CUMBUCA

Open Finance Brasil Load Testing Report

Capacity and Stability Assessment

Prepared by	Cumbuca
Date	April 2026
Classification	Confidential
Version	1.0

Contents

Executive Summary 1

1 Introduction 3

1.1 Background: Open Finance Brasil 3

1.2 The Cumbuca Open Finance Brasil Proxy 3

1.3 Objectives of This Assessment 3

1.4 Scope 4

2 Test Environment Architecture 5

2.1 System Under Test 5

2.2 Load Generator 5

2.3 Open Finance Brasil Simulator 5

2.4 Infrastructure Topology 6

3 Methodology 7

3.1 Flow Modelling Rationale 7

3.2 Latency Simulation 7

3.3 Scaling Approach 8

3.4 Metrics Collection 8

3.5 Scope and Limitations 8

4 Results 10

4.1 Throughput 10

4.2 Scenario A: Baseline Results 11

4.3 Scenario B: Realistic Results 11

4.4 Per-Step Request Distribution 11

4.5 Response Time Distribution 13

4.6 Error Analysis 14

4.7 Data Transfer 14

5 Capacity Analysis and Discussion 17

5.1 The mTLS Termination Bottleneck 17

5.2 Comparison to Open Finance Brasil Ecosystem Load 18

5.3 Benchmark: Google Pay Brasil 19
5.4 Proxy Instance Headroom 20
6 Scalability Roadmap 21
6.1 Load Balancer Sharding 21
6.2 Applicability Assessment 21
7 Conclusions 23
Bibliography 24
A Test Infrastructure Configuration 25
B Raw k6 Metrics 26
C Load Generator Design 27
D Open Finance Brasil Protocol Reference 29
E Ecosystem TPS Estimation Methodology 30

EXECUTIVE SUMMARY

What Was Tested, and What the Data Shows

This report presents the design, execution, and results of a load testing campaign conducted against the Cumbuca Open Finance Brasil access proxy. The testing was commissioned to demonstrate the system's capacity and stability under load representative of large-scale production deployment in the Open Finance Brasil ecosystem.

What was tested

The system under test is a specialised proxy that mediates and enforces access to Open Finance Brasil services on behalf of client applications. The proxy handles the full FAPI-BR security and consent protocol, including mutual TLS authentication, Pushed Authorization Requests, and multi-step payment and data consent flows. Testing was conducted by placing the proxy between a custom-built load generator and an Open Finance Brasil simulator, and scaling horizontally until the system's capacity ceiling was reached.

Results summary

Two test scenarios were run: a baseline scenario with no upstream latency (Scenario A), and a production-realistic scenario with upstream latency calibrated to Cumbuca's observed production telemetry (Scenario B). All three test runs converged to a sustained throughput of approximately **52,000 req/s**, within a range of 0.5% across runs.

- **Scenario A (Baseline):** 51,967 req/s at an error rate of 1.39%. Errors are concentrated in HTTP 401 and 502 responses attributable to mTLS handshake saturation at the load balancer under pathological short-connection conditions; they do not reflect proxy application faults.
- **Scenario B, Run 1 (Realistic):** 51,690 req/s at an error rate of **0.001%** (8 errors in 775,344 requests).
- **Scenario B, Run 2 (Realistic):** 51,927 req/s at an error rate of **0.007%** (52 errors in 778,905 requests).

Under production-realistic conditions, the system is effectively error-free at this scale.

Significance

Based on Cumbuca's internal analysis of public Dashboard do Cidadão data, the estimated effective business-hour average throughput of the entire Open Finance Brasil ecosystem is approximately **27,000 TPS**. The tested capacity of ~52,000

req/s is **nearly twice this figure**: the infrastructure as deployed could, in principle, absorb the regular load of every institution in the ecosystem simultaneously, while serving a single large client leaves substantial headroom.

To provide a concrete, institution-level benchmark: **Google Pay Brasil**, currently the largest payment initiator in Open Finance Brasil, recorded a peak weekly volume of 26.06 million requests during the week of 13 March 2026 — an average of approximately 43 TPS. Cumbuca's tested capacity is more than **1,200 times** this figure. At the tested throughput rate, Google Pay Brasil's entire peak weekly request volume would be processed in approximately **500 seconds**.

How the ceiling was found

The 52,000 req/s figure represents an **upper bound**: it is the first load level at which failures appeared that could not be resolved by adding further proxy instances. All lower load levels tested during the campaign were handled successfully by the current architecture through horizontal scaling alone. This means that any realistic production demand below this ceiling — including the client's expected load — is served by the current system without architectural change.

The ceiling and the path beyond it

The failures at 52,000 req/s are attributable to the load balancer's mutual TLS termination capacity, not the proxy application layer. Should throughput requirements approach or exceed this ceiling — relevant under peak ecosystem conditions, estimated at 85,000–100,000 TPS — the pre-planned architectural response is **load balancer sharding**: deploying multiple independent load balancer instances behind an L4 traffic distributor. This change scales mTLS termination capacity linearly, requires no modifications to the proxy application, and can be deployed without service interruption.

Conclusion

Cumbuca's Open Finance Brasil proxy is stable, highly available, and capacity-tested well beyond the regular operational demands of the ecosystem. The organisation has demonstrated both the engineering maturity to achieve these results and the technical depth to characterise the system's limits accurately and propose a credible path to extend them further.

CHAPTER 1

Introduction

1.1 Background: Open Finance Brasil

Open Finance Brasil (OFB) is a regulatory programme governed by the Banco Central do Brasil that mandates the controlled, standardised sharing of financial data and the initiation of payment transactions between regulated institutions, upon the explicit consent of the account holder. Since its phased roll-out beginning in 2021, the ecosystem has grown to encompass thousands of participating institutions and hundreds of millions of API calls per week.

The technical backbone of the ecosystem is the **FAPI-BR security profile**, a Brazilian adaptation of the Financial-grade API (FAPI) standard that mandates mutual TLS (mTLS) for all API connections, Pushed Authorization Requests (PAR, RFC 9126), Proof Key for Code Exchange (PKCE, RFC 7636), and short-lived access tokens with non-rotating refresh tokens. This security posture — designed for the regulated financial context — imposes a substantially higher per-request cryptographic cost than conventional web APIs.

Interactions in the ecosystem follow multi-step protocol flows. A **data flow** requires a regulated institution (the transmitter) to issue a data consent, authorise it through the account holder, exchange OAuth 2.0 tokens, and serve resource API calls against the authorised consent. A **payment flow** additionally requires the creation of a PIX payment consent, execution of the payment, and asynchronous status polling of both the consent and the payment state machines until a terminal state is reached.

1.2 The Cumbuca Open Finance Brasil Proxy

Cumbuca provides high-capacity, high-reliability infrastructure for connecting to Open Finance Brasil. The core product is a specialised proxy that:

- enforces FAPI-BR mTLS client authentication on all inbound connections;
- manages the lifecycle of data and payment consents on behalf of client applications;
- implements the full FAPI-BR authorization server interaction (PAR, PKCE, authorization code flow, client credentials flow, token introspection and revocation);
- routes authenticated, authorised API calls to the appropriate transmitting institution; and
- provides audit logging, access control, and rate-limit enforcement.

The proxy is stateful and protocol-aware: it is not a generic reverse proxy or API gateway. Its correct operation requires handling the complete multi-step consent and authorization sequences defined by the OFB specification.

