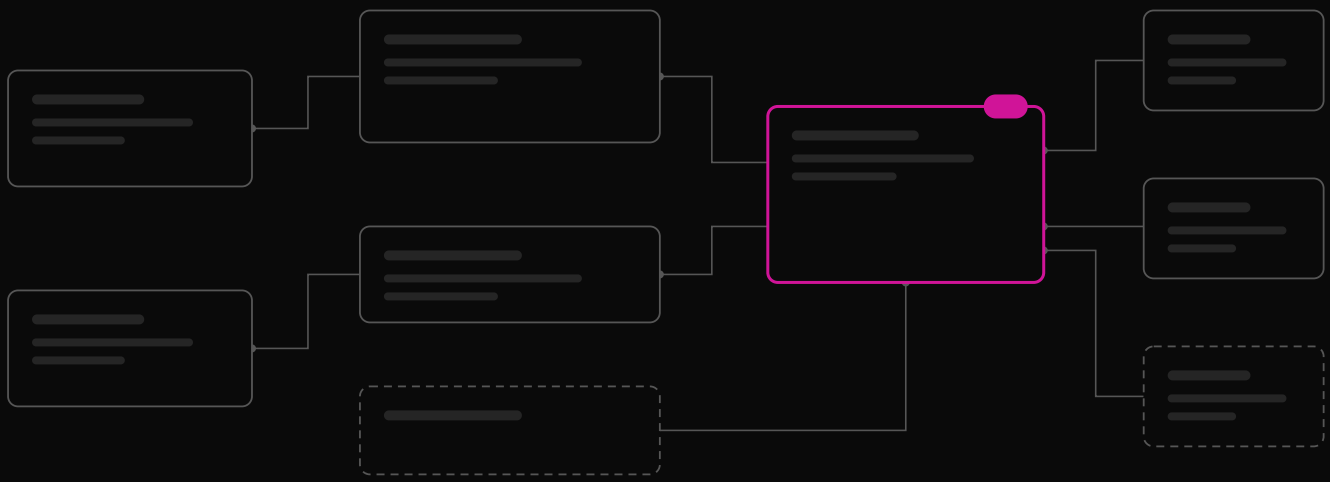
SAMPLE REPORT

Architecture Assessment

Client A (fictional)

B2B order and shipment management platform
September 2026



SAMPLE

Sample. All names, figures and findings are fictional.

This document shows the format and depth of a Future Formative Architecture Assessment. Client A does not exist.

Future Formative Technology LTD · London, UK

Future Formative Teknoloji A.Ş. · İzmir, Türkiye

ABOUT THIS SAMPLE

A condensed sample with fictional data

This document shows the kind of report delivered at the end of an Architecture Assessment. Besides the cover, it contains 13 pages condensed from a findings report that is typically 40 to 60 pages long.

> **Client A is fictional**
>
> The company, its system, every figure and every finding in this sample were invented. No real company, product or person is described. The scenario combines problems that are common in platforms of this age and size, so the reasoning is realistic even though the data is not.

The Architecture Assessment

DURATION	2 to 3 weeks, fixed scope
ACCESS	Read-only. Nothing in your systems is changed.
OUTPUTS	Findings report · risk matrix · C4 diagrams of the current and target state · 12-month roadmap with effort bands and a metric per stage · measurement baseline · 90-minute presentation session
OWNERSHIP	All outputs belong to the client.

What a real report adds

- **Full evidence appendix:** the queries, log extracts, code references and dashboard exports behind every finding.
- **Diagram source files:** editable sources of every C4 diagram, so your team can keep them current.
- **Presentation session:** 90 minutes with your engineering leadership to walk through the findings and agree priorities.
- **Findings at full length:** in this sample each finding is condensed into a single block.

Contents

About this sample 2
Executive summary 3
Scope, questions and method 4
System context 5
Current architecture 6
Measurement baseline 7
Findings F1 to F7 8
Risk matrix 11
Target architecture 12
12-month roadmap 13
What not to do, next steps, glossary 14

Reading notes

Effort is given as a band in engineer-weeks, for example 3 to 5 engineer-weeks.

Severity is High, Medium or Low to medium. Page 11 places each finding by likelihood and impact.

Cost appears only as relative change. No amounts or prices appear anywhere.

EXECUTIVE SUMMARY

The monolith is not the problem. How it is loaded and released is.

The slowdowns, duplicate shipments and rising cloud cost have specific, fixable causes. None of them needs microservices; four measured stages over 12 months address them all.

7.9 s
month-end order p95; 2.8 s over 30 days

22%
change failure rate; manual weekly release

+41%
cloud spend in 12 months; orders +12%

0.3%
of orders get a duplicate shipment event

Answers to the CTO's three questions

QUESTION 1

Why does month-end slow the whole platform?

