSL bridging setup** for RabbitMQ Let's focus on **HTTPS (SSL Terminate/Re-encrypt)** traffic, and create a **custom HTTPS health monitor** that validates the `/api/overview` endpoint.

Below is the **exact step-by-step** configuration on F5 (BIG-IP), both conceptually and practically

---

? GOAL

We want F5 to:

1. **Terminate SSL** from clients (Client SSL profile)
2. **Re-encrypt SSL** when connecting to RabbitMQ nodes (Server SSL profile)
3. **Perform health checks** on the RabbitMQ HTTPS Management API endpoint → `https://<rabbitmq-node>:15671/api/overview` → Expect a `200 OK` response

---

? STEP 1 — Create a Client SSL Profile

Purpose: Handle HTTPS from clients → F5

GUI Path:

```
Local Traffic → Profiles → SSL → Client → Create
```

Settings:

Field	Value
Name	`clientssl_rabbitmq`
Parent Profile	`clientssl`
Certificate	(your RabbitMQ public certificate or wildcard cert)
Key	(corresponding private key)
Chain	(optional, CA bundle if used)
Ciphers	`TLSv1.3:TLSv1.2:!RC4:!3DES:!aNULL`
Options	disable SSLv3 / TLSv1.0

This profile decrypts incoming HTTPS requests from clients.

? STEP 2 — Create a Server SSL Profile

Purpose: Re-encrypt HTTPS to backend RabbitMQ nodes

GUI Path:

```
Local Traffic → Profiles → SSL → Server → Create
```

Settings:

Field	Value
Name	`serverssl_rabbitmq`
Parent Profile	`serverssl`
Server Authentication	optional (enable if backend certs are valid and trusted)
Trusted Certificate Authorities	your CA cert bundle if verifying backend certs

This profile encrypts outbound traffic between F5 and RabbitMQ.

✅ STEP 3 — Create a Custom HTTPS Health Monitor

Purpose: Actively test the `/api/overview` management endpoint.

GUI Path:
`Local Traffic → Monitors → Create`

Settings:

Field	Value
Name	`https_rabbitmq_monitor`
Type	`HTTPS`
Send String	`GET /api/overview HTTP/1.1\r\nHost: localhost\r\nConnection: close\r\n\r\n`
Receive String	`200`
Interval	5 (seconds)
Timeout	16 (seconds)
Alias Service Port	15671
Secure	☑ (check this — uses SSL/TLS)

This tells F5 to connect securely to the management API over HTTPS, send a `GET`, and check for "200".

✅ STEP 4 — Create a Pool for RabbitMQ Nodes

Purpose: Define backend servers.

GUI Path:
`Local Traffic → Pools → Create`

Settings:

Field	Value
Name	`rabbitmq_pool`
Health Monitor	`https_rabbitmq_monitor`
Load Balancing Method	`round-robin` or `least-connections`
Members	Add your RabbitMQ nodes (port 15671)

Example:

```
10.0.0.11:15671
10.0.0.12:15671
10.0.0.13:15671
```

✅ STEP 5 — Create a Virtual Server (SSL Bridging)

Purpose: Accept HTTPS requests and proxy to backend HTTPS.

GUI Path:

```
Local Traffic → Virtual Servers → Create
```

Settings:

Field	Value
Name	`vs_rabbitmq_https`
Destination Address	Virtual IP (VIP), e.g. `10.0.0.100`
Service Port	`443`
Protocol	TCP
HTTP Profile	`http`
SSL Profile (Client)	`clientssl_rabbitmq`
SSL Profile (Server)	`serverssl_rabbitmq`
Pool	`rabbitmq_pool`

F5 terminates SSL on the client side, decrypts, applies HTTP logic, then re-encrypts when sending to RabbitMQ.

✅ Summary Diagram

```
Client (HTTPS)
     |
     v
 [ F5 Virtual Server :443 ]
```

```
    |  (ClientSSL decrypt)
    |
    v
HTTP Logic / Monitor / Persistence
    |
    v
[ F5 Pool → RabbitMQ nodes :15671 ]
    |  (ServerSSL encrypt)
    v
RabbitMQ HTTPS (TLS)
```

✅ Verification Checklist

Check	Command / Location
Verify health monitor	`Local Traffic → Pools → Members` (should show "green")
Test via curl	`curl -k https://<F5_VIP>/api/overview`
Check SSL handshake	`openssl s_client -connect <F5_VIP>:443`
Logs	`/var/log/ltm`

❓ 1 The meaning of the "Send String" command

Here's the command from the HTTPS health monitor:

```
GET /api/overview HTTP/1.1\r\nHost: localhost\r\nConnection: close\r\n\r\n
```

This is a **raw HTTP request** that F5 sends directly to the backend server (RabbitMQ in this case) during the health check. Let's break it down piece by piece:

Segment	Meaning
`GET /api/overview`	The HTTP method (`GET`) and URI path you want to test. RabbitMQ's management API exposes `/api/overview`, which returns cluster health/status.
`HTTP/1.1`	The HTTP version being used. HTTP/1.1 requires a `Host` header.
`\r\n`	Carriage return and line feed — end of the line (same as pressing Enter). F5 requires them to format the HTTP request properly.
`Host: localhost`	The required **Host header** in HTTP/1.1. You can replace `localhost` with the actual backend hostname (e.g. `rabbit1.internal`). F5 just sends this as a header string; the backend doesn't use DNS resolution here.