1.3 Objectives of This Assessment

This assessment was designed to answer four specific questions:

1. **Can the system sustain high throughput without degradation?** Specifically, can it handle request volumes that exceed the expected production demand of a single large client?
2. **Does the system remain stable under production-realistic conditions?** Conditions including real-world latency profiles and the full FAPI-BR protocol overhead.
3. **Where is the capacity ceiling, and what causes it?** Characterising the bottleneck is as important as observing its location: an understood ceiling is manageable; an unexplained one is not.
4. **What is the path to greater capacity if required?** If future demand exceeds the tested ceiling, what architectural change addresses it, and how disruptive would that change be?

1.4 Scope

This report covers load testing of the Cumbuca proxy in isolation: the proxy's ability to accept inbound mTLS connections, process FAPI-BR protocol flows, and route requests to an upstream Open Finance Brasil institution. It does not cover the internal logic of the upstream institution, real-money payment settlement, the Raidiam Directory (participant discovery), or long-term soak testing. These are explicitly noted where relevant to interpreting the results.

CHAPTER 2

Test Environment Architecture

2.1 System Under Test

The system under test is the Cumbuca Open Finance Brasil proxy, deployed as a horizontally scaled pool of **120 instances**, each running on an `m8i.large` virtual machine (2 vCPU, 8 GB RAM). All instances are homogeneous and stateless with respect to request routing; session state relevant to consent and token management is handled by the proxy application layer.

Inbound connections terminate at a **managed application load balancer**, which performs full TLS termination including mutual TLS: it presents a server certificate and verifies the client certificate on every new connection, as required by the FAPI-BR security profile. The load balancer distributes requests across the proxy instance pool using a round-robin policy.

2.2 Load Generator

The load generator is a custom-built harness using **k6** as the execution engine, extended with orchestration logic that replicates complete Open Finance Brasil user flows rather than individual HTTP requests. Two `c5.9xlarge` machines (36 vCPU, 72 GB RAM each) ran k6 agents in parallel, with results combined into aggregate reports at the end of each run.

The generator implements both Open Finance flow types in their entirety:

Payment flow (dominant workload). Each virtual user executes the complete PIX payment initiation sequence: creating a payment consent, submitting a Pushed Authorization Request, simulating institution-side user authorisation, exchanging tokens (client credentials and authorization code grants), initiating the PIX payment, and polling both the payment and consent status endpoints until the flow completes. All PKCE code challenges are generated and verified correctly.

Data flow. Each virtual user creates a data consent, navigates the same FAPI-BR authorization sequence, and then retrieves account list and transaction data using the resulting Bearer token against an authorised consent.

The ratio of payment to data flows is **4:1**, chosen to reflect the client's anticipated payment-heavy usage pattern. This ratio is consistent with the payment-intensive character of the OFB ecosystem, where payment initiation flows are disproportionately high-value relative to their volume share (see Section 5.2). The specific ratio is an assumption of the test design and is discussed further in Section 3.5.

2.3 Open Finance Brasil Simulator

To exercise the proxy without requiring connections to live financial institutions, a dedicated **Open Finance Brasil Simulator** was deployed across 16 `m8i.large` instances. The simulator is a purpose-built Phoenix/Elixir service that implements the full OFB API surface, including:

- FAPI-BR v2 / OIDC: PAR, authorization code flow with PKCE (S256 only), client credentials, refresh tokens, token introspection and revocation;
- Consents API v3.3.1 with full consent state machine enforcement;
- Payments API v4.0.0: PIX payment consents and payment initiation (single-use consent model);
- Accounts API v2.4.2, Customers API v2.2.1, Resources API v3.1.0, and all major Phase 2 data APIs;
- OFB-compliant error envelopes on all 4xx/5xx responses;
- CPF validation using the standard Brazilian check-digit algorithm.

The simulator does not expose shortcut routes: the load generator must navigate the complete protocol sequence — including real PKCE challenges, valid consent IDs in authorization requests, and properly scoped Bearer tokens — in order to receive successful responses. Simulator correctness is validated by a suite of 647 unit tests and 282 end-to-end assertions covering five complete flow scenarios.

2.4 Infrastructure Topology

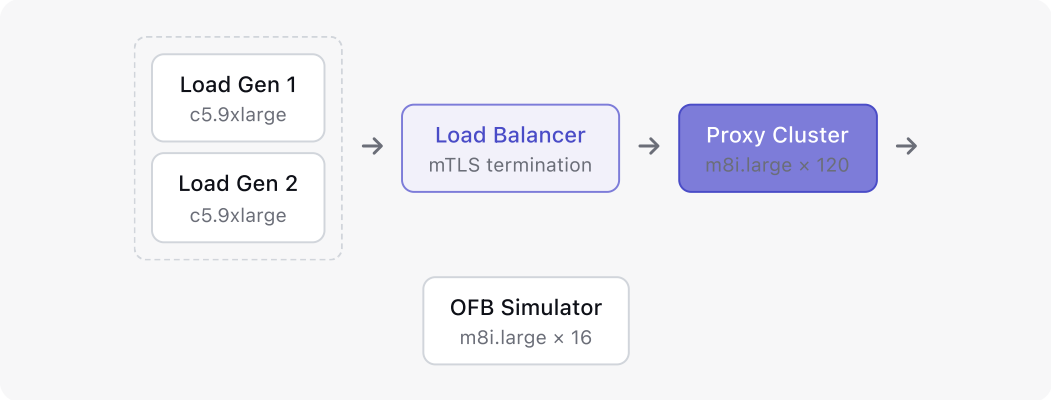

Figure 1. Test environment topology. The proxy cluster (120 instances) is placed between the load balancer and the Open Finance Brasil simulator. Mutual TLS is enforced at the load balancer; all connections from the load generators carry a valid FAPI-BR client certificate.

COMPONENT	INSTANCE TYPE	COUNT	VCPU (TOTAL)	ROLE
Load generators	`c5.9xlarge`	2	72	k6 agent execution
Load balancer	Managed ALB	1	—	mTLS termination and request distribution
Proxy instances	`m8i.large`	120	240	FAPI-BR proxy application
Simulator instances	`m8i.large`	16	32	OFB API simulation

Table 1. Test infrastructure configuration.

CHAPTER 3

Methodology

3.1 Flow Modelling Rationale

Open Finance Brasil interactions are not independent HTTP requests: they are multi-step protocol flows where each step depends on outputs of prior steps (consent IDs, request URIs, authorization codes, access tokens). A meaningful load test must replicate these dependencies at scale, not simply send arbitrary HTTP traffic. The load generator implements two flow types that together cover the two primary use cases of the OFB ecosystem (Figure 2). Each virtual user executes a complete flow from start to finish, replicating the exact request sequence a real Open Finance client would perform.