Month-end reports and CSV exports run on the primary database and lock the orders table, so order submission p95 almost triples, from 2.8 s to 7.9 s. Waiting requests hold threads, so every page slows, not only the reports. A read replica already exists but is unused: moving reporting there is the fastest win (F1).

QUESTION 2

Should we move to microservices?

Not now. The causes are data access, calls without timeouts, message handling and delivery; splitting the code fixes none of them. Services would add network calls to an order path with no timeouts, and more releases to a process with a 46-minute build and a 22% change failure rate. Make the monolith modular first (Orders, Shipments, Billing, Tenancy) and extract a service only when a module needs independent scaling or release.

QUESTION 3

Why has cloud cost grown faster than order volume?

Spend grew 41% in 12 months while order volume grew 12%. The database is sized for the month-end peak all month, and four non-production environments run 24/7. Once reporting leaves the primary (F1), the database can be rightsized and non-production put on schedules (F7), so that spend grows no faster than order volume.

Top risks

F1 Reporting and exports on the primary database
Likely · high impact

F2 Six synchronous calls without timeouts
Likely · high impact

F3 Message handlers that are not idempotent
Certain · medium impact · about 20 hours a month of clean-up

First 90 days

- Timeouts and circuit breakers on order submission F2
- Idempotency keys, inbox and outbox tables F3
- OpenTelemetry tracing on the order path F6
- Reports and exports moved to the read replica F1

Checked at the end of month 3: month-end order p95 below 3 s and zero duplicate shipments.

SCOPE AND METHOD

Scope, questions and method

Three weeks, read-only access and three questions from the CTO. Every finding in this report answers at least one of them.

The questions

#	QUESTION FROM THE CTO	ANSWERED BY
1	Why does month-end slow the whole platform?	F1 F2 F5
2	Should we move to microservices?	F2 F3 F4 F6
3	Why has cloud cost grown faster than order volume?	F7 F1

Sources

SOURCE	WHAT WE USED	PERIOD
Code	Repositories of the monolith, both .NET 8 services and the React front end	Current state
Telemetry	Request, application and database metrics and logs	30 days
Incidents	31 incident records, 4 of them P1	12 months
Cloud bill	Spend by service and environment	12 months
Delivery	Version control, CI runs and the release checklist	30 days
Interviews	9 roles across engineering, operations, product and support. No names recorded.	Weeks 1 and 2

Three weeks

WEEK 1

Understand

Kick-off and read-only access. Architecture walkthrough, code reading, first interviews. Baseline measures defined.

WEEK 2

Measure

Telemetry, incident and cost analysis. Remaining interviews. Each hypothesis checked against data.

WEEK 3

Recommend

Findings, risk matrix, target architecture and roadmap. Report delivered, then the 90-minute presentation session.

Scope and limitations

In scope

The order and shipment platform: ASP.NET monolith, carrier integration and notifications services, React front end, PostgreSQL, RabbitMQ, Redis, AWS infrastructure (EC2 and RDS, one region), and the build and release process.

Not in scope

Security testing and user-interface design.

Limitations

- Read-only: no load tests were run and nothing was changed. Month-end behaviour comes from telemetry, not from a reproduction.
- The 30-day telemetry window contains one month-end.
- Cost is analysed as relative change by service and environment. This sample shows no amounts.
- Effort bands assume engineers who know the code base. They are planning ranges, not quotes.
- Interview statements were treated as leads and checked against data.