Segment	Meaning
`Connection: close`	Tells the server to close the TCP connection after the response. This helps prevent half-open connections during monitoring.
`\r\n\r\n`	Two newlines in a row signal the **end of the HTTP headers** (empty line before message body).

So the full request looks like this when sent:

```
GET /api/overview HTTP/1.1
Host: localhost
Connection: close
```

If the backend replies with a **200 OK**, F5 considers that node **UP**.

🔍 2 Why port 443 for the Virtual Server (and not 15671)

This is a **key architectural point** in SSL bridging design.

🔍 The logic:

- **Port 443** = The **standard HTTPS port** that your *clients* will connect to (e.g. browsers, apps, API clients).
- **Port 15671** = The **backend RabbitMQ management API port** (where RabbitMQ itself listens for HTTPS).

So, F5's job here is **to expose a clean, public-facing HTTPS endpoint** (443), and then forward that securely to the backend service on its *internal* port (15671).

🔍 Flow Illustration:

```
Client → F5 VIP:443 → RabbitMQ Node:15671
```

- **At the client side:**
  - The user or application calls: `https://f5-rabbit.example.com/api/overview`
  - (Port 443 is implicit in HTTPS)
- **At the F5:**
  - The **Virtual Server** listens on port 443.
  - It decrypts incoming SSL using the **Client SSL Profile**.

- It forwards the HTTP request to the pool member on **port 15671** (RabbitMQ's management HTTPS port).
- Before sending, it re-encrypts the traffic using the **Server SSL Profile**.

- **At the RabbitMQ node:**
  - The backend receives traffic on port 15671 (its configured HTTPS listener).

⚙ Example configuration tie-in:

Component	Listen Port	Function
F5 Virtual Server	`443`	Client-facing HTTPS (standard port)
F5 Pool Members	`15671`	Backend RabbitMQ HTTPS
Client SSL Profile	`clientssl_rabbitmq`	Decrypt incoming HTTPS
Server SSL Profile	`serverssl_rabbitmq`	Re-encrypt to backend HTTPS

❓ Why this is the right approach

- Keeps the **client side** consistent (they always use standard `https://...:443`)
- Allows F5 to **load balance** and **health check** internal ports transparently
- Enables you to **offload / re-encrypt** traffic seamlessly

❓ Example quick curl test:

After this setup, if your F5 virtual IP is `10.0.0.100`, you'll access:

```
curl -k https://10.0.0.100/api/overview
```

F5 will decrypt it → forward securely to `rabbit1:15671` → re-encrypt → get the JSON response → return it to you.

Exactly right — you've got it!

In an **SSL bridging** setup with RabbitMQ (or any HTTPS backend), your **F5 virtual server** should indeed listen on **port 443** — the standard HTTPS port — while the **pool members** (the RabbitMQ nodes) listen on **their actual HTTPS port**, which in this case is **15671**.

Let's recap why this is correct and what happens behind the scenes

---

❓ Full Traffic Path

```
Client → F5 Virtual Server (VIP:443) → RabbitMQ Node (15671)
```

1 Client side

- The client (browser, API, curl, etc.) connects to:

  `https://f5-rabbitmq.example.com/api/overview`

  Port **443** is implicit in HTTPS (no need to specify it).

- F5 decrypts this traffic using its **Client SSL profile** (e.g. `clientssl_rabbitmq`).

---

2 F5 Virtual Server configuration

- **Destination (VIP)** → e.g. `10.0.0.100:443`
- **Client SSL Profile** → `clientssl_rabbitmq` (decrypts client traffic)
- **Server SSL Profile** → `serverssl_rabbitmq` (re-encrypts to backend)
- **Pool** → `rabbitmq_pool`
  - Pool members (RabbitMQ nodes) listen on **port 15671**
  - Health monitor checks `/api/overview` on that same port

So, F5 acts as a **secure reverse proxy / TLS bridge**:

- Decrypts on port **443**
- Re-encrypts internally to port **15671**

---

3 Backend (RabbitMQ nodes)

- Each RabbitMQ node runs its management API on:

  `https://<node-ip>:15671`

- F5's **Server SSL profile** re-encrypts traffic before sending it to the node.

⍰ Summary Table

Component	Port	Role
Client → F5 VIP	443	Client sends HTTPS request
F5 Virtual Server	443	Decrypts (ClientSSL) → re-encrypts (ServerSSL)
F5 Pool Members (RabbitMQ nodes)	15671	Receives HTTPS requests

Component	Port	Role
Monitor Port	15671	F5 health check to `/api/overview`

Bonus Tip

You can also use a **different VIP port** (for example, 8443) *if you explicitly want to expose a nonstandard HTTPS port* — but **443** is the convention for production and automation tools.

So yes — your **Virtual Server should use port 443**, and your **pool members should use port 15671**.

Would you like me to show you the equivalent **TMSH commands** for this entire setup (client SSL + server SSL + pool + monitor + virtual server on 443)? It's super handy if you ever need to deploy it via automation or Ansible.