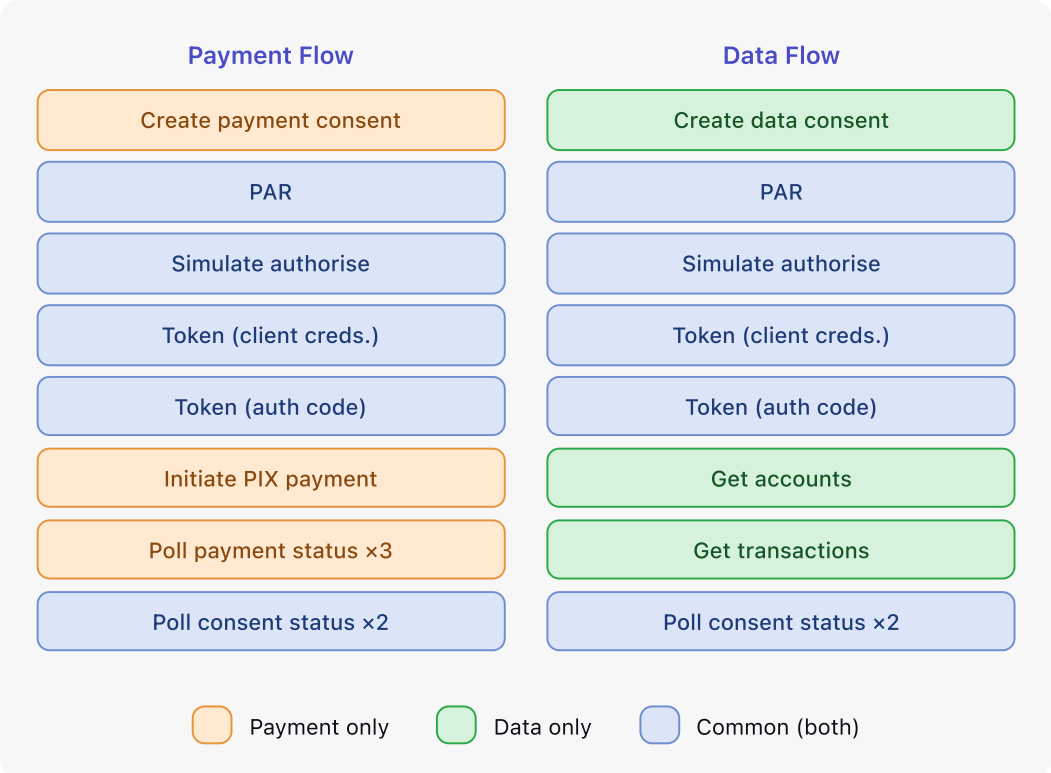

Figure 2. Open Finance Brasil protocol flows implemented by the load generator. Step colours follow the legend: orange for payment-only steps, green for data-only steps, and light blue for steps common to both flow types (the FAPI-BR authorization sequence). The payment flow accounts for 80% of all flows; the data flow accounts for 20%. Polling counts shown are the fixed counts used per flow execution.

3.2 Latency Simulation

Two scenarios were constructed to provide complementary pictures of system behaviour.

3.2.1 Scenario A — Baseline (No Latency)

The simulator responds to all requests as fast as hardware permits, without artificial delay. This is not representative of production conditions: real upstream institutions introduce response latency due to network transit, internal queuing, and database I/O. Scenario A is therefore a stress test of the proxy's pure concurrency and connection-management capacity. It is useful for establishing a conservative lower bound on the system's architectural ceiling and for isolating proxy-side bottlenecks from network effects.

3.2.2 Scenario B — Realistic (Latency Calibrated to Production Telemetry)

The simulator introduces per-response delays drawn from a **piecewise-linear cumulative distribution function** (CDF) anchored to the four parameters listed in Table 2. These values were taken directly from Cumbuca's production telemetry, representing the observed response time distribution of upstream Open Finance Brasil institutions. The piecewise-linear CDF correctly reproduces the major structural features of the real distribution — slow median, moderate 90th percentile, heavy tail — without requiring an assumed parametric form.

PERCENTILE	DELAY
p50	130 ms
p90	350 ms
p99	1,350 ms
max	8,500 ms

Table 2. Latency simulation profile (Scenario B), derived from production telemetry.

Scenario B is the primary scenario for production-relevance conclusions. Its higher per-request latency causes connections to be held open longer, enabling TLS session reuse and substantially reducing the mTLS handshake rate per second relative to Scenario A.

3.3 Scaling Approach

The campaign involved multiple test runs at progressively increasing load levels. At each level, additional proxy instances were added as required; all lower load levels were handled successfully through horizontal scaling alone. The runs reported in this document represent the highest load level reached: approximately 52,000 req/s.

At this load level, failures were observed that could not be resolved by adding further proxy instances — the constraint had shifted from the proxy tier to the load balancer. This load level is therefore characterised as the **upper bound** of the current architecture's capacity: the highest throughput that can be sustained with the current infrastructure configuration, and the first scale at which the load balancer's mTLS termination became the binding constraint. At peak, 120 proxy instances were running.

3.4 Metrics Collection

All metrics were collected via k6's built-in instrumentation. Each load generator machine produced an independent JSON summary at the end of each run, capturing: total and per-step request counts, aggregate throughput (req/s), HTTP status code distribution, error rate, response time distribution (p50, p90, p95, p99, mean, min, max), virtual user count, and data transferred. The two per-machine summaries were then combined into a single aggregate report.

3.5 Scope and Limitations

The following limitations are stated explicitly to support accurate interpretation of the results.

Flow ratio by assumption. The 4:1 payment-to-data flow ratio was chosen to reflect the anticipated usage pattern of the client's deployment, not measured against aggregate ecosystem statistics. A different ratio would produce a different per-step request distribution, but would not materially affect the aggregate throughput conclusions, since both flow types traverse the same proxy infrastructure.

Uniform load profile. k6 virtual users execute flows as fast as possible, without think time. Real user sessions have natural pacing variability. The generator is therefore more aggressive than real users — a conservative assumption for a capacity test — but it does not replicate the burst-then-idle patterns of human traffic.

Single upstream institution. The ecosystem comprises hundreds of regulated institutions. This test used a single simulator representing one institution. The test measures the proxy's own capacity ceiling accurately; it does not characterise per-institution routing behaviour under mixed upstream conditions.

Latency distribution approximation. Real-world upstream latency distributions are right-skewed: a small fraction of requests are significantly slower than the p99 anchor of 1,350 ms, creating a heavier tail than the piecewise-linear CDF produces. Empirical testing confirmed that latency differences on the order of 50–100 ms in the average do not materially affect proxy throughput. The test is marginally optimistic about per-request tail latency; the throughput and stability conclusions are not sensitive to this approximation.

Simulator state management. The simulator maintains all state in memory (no persistent database). Real institutions operate relational databases with write-ahead logging and replication, which contributes additional latency variance under load. The latency injection in Scenario B compensates for the aggregate response time, but does not replicate database-specific contention patterns.

mTLS certificate chain. The load generator uses test certificates. In production, Open Finance Brasil clients present ecosystem-issued certificates verified against the Raidiam Directory trust store. Certificate chain depth and OCSP/CRL checking can marginally affect handshake duration and thus the effective mTLS throughput ceiling. The nature and location of the bottleneck identified in this test would remain

the same, but the exact request-per-second figure at which it occurs may differ slightly.

CHAPTER 4

Results

4.1 Throughput

All three test runs converged to a sustained throughput within a 0.5% band (51,690–51,967 req/s), across both scenarios and across the two independent Scenario B runs. This convergence is strong evidence that the observed value represents a genuine architectural ceiling, not measurement noise.

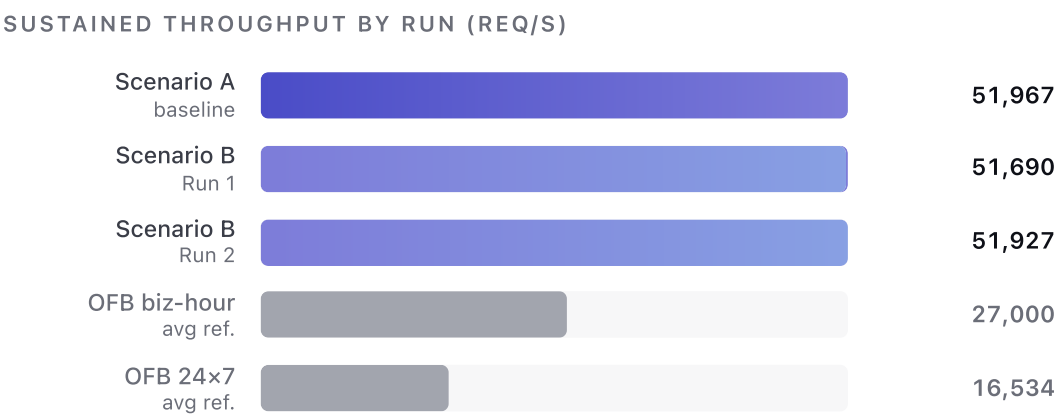

Figure 3. Sustained throughput for all three test runs, with ecosystem reference lines. All three runs land within 0.5% of each other. Reference lines show the estimated ecosystem flat 24×7 average (16,534 TPS) and effective business-hour average (27,000 TPS).