CURRENT STATE · C4 LEVEL 1

System context

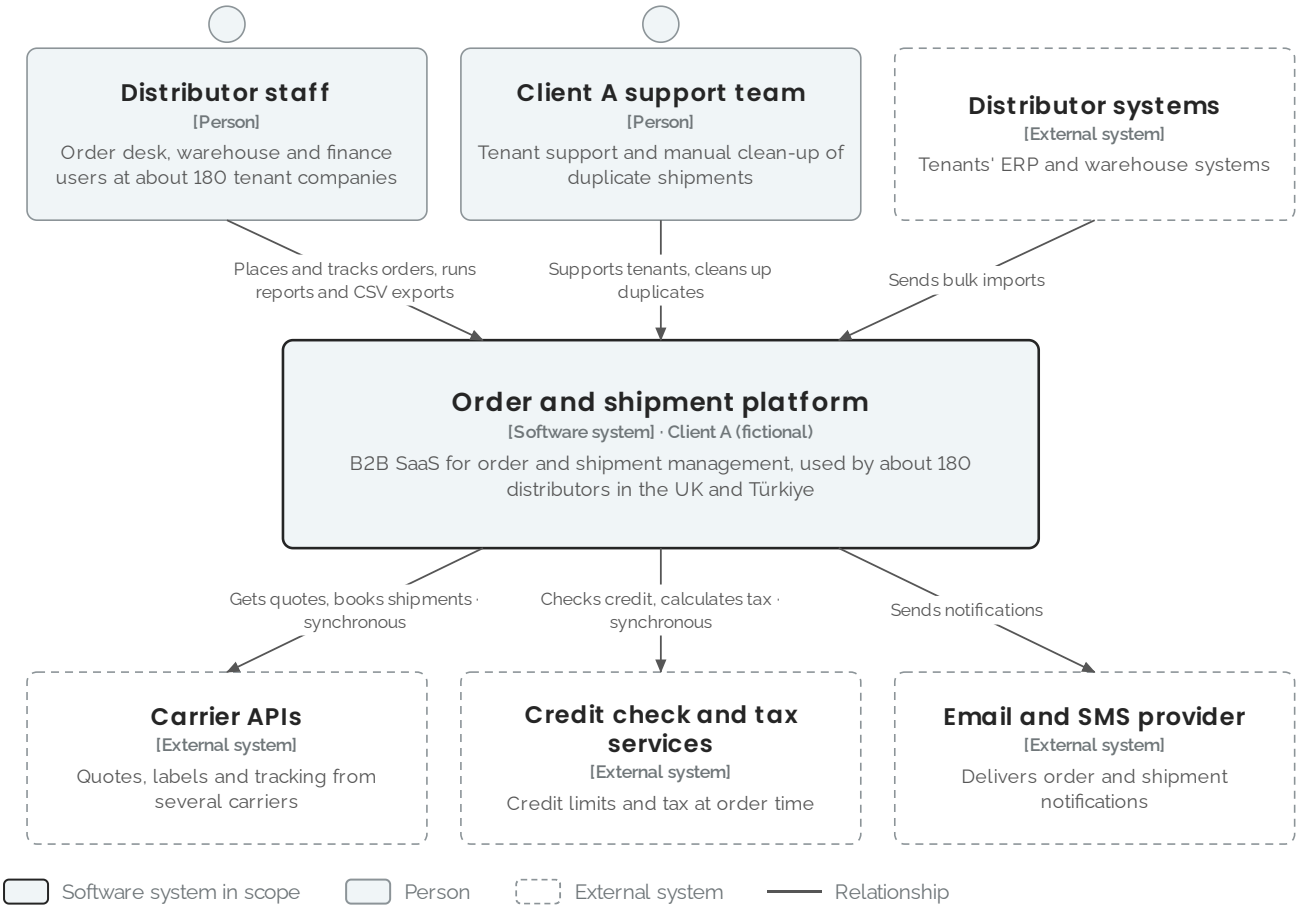
Who uses the platform and which systems it depends on. At this level the platform is a single system; the next page opens it up.

About 180
tenant companies in the UK and Türkiye

38 engineers
in 5 teams

7 years
old ASP.NET monolith, plus two .NET 8 services

1 region
on AWS: EC2 and RDS



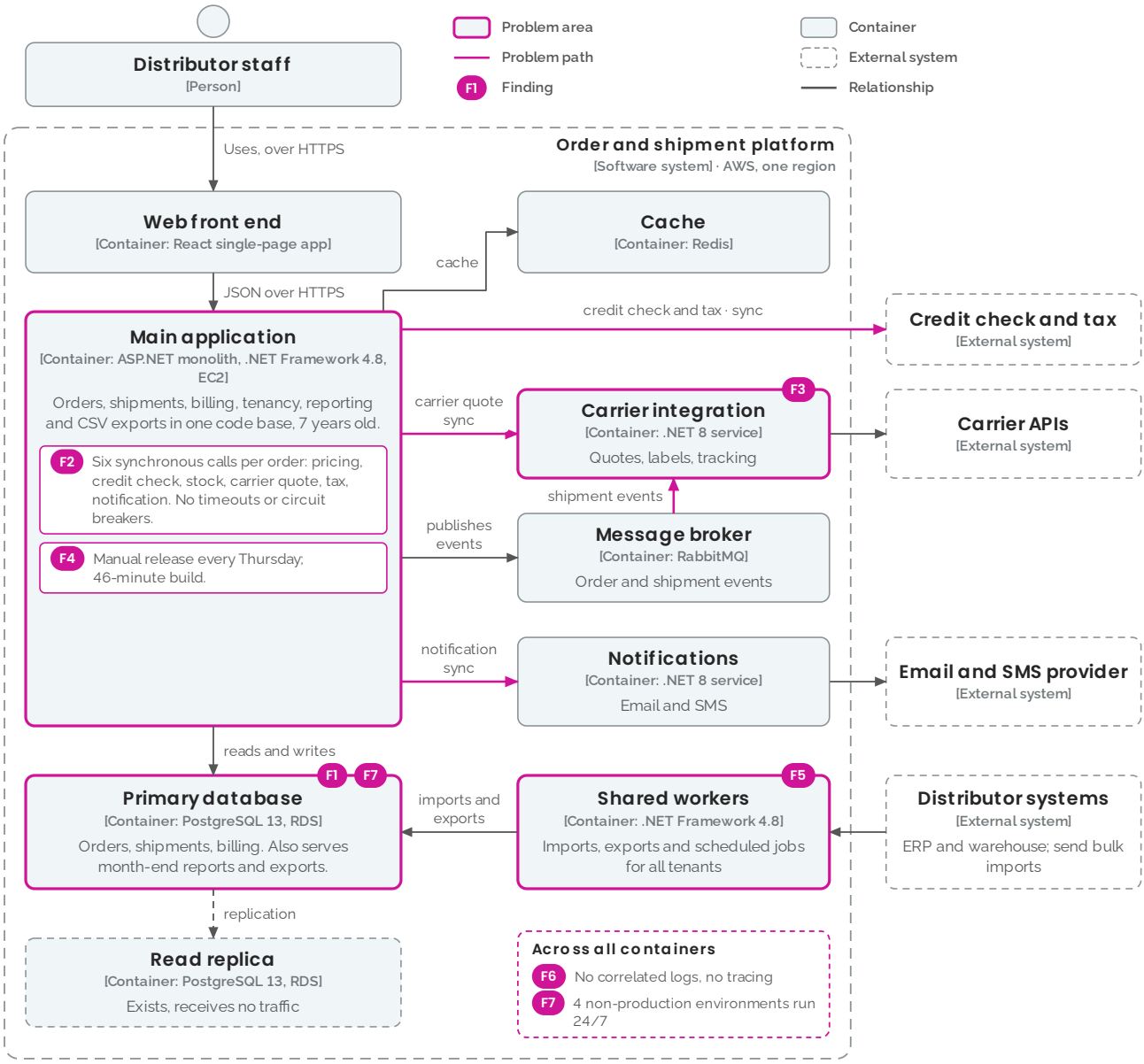
What the context shows

- Every order submission waits on external systems: carrier APIs, the credit check and tax services and, through the notifications service, the email and SMS provider. Without timeouts, their latency becomes Client A's latency F2
- At month-end, distributor staff run reports and CSV exports while orders keep arriving. Both hit the same database F1
- Distributor systems send bulk imports, which are processed on workers shared by all tenants F5
- Client A's support team spends about 20 hours a month cleaning up duplicate shipments F3
- All tenants share one deployment, released once a week on Thursdays with a manual checklist F4

CURRENT STATE · C4 LEVEL 2

Current architecture: containers

The deployable parts of the platform and how they connect. Problem areas are outlined in colour and labelled with their finding.



F1 Month-end reports and CSV exports run on the primary; the replica is idle.

F2 Six synchronous calls per order; no timeouts or circuit breakers.

F3 Handlers are not idempotent; redelivery creates duplicate shipments.

F4 One manual release a week, on Thursdays; 46-minute build.

F5 Large tenants' bulk imports saturate the shared workers.

F6 Logs are not correlated; there is no distributed tracing.

F7 Sized for month-end all month; four non-production environments run 24/7.

MEASUREMENT BASELINE

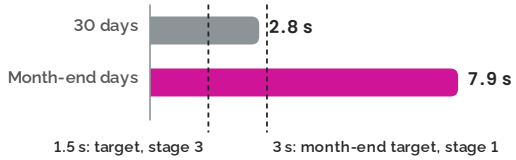
Where the platform stands today

The starting line for the roadmap: every stage ends with a check against these numbers.

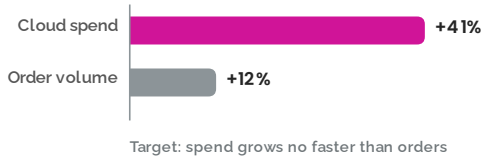
AREA	METRIC	BASELINE	TARGET AND STAGE
LATENCY	Order submission latency, p95	**2.8 s**	Below 1.5 s · months 7 to 9
	Order submission latency, p99	**9.4 s**	Tracked
	Order submission p95 on month-end days	**7.9 s**	Below 3 s · months 1 to 3
	Search latency, p95	**1.2 s**	Tracked
DELIVERY	Deployment frequency	**1 per week**	Tracked
	Lead time for changes, median	**9 days**	Below 3 days · months 4 to 6
	Change failure rate	**22%**	Below 10% · months 4 to 6
	Time to restore, median	**3 h 50 min**	Tracked
	CI build time	**46 min**	Tracked
	Test runs that fail on flaky tests	**7%**	Tracked
RELIABILITY	Incidents in 12 months	**31, of which 4 P1**	Tracked
	Orders with a duplicate shipment event	**0.3%**	Zero · months 1 to 3
	Manual clean-up of duplicates	**About 20 h a month**	None needed · months 1 to 3
COST	Cloud spend against order volume, 12 months	**+41% against +12%**	Spend grows no faster than orders · months 10 to 12

Period: 30 days unless stated. Tracked: reported at every stage check, without a target of its own.