SCENARIO	RUN	DURATION (S)	REQ/S	ERROR RATE	VUS
A	Baseline	39.8	51,967	1.3860%	53,464
B	Run 1	45.0	51,927	0.0067%	57,976
B	Run 2	44.5	51,690	0.0010%	53,753

Table 3. Aggregate results by test run. Duration is the active measurement window; VUs is the peak virtual user count reached during the run.

4.2 Scenario A: Baseline Results

Scenario A sustained 51,967 req/s for 39.8 seconds at an error rate of 1.39%. Analysis of the error breakdown (§ 4.6) shows that these errors are concentrated in HTTP 401 and HTTP 502 responses, which together account for approximately 81.6% of all errors. Both codes are consistent with mTLS session establishment failures at the load balancer under high connection churn — the mechanism is explained in detail in § 5.1.

The median response time was 47 ms, reflecting the absence of simulated upstream latency. This is not representative of production, where even fast

institutions contribute at least tens of milliseconds of response time.

4.3 Scenario B: Realistic Results

Two independent Scenario B runs were conducted to verify reproducibility. Both runs used the same latency profile (Table 2) and the same infrastructure configuration.

Run 1. 51,690 req/s over 44.5 seconds at an error rate of **0.001%**. The errors span multiple HTTP status codes (401, 400, 404, 422, 500), but this is expected: the load generator does not abort a flow on receiving an error response. A single upstream failure early in a flow produces a cascade of subsequent errors as the generator continues with missing or invalid state. The spread across error classes therefore indicates a small number of affected flows, not independent faults of each type.

Run 2. 51,927 req/s over 45.0 seconds at an error rate of **0.007%**. All errors were TCP connection failures (code 0), within the expected noise floor for a test at this concurrency level.

The two runs are within 0.5% of each other in throughput and both have error rates below 0.01%, confirming that Scenario B results are stable and reproducible.

The higher response times in Scenario B relative to Scenario A are expected: they reflect the simulated upstream latency, not proxy overhead. The proxy's own contribution to per-request latency can be approximated from the difference between the simulator's configured p50 (130 ms) and the observed proxy-side p50 (~220 ms), implying a proxy median processing overhead of approximately **90 ms**.

4.4 Per-Step Request Distribution

Table 4 presents the per-step request breakdown for Scenario B, Run 2 (the representative run). The distribution is consistent across all three runs. Two observations are worth highlighting.

First, three of the four shared FAPI-BR authorization steps (PAR, simulate authorise, token via authorization code) each appear at exactly the total flow rate — one invocation per flow, regardless of flow type. Token via client credentials appears at a marginally higher rate; the excess corresponds to retries associated with the transient TCP failures observed in this run (§ 4.3). Every flow, regardless of type, requires the full authorization sequence. This is an accurate representation of FAPI-BR's design: there is no shortcut.

Second, the dominance of polling steps (`poll_payment_status` at 22.7% and `poll_consent_status` at 18.9% of total requests) is a structural characteristic of the OFB protocol, not a test artefact. It mirrors the real-world ecosystem pattern: in the Dashboard do Cidadão data for the reference week, the three payment polling endpoints account for 74% of all Payment Initiation API calls. The load generator's polling pattern is ecologically valid.

STEP	FLOW	% OF TOTAL
Poll payment status	Payment	22.6%
Poll consent status	Both	18.9%
Pushed Authorization Request	Both	9.4%
Simulate authorisation	Both	9.4%
Token (auth code grant)	Both	9.4%
Token (client credentials)	Both	9.4%
Create payment consent	Payment	7.5%
Initiate PIX payment	Payment	7.5%
Create data consent	Data	1.9%
List accounts	Data	1.9%
Get transactions	Data	1.9%
Total		**100.0%**

Table 4. Per-step request distribution, Scenario B Run 2. Flow column: **S** = shared (both flow types), **P** = payment only, **D** = data only.

4.5 Response Time Distribution

Table 5 presents the response time distributions across all three runs. The contrast between scenarios is by design.

SCENARIO	RUN	P50 (MS)	P90 (MS)	P99 (MS)	AVG. (MS)	MAX. (MS)
A	Baseline	47.0	201.5	363.2	131.2	5,276.7
B	Run 1	217.2	588.6	1,789.7	355.7	12,841.7
B	Run 2	221.8	571.1	1,578.2	335.7	9,270.9

Table 5. Response time percentiles by run (all values in milliseconds). Latency in Scenario B reflects the production-calibrated upstream delay profile plus proxy processing overhead.

In Scenario A, the p50 of 47 ms and the p90 of 201 ms reflect the proxy's processing overhead in the absence of any upstream latency. The p99 of 363 ms indicates limited tail inflation under these conditions. In Scenario B, the p99 of ~1,578–1,790 ms is above the simulator's configured p99 of 1,350 ms, reflecting tail amplification from proxy processing overhead and connection queuing under load. The maximum response times (5,277 ms in Scenario A; up to 12,842 ms in Scenario B) are outliers: at 52,000 req/s, even a handful of very slow requests produces a high observed maximum. These values do not indicate sustained latency degradation.

4.6 Error Analysis

The three-order-of-magnitude difference in error rate between Scenario A and Scenario B is the central finding of this analysis.

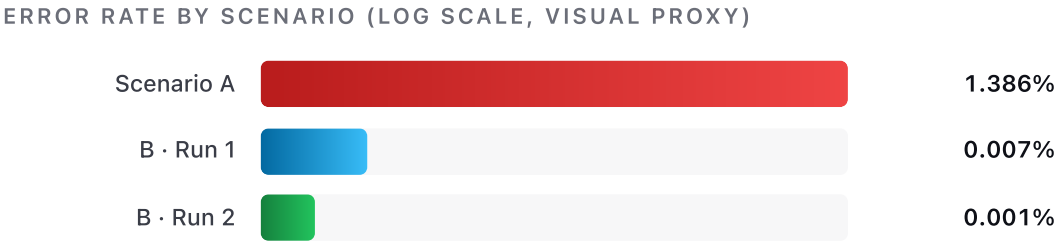

Figure 5. Error rate by scenario. Under production-realistic conditions (Scenario B), the error rate is three to four orders of magnitude lower than in the zero-latency baseline. Bar widths are compressed for visual clarity; absolute values are annotated.

Figure 6 shows the error breakdown for Scenario A. The root-cause errors are HTTP 502 and TCP connection failures: when the load balancer's handshake capacity is saturated, new connection establishment fails, producing 502 (Bad Gateway) responses from the load balancer or TCP-level rejections. HTTP 401 responses arise from the same saturation event when an incomplete TLS session cannot be associated with a valid client certificate. Together these account for the large majority of errors (see § 5.1 for the full mechanism).

The remaining codes — 400, 404, and 422 — are cascading consequences of the same root cause. Because the load generator does not abort a flow on an error response, a flow that encounters a 502 or TCP failure early in its execution continues running with missing or invalid state: subsequent requests attempt to query resources that were never successfully created (producing 404), use tokens that were never issued (producing 401), or submit request bodies referencing IDs from failed prior steps (producing 400 and 422). Every error in Scenario A traces back to the single root cause of mTLS handshake saturation.