Order submission p95, seconds



Growth over 12 months



How it was measured

Latency

Load balancer and application request logs, all tenants, 30 days. Month-end days: the last two working days of the month and the first working day of the next.

Delivery

Version control and release records; lead time is commit to production. Change failure rate: share of releases needing a rollback or hotfix, or causing an incident. Time to restore: incident start to service restored. Build time and flaky runs from CI logs.

Reliability

12 months of incident records. Duplicates: repeated shipment events for one order in the message logs. Clean-up hours from support records.

Cost

The cloud bill grouped by service and environment, compared with order counts over the same 12 months. Relative change only; no amounts.

FINDINGS

Findings

Seven findings in order of risk. Every block has the same six parts.

Severity: High Medium Low to medium

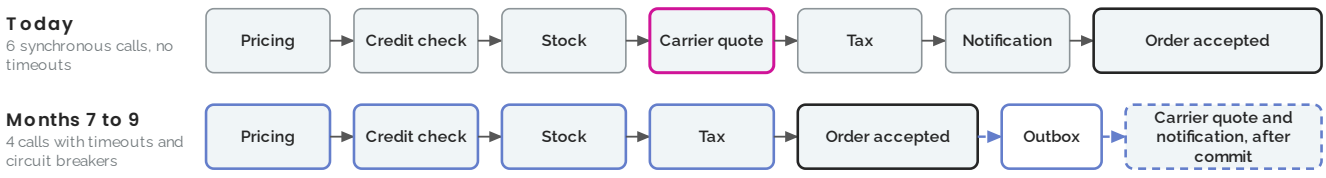
F1 Reporting and exports run on the primary database — High

EVIDENCE	Month-end reports and CSV exports run against the primary PostgreSQL 13 instance and lock the orders table while they read. On month-end days, order submission p95 rises from 2.8 s to 7.9 s. The read replica receives no traffic.
IMPACT	Order submission latency almost triples for all tenants, on the days distributors rely on the platform most. Waiting requests hold threads, so the slowdown spreads to every page.
ROOT CAUSE	Reporting was built into the monolith on the transactional tables. The export job locks the table to get a consistent snapshot, which a repeatable-read transaction would give without blocking writes. Nothing separates analytical load from order traffic.
RECOMMENDATION	**Months 1 to 3:** point month-end reports and CSV exports at the existing read replica and drop the table lock. **Months 7 to 9:** a reporting store fed by change data capture (CDC), so no reporting load reaches the primary.
EFFORT BAND	**3 to 5 engineer-weeks**
DEPENDENCIES	None to start. Enables F7: the database can be rightsized once the month-end peak has gone.

F2 Six synchronous calls on order submission — High

EVIDENCE	Submitting an order calls pricing, credit check, stock, carrier quote, tax and notification in sequence. None of the calls has a timeout or a circuit breaker. p99 is 9.4 s; in the slow requests we sampled, most of the time went on waiting for a carrier quote.
IMPACT	One slow carrier API blocks request threads until the pool is exhausted, and the whole platform slows. The latency of every dependency is added to the customer's wait.
ROOT CAUSE	The order flow grew over seven years by adding calls to a single request. Carrier quote and notification do not need to finish before the order is accepted, but they sit in the same synchronous chain.
RECOMMENDATION	**Months 1 to 3:** timeouts and circuit breakers on all six calls, with a defined fallback for each. **Months 7 to 9:** carrier quote and notification become asynchronous through a transactional outbox, after the order is committed.
EFFORT BAND	**4 to 6 engineer-weeks**
DEPENDENCIES	Tracing (F6) shows which call to fix first. Part 2 reuses the outbox tables introduced by F3.

FIGURE · THE ORDER SUBMISSION PATH TODAY AND AFTER MONTHS 7 TO 9



FINDINGS, CONTINUED

F3 Message handlers are not idempotent

High (certain to recur)

EVIDENCE	0.3% of orders produce a duplicate shipment event. The duplicates we sampled all follow a RabbitMQ redelivery after a consumer restart or a lost acknowledgement. Clean-up takes about 20 hours of manual work a month.
IMPACT	Duplicate shipment events reach carriers and customers and have to be undone by hand. The problem is certain to recur: RabbitMQ delivers at least once, so redelivery is normal behaviour, not a fault.
ROOT CAUSE	Handlers assume that each message arrives exactly once. Nothing records which messages have already been processed, so a redelivered message is processed again.
RECOMMENDATION	An idempotency key on every message, an inbox table in each consumer that records processed keys, and an outbox table so that a state change and its message are written in one transaction.
EFFORT BAND	**2 to 3 engineer-weeks**
DEPENDENCIES	None. Do it first: F2 part 2 builds on the same inbox and outbox tables.