CODE	MEANING	SCENARIO A	SCENARIO B · R1	SCENARIO B · R2
200	OK (success)	564,961	573,116	570,525
201	Created (success)	203,739	205,737	204,811
400	Bad Request	817	—	1
401	Unauthorized	7,246	—	4
404	Not Found	655	—	1
422	Unprocessable Entity	480	—	1
500	Internal Server Error	0	—	1
502	Bad Gateway	1,575	—	—
—	TCP connection error	31	52	—
Error rate		**1.386%**	**0.00668%**	**0.00103%**

Table 6. HTTP status code distribution across all runs. Dashes (—) indicate zero occurrences. Success codes (200, 201) dominate in all scenarios; error codes are almost exclusively a Scenario A phenomenon.

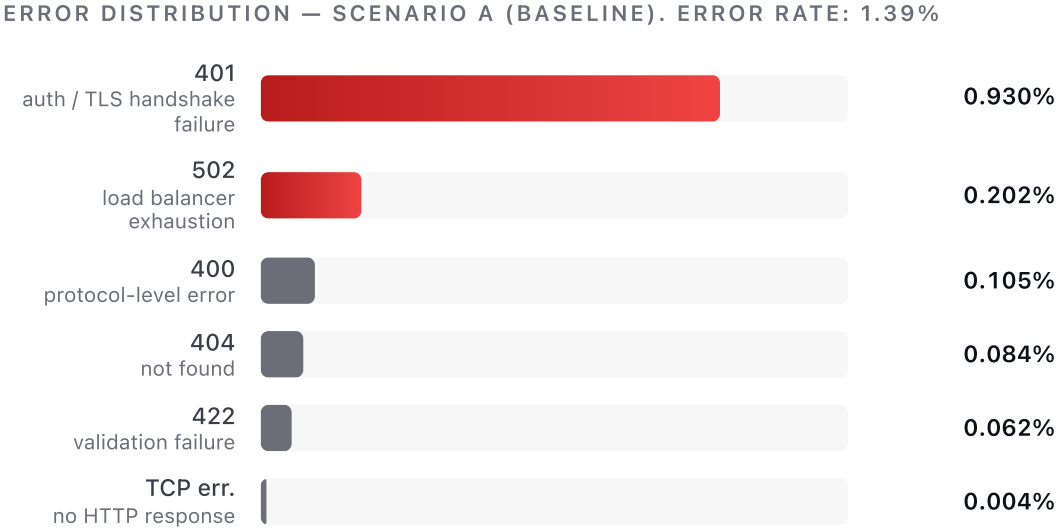

Figure 6. Error distribution by HTTP status code, Scenario A only. Errors highlighted in red (401, 502) are the direct consequence of mTLS handshake saturation at the load balancer. Grey errors (400, 404, 422, TCP) are cascading: the load generator continues executing a flow after an initial failure, producing downstream errors as subsequent steps reference resources or tokens that were never successfully created.

4.7 Data Transfer

Each run transferred approximately 2.2 GB of combined request and response payload over the measurement window of approximately 38–45 seconds. These figures confirm that the proxy was correctly forwarding full response bodies and not silently dropping or truncating content.

SCENARIO	RUN	RECEIVED (GB)	SENT (GB)	TOTAL (GB)
A	Baseline	1.51	0.83	2.34
B	Run 1	1.54	0.84	2.39
B	Run 2	1.52	0.83	2.35

Table 7. Data transfer summary per run. Received refers to data transferred from the simulator to the load generator (via the proxy); Sent refers to the reverse direction.

CHAPTER 5

Capacity Analysis and Discussion

5.1 The mTLS Termination Bottleneck

The entry point to the proxy cluster is an application load balancer configured to perform full mutual TLS termination: for each new inbound connection, it completes a TLS handshake, requests the client certificate, and verifies the certificate chain. This is the correct configuration for FAPI-BR compliance, where every client must authenticate via an ecosystem-issued mTLS certificate.

mTLS handshake processing involves asymmetric cryptographic operations whose cost is bounded by the load balancer's compute resources, independently of the proxy instances behind it. The effective TLS handshake rate the load balancer can sustain is therefore a hard ceiling on the system's request throughput — provided that new connections are being established at a rate approaching that ceiling.

Why Scenario A reaches this ceiling

In Scenario A, the simulator responds in tens of milliseconds. Connections are very short-lived: the median request completes in 47 ms. At 52,000 req/s with a 47 ms median duration, the connection establishment rate is high enough to approach the load balancer's TLS handshake processing capacity. The resulting back-pressure manifests as HTTP 502 (Bad Gateway) responses when the load balancer cannot accept new connections fast enough, and as HTTP 401 responses where handshakes cannot be completed before client-side timeout. This matches the observed error signature precisely.

Why Scenario B does not reach this ceiling

In Scenario B, the simulator responds in ~220 ms at median. Connections are held open substantially longer, and the TLS session established for each connection is reused across the duration of that connection. At the same aggregate throughput of 52,000 req/s, the effective new-connection rate is a fraction of what it was in Scenario A. The load balancer operates well within its handshake capacity, producing the observed near-zero error rate.

> **Why the tested ceiling is conservative**
>
> While ~52,000 req/s is characterised as the upper bound of the current architecture (the first load level where failures appeared that horizontal proxy scaling could not resolve), the realistic production ceiling is not lower — it is at least as high, and likely higher. Under production-realistic connection lifecycles — which include latency-induced TLS session reuse — the same throughput is sustained without error. The ~52,000 req/s figure from Scenario A is a conservative measurement of where the boundary lies.

The proxy instances were not the bottleneck. At no point during the test did the proxy instances exhibit signs of saturation. All horizontal scaling capacity had been absorbed by the load balancer before proxy-level constraints became visible. This is a positive finding: it means the system's scaling path is well-defined and requires changes at one specific layer.

5.2 Comparison to Open Finance Brasil Ecosystem Load

To contextualise the tested capacity, Cumbuca conducted an analytical estimation of the aggregate throughput sustained by the Open Finance Brasil ecosystem, based on publicly available Dashboard do Cidadão volumetric data. The full methodology is reproduced in Appendix E; the headline results are summarised in Table 8 and Figure 7.

REFERENCE	TPS	RATIO TO TESTED CAPACITY
Ecosystem flat 24×7 average	16,534	0.32
Ecosystem effective biz.-hr. avg.	27,000	0.52
Cumbuca tested capacity	**~52,000**	**1.00**
Ecosystem peak (pessimistic)	85,000–100,000	1.63–1.92

Table 8. Open Finance Brasil ecosystem TPS estimates vs. Cumbuca tested capacity. Ecosystem figures use a pessimistic input of 10 billion API calls per week, approximately 47% above the observed weekly peak of 6.79 billion for the reference period.

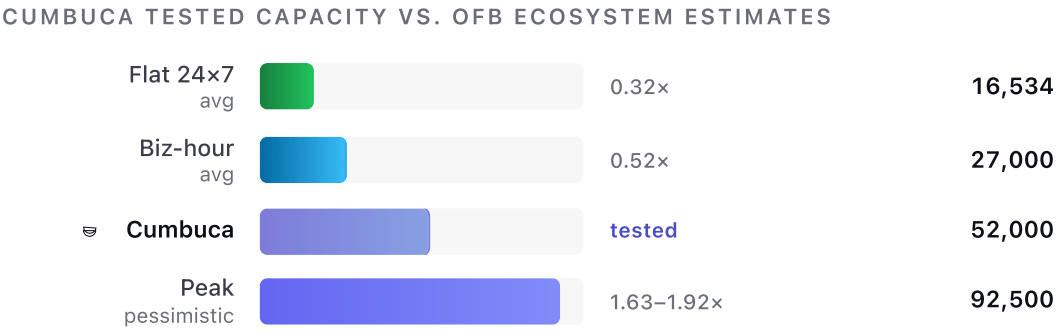

Figure 7. Cumbuca tested capacity against Open Finance Brasil ecosystem TPS estimates. The Cumbuca bar (highlighted) exceeds both the flat average and the business-hour average. The ecosystem peak is a pessimistic estimate and exceeds the current tested ceiling; it is addressed by the load balancer sharding architecture described in Chapter 6.