F4 Release process

Medium

EVIDENCE	One manual release a week, on Thursdays, following a checklist. The CI build takes 46 minutes, and 7% of test runs fail on flaky tests that pass on retry. Median lead time for changes is 9 days, change failure rate 22%, median time to restore 3 h 50 min.
IMPACT	Changes wait up to a week and ship in large batches, so each release carries more risk and a failure is harder to trace to one change. Flaky tests teach teams to ignore red builds.
ROOT CAUSE	One test suite for the whole monolith runs in sequence, long-lived branches are merged just before the release, and rollback is a manual step on the checklist.
RECOMMENDATION	Split and parallelise the test suite, quarantine flaky tests and fix them from a tracked list, move to trunk-based development with feature flags, and automate rollback.
EFFORT BAND	**4 to 8 engineer-weeks across teams**
DEPENDENCIES	Needed before any service extraction: more deployable units need a faster and safer release.

F5 No tenant isolation for bulk imports

Medium

EVIDENCE	Bulk imports from large tenants run on the same worker pool as every other tenant's jobs. During large imports the workers are saturated and other jobs queue behind them.
IMPACT	One tenant's import delays other tenants' imports, exports and scheduled jobs: the noisy neighbour effect. The risk grows with every large tenant that joins.
ROOT CAUSE	One shared queue and worker pool, first in first out, with no per-tenant limits.
RECOMMENDATION	Per-tenant queues and rate limits, and separate import workers, so that imports cannot starve other jobs.
EFFORT BAND	**3 to 4 engineer-weeks**
DEPENDENCIES	None. Tracing (F6) makes the effect visible per tenant.

FINDINGS, CONTINUED

F6 Observability gaps

Medium

EVIDENCE	Logs are kept per container and carry no correlation ID, and there is no distributed tracing. In the incident records, most of the time to restore went into finding the cause.
IMPACT	Incidents take hours to diagnose (median time to restore 3 h 50 min), and nobody can see what a customer actually experienced.
ROOT CAUSE	Logging grew component by component. The monolith and the two .NET 8 services were never instrumented as one system, and no service level objectives exist.
RECOMMENDATION	OpenTelemetry tracing on the order path first, across front end, monolith, services, database and broker. Then SLOs for three customer journeys: submitting an order, searching, and tracking a shipment.
EFFORT BAND	**2 to 4 engineer-weeks**
DEPENDENCIES	None. Start early: tracing measures the effect of every other change.

F7 Oversized and idle infrastructure

Low to medium

EVIDENCE	The primary database is sized for the month-end peak all month. Four non-production environments run 24 hours a day, 7 days a week. Cloud spend grew 41% in 12 months, order volume 12%.
IMPACT	Spend grows faster than the business. The effect is a steady trend rather than an outage, hence the lower severity.
ROOT CAUSE	Capacity was raised to survive month-end (see F1) and never lowered. Non-production environments have no schedules.
RECOMMENDATION	After F1, rightsize the database for normal load and run non-production environments on working-hour schedules. Expected effect: spend growth falls back in line with order volume.
EFFORT BAND	**1 to 2 engineer-weeks**
DEPENDENCIES	Requires F1. Rightsizing before reporting leaves the primary would bring the month-end slowdown back.