The primary comparison. The operationally relevant reference for production capacity planning is the effective business-hour average of approximately **27,000 TPS**. This represents the sustained throughput the ecosystem carries during normal working hours, after diurnal adjustment for the mix of machine-driven (continuous) and user-driven (business-hour concentrated) traffic. The tested capacity of ~52,000 req/s is **approximately 1.9 times this figure**.

Ecosystem peak and the sharding path. The estimated ecosystem peak of 85,000–100,000 TPS is built on a pessimistic 1.47× input buffer above observed data and applies retail banking peak-hour multipliers of 2.5–3.0× the business-hour average. This figure exceeds the current tested ceiling. Two considerations are relevant. First, the operational requirement is not to serve the entire ecosystem at its peak simultaneously — it is to serve a single client, whose proportional share of ecosystem peak is far smaller. Second, the pre-planned architectural response (load balancer sharding, described in Chapter 6) scales the mTLS termination capacity linearly without application changes and is ready to deploy.

The polling structure validates the test model. An independent validation point emerges from the ecosystem data. In the reference week, 74% of all Payment Initiation API calls were status polling calls. In the load generator's step distribution (Table 4), the two polling steps (`poll_payment_status` and `poll_consent_status`) account for 41.6% of all requests. The difference is explained by the fact that the load generator includes data flows (which have no payment polling), and by the fixed-poll-count design of the test; the structural dominance of polling in both the real ecosystem and the test model confirms that the flow design is ecologically valid.

5.3 Benchmark: Google Pay Brasil

Aggregate ecosystem figures are useful for broad context but abstract. A more concrete and commercially meaningful benchmark is provided by the largest single payment initiator operating in the Open Finance Brasil ecosystem.

Google Pay Brasil is currently the highest-volume payment initiator on the platform, as reported by the Dashboard do Cidadão. During its peak week (ending 13 March 2026), Google Pay Brasil recorded **26.06 million API requests** across its payment initiation flows. Averaged over the full 604,800 seconds of that week, this corresponds to approximately **43 TPS** — their average request rate at peak weekly volume.

METRIC	SUBJECT	VALUE
Peak weekly requests	Google Pay Brasil	26,060,000
Average TPS (peak week)	Google Pay Brasil	~43 TPS
Sustained tested capacity	Cumbuca	~52,000 req/s
Capacity ratio	Cumbuca / Google Pay	**>1,200×**
Time to serve Google Pay peak volume	Cumbuca (at tested rate)	**~500 seconds**

Table 9. Cumbuca tested capacity vs. Google Pay Brasil peak weekly throughput.

Three observations frame this comparison correctly.

First, the 43 TPS figure is Google Pay Brasil's *weekly average during their busiest week on record* in the reference period — not a typical week. The comparison is therefore against their best-case average, not a depressed baseline.

Second, as noted in § 3.5, Cumbuca's tested throughput of ~52,000 req/s includes all request types across both payment and data flows: authorization tokens, PAR, consent management, payment initiation, and status polling. If only the payment-flow-specific requests from our test are counted — still a like-for-like comparison with Google Pay's payment initiation traffic — the figure is ~19,600 req/s, which is still **more than 450 times** Google Pay Brasil's peak weekly average. Payment-flow-specific requests are those exclusively generated by payment flows and absent from data flows: **Create payment consent** (`POST /payments/v4/consents`), **Initiate PIX payment** (`POST /payments/v4/pix/payments`), and **Poll payment status** (`GET /payments/v4/pix/payments/{id}`). Together these three step types account for approximately 37.7% of total requests in the test (Table 4), yielding 37.7% × 52,000 req/s ≈ 19,600 req/s.

Third, Google Pay Brasil is, by definition, the ceiling of what the largest payment initiator in the ecosystem currently demands. Any production deployment serving a single large client would be expected to carry a fraction of that volume, for which the current architecture provides substantial headroom without any changes.

5.4 Proxy Instance Headroom

At the tested ceiling, 120 proxy instances were running. The load balancer became the bottleneck before any proxy-level constraint was observed. This means the proxy application layer has headroom: further instance additions would yield more throughput if the load balancer constraint were removed. The sharding architecture described in Chapter 6 unlocks this headroom.

CHAPTER 6

Scalability Roadmap

6.1 Load Balancer Sharding

The mTLS termination bottleneck is a capacity limit on a single load balancer instance, not an architectural constraint of the system. It can be addressed directly by **sharding the load balancer tier**: replacing the single load balancer with multiple independent instances, each handling a distinct subset of connections, behind a lightweight L4 traffic distributor (Figure 8).

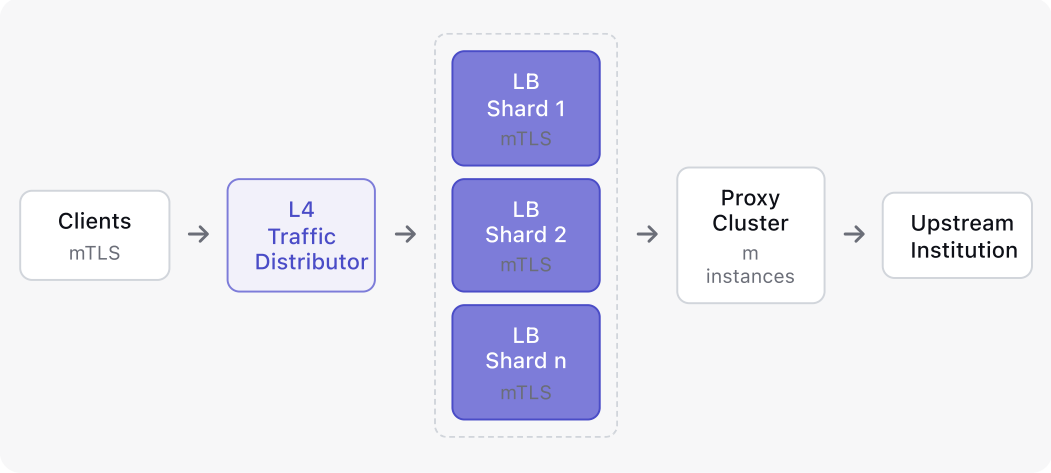

Figure 8. Load balancer sharding architecture. An L4 traffic distributor partitions incoming connections across n independent load balancer shards, each performing mTLS termination. mTLS termination capacity scales linearly with n. The proxy cluster and upstream institution are unchanged.

The properties of this approach are:

- **Linear scaling.** Each additional load balancer shard adds an equal increment of mTLS processing capacity. Two shards double the ceiling; three shards triple it; and so on.
- **No application changes.** The proxy instances, their configuration, and the upstream routing logic are unaffected. The change is confined entirely to the infrastructure layer.
- **Client transparency.** From a client's perspective, the L4 dispatcher is the single entry point. mTLS authentication is performed independently at each shard; clients are not aware of the sharding topology.
- **Non-disruptive deployment.** Shards can be added incrementally. A rolling deployment procedure adds a new shard, validates it under live traffic, then repeats until the target capacity is reached, without interrupting service.
- **Infrastructure agnosticism.** The architecture is implementable on any major cloud provider or on-premises environment. The L4 dispatcher function is

provided by hardware load balancers, cloud networking primitives, or DNS-based routing depending on the deployment context.