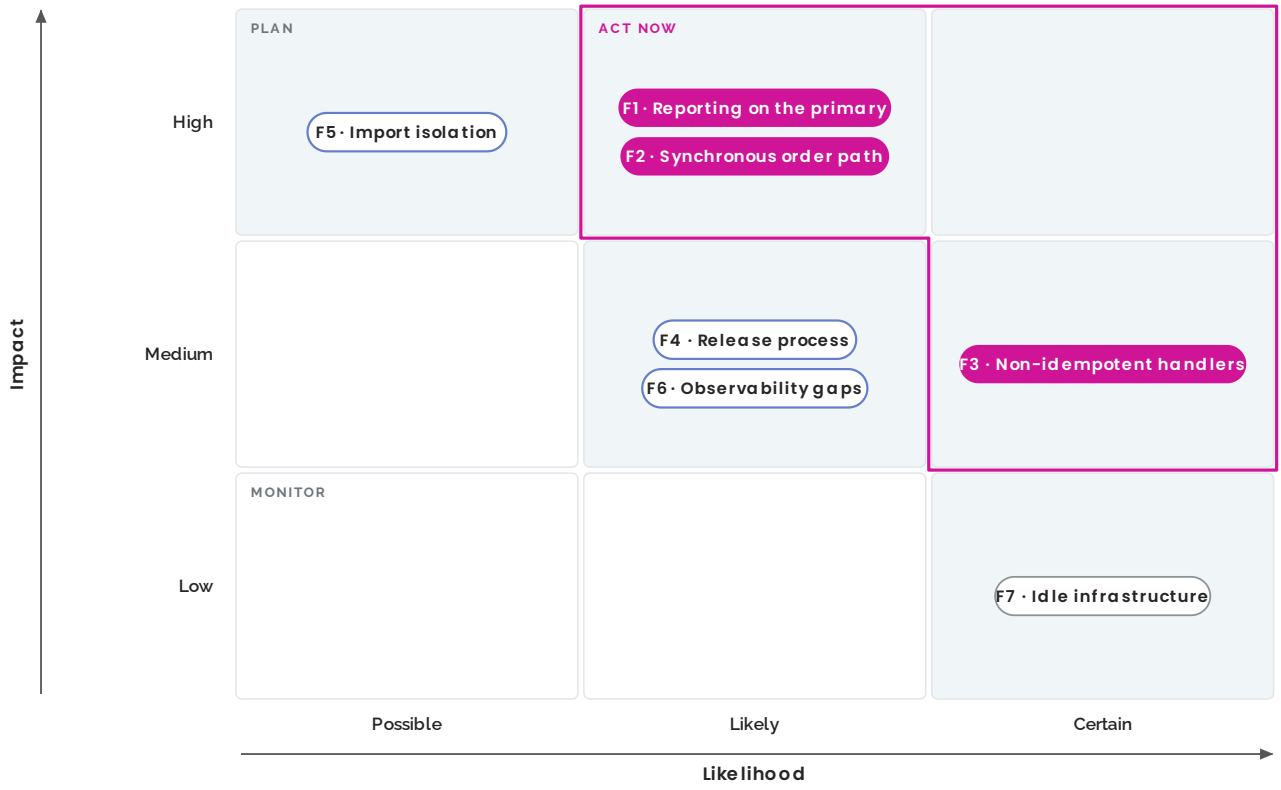
Findings at a glance

FINDING	SEVERITY	EFFORT BAND, ENGINEER-WEEKS	ROADMAP STAGE
F1 Reporting and exports on the primary database	High	3 to 5	1 and 3
F2 Six synchronous calls on order submission	High	4 to 6	1 and 3
F3 Message handlers are not idempotent	High	2 to 3	1
F4 Release process	Medium	4 to 8, across teams	2
F5 No tenant isolation for bulk imports	Medium	3 to 4	2
F6 Observability gaps	Medium	2 to 4	1 and 4
F7 Oversized and idle infrastructure	Low to medium	1 to 2	4

RISK MATRIX

Where the risks sit

Each finding placed by how likely it is to cause harm in the next 12 months and how much harm it does. The outlined cells go first.



How to read it

LIKELIHOOD	**Possible:** needs a particular combination of conditions. **Likely:** expected within 12 months under normal load. **Certain:** happening now and will recur.
IMPACT	**Low:** cost or efficiency. **Medium:** degraded service or manual work. **High:** platform-wide slowdown or failed orders.
SEVERITY	Shown by the chip style: solid for High, blue outline for Medium, grey outline for Low to medium.