6.2 Applicability Assessment

At the tested capacity of ~52,000 req/s, the current single-shard architecture handles the estimated regular load of the entire Open Finance Brasil ecosystem with approximately 90% headroom over the business-hour average (Section 5.2). Activation of the sharding architecture is not required for serving a single large client at foreseeable demand levels.

The sharding path is documented here as a pre-planned, engineering-ready response if demand requirements change. The time-to-deploy for a first shard addition is bounded by infrastructure provisioning and load balancer configuration time — an operational matter, not an engineering one.

CHAPTER 7

Conclusions

The following conclusions are supported by the test data:

1. **The proxy sustains approximately 52,000 requests per second.** All three test runs, across two scenarios, converged to a throughput of 51,690–51,967 req/s (a range of 0.5%), demonstrating that this represents a stable, reproducible performance characteristic of the system.
2. **Under production-realistic conditions, the system is effectively error-free.** Scenario B, which uses a latency profile calibrated to Cumbuca's production telemetry, produced error rates of 0.001% and 0.007% across two independent runs. No systematic fault was observed.
3. **The capacity upper bound is precisely characterised.** Approximately 52,000 req/s is the first load level at which failures appeared that could not be resolved by adding further proxy instances. All lower load levels tested during the campaign were served successfully through horizontal scaling alone. The failures at this level are attributable to the load balancer's mTLS handshake processing capacity; under production-realistic conditions the same throughput is sustained without error. The proxy application layer showed no signs of saturation at any point.
4. **The tested capacity substantially exceeds all relevant ecosystem benchmarks.** The estimated effective business-hour average of the entire Open Finance Brasil ecosystem is approximately 27,000 TPS; the tested capacity is ~1.9× this figure. More concretely: Google Pay Brasil, the largest payment initiator in the ecosystem, averaged approximately 43 TPS during its peak week on record. The tested capacity exceeds this by a factor of more than 1,200, and would process Google Pay Brasil's entire peak weekly request volume in under 10 minutes of operation.
5. **A credible and pre-planned path to higher capacity exists.** Load balancer sharding scales mTLS termination capacity linearly, requires no proxy application changes, and can be deployed incrementally without service interruption. This path is ready to activate if demand requirements increase beyond the current tested ceiling.

BIBLIOGRAPHY

References

[1] Cumbuca. *Open Finance Brasil — TPS Estimation: Technical Estimation of the Open Finance Brasil Ecosystem Throughput*. Internal technical document, 2026. Based on Dashboard do Cidadão snapshot of 2026-04-14. Detailed methodology reproduced in Appendix E.

[2] Open Finance Brasil. *Financial-grade API Security Profile v2.2.0 (PT)*. Open Finance Brasil Confluence, page 1675395089. Restricted to registered participants, 2024.

[3] Dashboard do Cidadão — Open Finance Brasil. Public dashboard, `dashboard.openfinancebrasil.org.br`, 2026. Data snapshot reviewed: 2026-04-14 19:43.

[4] Dashboard do Cidadão — *Payment Initiator Ranking*. Public dashboard, `dashboard.openfinancebrasil.org.br`, 2026. Google Pay Brasil peak weekly volume (week of 2026-03-13): 26.06 million requests. Data reviewed 2026-04-19.

[5] B. Campbell, N. Sakimura, D. Tonge, and F. Skokan. *OAuth 2.0 Pushed Authorization Requests*. RFC 9126, IETF, 2021.

[6] N. Sakimura, J. Bradley, and N. Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. RFC 7636, IETF, 2015.

[7] Open Finance Brasil. *Máquina de Estados v4.0.0 — [SV] Pagamentos*. Open Finance Brasil Confluence, page 347078805, 2024.

[8] Grafana Labs. *k6 — Open Source Load Testing Tool*. `k6.io`, 2024.

[9] Dashboard do Cidadão — *Payment Initiation Endpoint Ranking*. `dashboard.openfinancebrasil.org.br/payment-initiation/ranking/endpoints`, 2026. Reference week ending 2026-03-27, reviewed 2026-04-14.

[10] Dashboard do Cidadão — *Transactional Data API Calls Evolution*. `dashboard.openfinancebrasil.org.br/transactional-data/api-requests/evolution`, 2026. Weekly volumes Jan–Mar 2026 reviewed 2026-04-14.

[11] Open Finance Brasil. *Limites de tráfego*. Open Finance Brasil Confluence, page 17989722, 2024.

APPENDIX A

Test Infrastructure Configuration

COMPONENT	TYPE	COUNT	SPECIFICATION
Load generators	`c5.9xlarge`	2	36 vCPU, 72 GB RAM. Two independent k6 agents; results aggregated post-run.
Load balancer	Managed ALB	1	Application load balancer; full TLS termination including mTLS client certificate verification; round-robin distribution to proxy pool.
Proxy instances	`m8i.large`	120	2 vCPU, 8 GB RAM each (240 vCPU / 960 GB RAM aggregate). Stateless request routing; application layer manages consent and token lifecycle.
Simulator instances	`m8i.large`	16	2 vCPU, 8 GB RAM each (32 vCPU / 128 GB RAM aggregate). Phoenix/Elixir service; in-memory state (ETS); no persistent database.

Table 10. Detailed infrastructure configuration for all test runs.

APPENDIX B

Raw k6 Metrics

The following table reproduces the complete set of aggregate metrics from the combined JSON output files produced by k6. Per-machine (non-aggregated) JSON files are available separately and are the authoritative source for per-machine percentile values.

METRIC	UNIT	SCENARIO A	SCENARIO B · R1	SCENARIO B · R2
THROUGHPUT				
Duration	s	39.78	44.49	45.03
Total requests	—	779,504	775,344	778,905
Requests per second	req/s	51,967	51,690	51,927
Virtual users (peak)	—	53,464	53,753	57,976
FLOWS				
Payment flows	—	58,865	58,521	58,807
Data flows	—	14,699	14,625	14,734
Payment / data ratio	—	4.00	4.00	3.99
ERRORS				
Error count	—	10,804	8	52
Error rate	%	1.386	0.00103	0.00668
RESPONSE TIME				
p50	ms	47.0	221.8	217.2
p90	ms	201.5	571.1	588.6
p99	ms	363.2	1,578.2	1,789.7
Mean	ms	131.2	335.7	355.7
Maximum	ms	5,276.7	9,271.0	12,841.7
DATA TRANSFER				
Received	GB	1.41	1.42	1.44
Sent	GB	0.77	0.77	0.79

Table 11. Complete aggregate metrics — all runs.

APPENDIX C

Load Generator Design

The load generator is a purpose-built distributed load testing tool implemented in two layers: an **Elixir orchestration layer** that provisions cloud infrastructure, distributes the test, and aggregates results; and a **k6 execution layer** that runs the actual Open Finance flows on each machine.

Technology Stack

- **Orchestrator:** Elixir / OTP, using the AWS SDK (`ex_aws`) for EC2 instance provisioning and AWS Systems Manager (SSM) for remote command execution. No SSH is required; all remote execution uses IAM-controlled SSM channels.
- **Load engine:** k6 v1.7.1 (Grafana Labs), running the `open_finance.js` scenario script.
- **Cryptography:** JOSE library for PS256-signed JWTs (client assertions per FAPI-BR `private_key_jwt` client authentication); mTLS via k6's native `tlsAuth` configuration.

k6 Scenario Configuration

The `open_finance.js` script uses k6's `constant-arrival-rate` executor, which maintains a target flow rate regardless of response latency by spawning additional virtual users as needed. The key parameters are derived from the `TARGET_TPS` environment variable:

```
FlowsPerSec = ⌈ TARGET_TPS / 10 ⌉   (10 req/flow as conservative estimate)
PreAllocVUs = FlowsPerSec × 10
MaxVUs = PreAllocVUs × 5   (5× buffer for high-latency conditions)
```

In distributed mode (multiple machines), `TARGET_TPS` is divided equally across instances; each machine runs an independent k6 agent with its own connections and TLS sessions.

Flow Implementation

Both flow types perform the complete FAPI-BR authorization sequence. Polling steps include randomised inter-poll delays (0.5–1.5 s for consent polls; 0.5–2.0 s for payment polls), introducing natural pacing variability. PKCE code challenges are included in every PAR request as required by the specification; the challenge and

verifier values are fixed constants in the test script, which is appropriate for a load test against a simulator but would not be suitable for a production security audit.

Execution and Infrastructure Lifecycle

Each test run follows a five-phase lifecycle managed by the Elixir orchestrator:

1. **Setup.** EC2 instances are provisioned with the required IAM role, security group, and user-data script that installs k6. The orchestrator waits for SSM agent readiness before proceeding.
2. **Upload.** mTLS certificates and the k6 scenario script are uploaded to each instance via SSM (base64-encoded to stay within SSM command size limits).
3. **Run.** k6 is invoked on each instance simultaneously via SSM with environment variables configuring the target URL, TPS, duration, and flow distribution. The k6 Web Dashboard is enabled and exported to HTML for post-run review. Results are written to a JSON file on the instance upon completion.
4. **Collection.** The orchestrator retrieves the JSON results and dashboard HTML from each instance. Per-machine reports are saved locally.
5. **Aggregation.** If multiple machines were used, per-machine results are combined: request counts, error counts, and status code distributions are summed; response time percentiles and averages are combined using a request-count-weighted average; min and max values are taken across all machines.
6. **Teardown.** EC2 instances are terminated.

APPENDIX D

Open Finance Brasil Protocol Reference

The following table summarises the OFB API endpoints exercised by the load generator, cross-referenced with the simulator routes and the specification version implemented.

STEP	METHOD	PATH	SPEC
Token (client creds.)	POST	/auth/token	FAPI-BR v2
Token (auth code)	POST	/auth/token	FAPI-BR v2
PAR	POST	/auth/par	RFC 9126
Simulate authorise	POST	/simulate/authorize	—
Create payment consent	POST	/open-banking/payments/v4/consents	Payments v4.0.0
Initiate PIX	POST	/open-banking/payments/v4/pix/payments	Payments v4.0.0
Poll payment status	GET	/open-banking/payments/v4/pix/payments/{id}	Payments v4.0.0
Poll consent status	GET	/open-banking/consents/v3/consents/{id}	Consents v3.3.1
Create data consent	POST	/open-banking/consents/v3/consents	Consents v3.3.1
List accounts	GET	/open-banking/accounts/v2/accounts	Accounts v2.4.2
Get transactions	GET	/open-banking/accounts/v2/.../transactions	Accounts v2.4.2

Table 12. OFB API endpoints exercised in the load test.

The `/simulate/authorize` endpoint is specific to the simulator and has no counterpart in the live ecosystem; it stands in for the institution-side user

interaction that would, in production, involve a redirect to the institution's authorisation portal.

APPENDIX E

Ecosystem TPS Estimation Methodology

This appendix reproduces the methodology used to estimate the aggregate throughput of the Open Finance Brasil ecosystem, as cited in Section 5.2. The analysis is based on publicly available Dashboard do Cidadão data.

Input Data

Transactional data volumes. Weekly API call counts for the five reference weeks Jan–Mar 2026:

WEEK ENDING	API CALLS
2026-01-02	5.71 B
2026-01-30	5.98 B
2026-02-06	6.61 B
2026-03-06	6.79 B
2026-03-27	5.96 B
Mean	**6.21 B**

Working value. A pessimistic input of $V = 10 \times 10^9$ API calls per week (10 B/week) was adopted, approximately 1.47× the observed weekly peak of 6.79 B. This absorbs short-term ecosystem growth, unreported weeks, and failed calls not always classified in the primary tab.

Fundamental Conversion

A week contains $T_{week} = 7 \times 24 \times 3600 = 604{,}800$ s. The flat 24×7 average TPS is:

$$TPS_{flat} = V / T_{week} = 10^{10} / 604{,}800 \approx \mathbf{16{,}534\ TPS}$$

Temporal Distribution Adjustment

Traffic decomposes into two classes:

- **Machine-driven ($\alpha_M \approx 50\%$):** status polling, periodic data refresh. Continuous, approximately uniform over 24×7.

- **User-driven ($1 - \alpha_M \approx 50\%$):** consent creation, authorisation, first-fetch of data. Concentrated on weekday business hours.

The 50% machine-driven estimate is supported by the payment initiation endpoint breakdown, where 74% of all calls are polling-type, and by the structural composition of the top transactional API families (Accounts, Credit Cards, Loans).

$$TPS_M = 0.5 \times 10^{10} / 604{,}800 \approx \mathbf{8{,}267\ TPS}\ \text{(continuous)}$$

$$TPS_U^{bh} = 0.8 \times 0.5 \times 10^{10} / 216{,}000 \approx \mathbf{18{,}519\ TPS}$$

$$TPS_{bh} = TPS_M + TPS_U^{bh} \approx 8{,}267 + 18{,}519 \approx \mathbf{26{,}786\ TPS}$$

Tightening to 90% of user traffic in 10 h × 5 weekdays yields ~35,000–40,000 TPS. Peak-hour factors of 2.5–3.0× applied to the business-hour average give the pessimistic peak band of ~85,000–100,000 TPS.

Sensitivity Analysis

A_M	$K_{PEAK} = 2.0$	$K_{PEAK} = 2.5$	$K_{PEAK} = 3.0$
0.40	~51,000	~62,000	~73,000
0.50	~45,000	~55,000	~64,000
0.60	~40,000	~47,000	~54,000
0.70	~34,000	~39,000	~45,000

Table 13. Peak TPS as a function of machine-driven share (α_M) and peak multiplier (k_{peak}), with V = 10 B/week fixed.

Relationship to the 85,000–100,000 TPS peak estimate

The table above uses the base temporal concentration assumption: 80% of user-driven traffic concentrated in 12 h × 5 weekday business hours (216,000 s), giving $TPS_U^{bh} \approx 18{,}519$ and $TPS_{bh} \approx 27{,}000$. The peak band of **85,000–100,000 TPS** cited in the main text uses a tighter concentration: 90% of user-driven traffic in 10 h × 5 weekdays (180,000 s):

$$TPS_U^{bh,tight} = 0.9 \times 0.5 \times 10^{10} / 180{,}000 \approx \mathbf{25{,}000}$$

$$TPS_{bh}^{tight} = 8{,}267 + 25{,}000 \approx \mathbf{33{,}000}$$

Applying peak multipliers of $k_{peak} = 2.5$–3.0 to this tighter business-hour average gives 33,000 × 2.5 ≈ 83,000 to 33,000 × 3.0 ≈ 99,000 TPS, rounded to the **85,000–100,000 TPS** band. The sensitivity table therefore represents the *base-case* parameter space; the headline peak estimate applies the more conservative (peak-amplifying) temporal assumption.

Cross-Check Against Regulatory Baseline

Per the OFB traffic limits specification, each data-holding institution must sustain a minimum of **300 TPS**, with automatic +150 TPS escalations on breach. With approximately 30 major Tier-1 institutions, the regulatory floor is ~9,000–10,000 TPS. The derived flat average of ~16,500 TPS is consistent with, and modestly above, this floor — indicating that most Tier-1 institutions have already undergone multiple escalations.