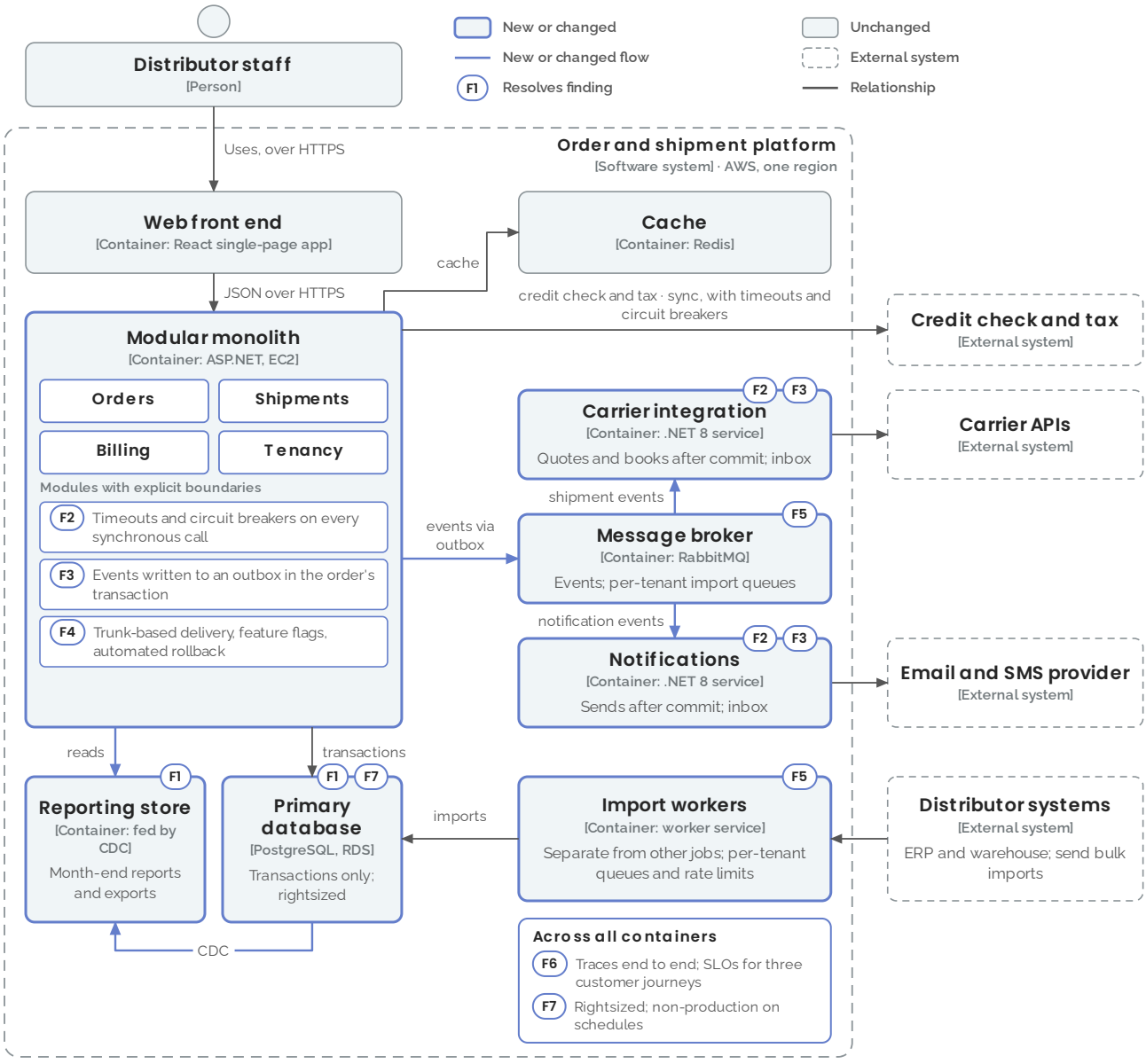
What it tells us

- **F1, F2 and F3** sit in the act-now zone. All three start in stage 1, months 1 to 3.
- **F6** is medium, but its tracing starts in stage 1 because it measures every other change.
- **F4** is likely and medium: it is the precondition for any later service extraction.
- **F5** is only possible today, but its impact is high and grows with every large tenant.
- **F7** is certain but low impact: it costs money, not customers, and it waits for F1.

TARGET STATE · C4 LEVEL 2

Target architecture at month 12

The monolith becomes modular, reporting leaves the primary and the order path becomes asynchronous. No service is extracted until a module needs independent scaling or release.



Principles behind the target

1. **Modules before services.** A module becomes a service only for independent scaling or release.
2. **The primary serves transactions.** Reports read the replica first, then the reporting store fed by CDC.
3. **The order is accepted first.** Carrier quote and notification follow after commit, through the outbox.
4. **Every message may arrive twice.** Handlers use idempotency keys and inbox tables.
5. **No tenant can starve the others.** Imports run on their own workers, with per-tenant limits.
6. **Customer experience is measured.** Traces follow each order; SLOs cover three journeys.
7. **Capacity follows demand.** Sized for normal load; non-production runs on schedules.

12-MONTH ROADMAP

Four stages, each ending with a measurement

Each stage has a few work items and one metric, checked against the baseline on page 7.

M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12

STAGE 1 · MONTHS 1 TO 3

Stabilise

- Timeouts and circuit breakers on the order path F2
- Idempotency keys, inbox and outbox tables F3
- OpenTelemetry tracing on the order path F6
- Reports and exports on the read replica F1

STAGE METRIC

Month-end order p95 below 3 s; zero duplicate shipments.

STAGE 2 · MONTHS 4 TO 6

Deliver safely

- Split and parallelise the test suite; quarantine flaky tests F4
- Trunk-based development with feature flags; automated rollback F4
- Per-tenant queues and rate limits; separate import workers F5

STAGE METRIC

Change failure rate below 10%; lead time below 3 days.

STAGE 3 · MONTHS 7 TO 9

Draw boundaries

- Module boundaries: Orders, Shipments, Billing, Tenancy
- Outbox pipeline: carrier quote and notification become asynchronous F2
- Reporting store fed by change data capture F1

STAGE METRIC

Order p95 below 1.5 s; no reporting load on the primary.

STAGE 4 · MONTHS 10 TO 12

Cost and scale

- Rightsizing and schedules for non-production F7
- SLO review F6
- Capacity model
- Decision on a first service extraction

STAGE METRIC

Cloud spend grows no faster than order volume.

Findings across the stages

FINDING	STAGE 1	STAGE 2	STAGE 3	STAGE 4
F1	Read replica		CDC reporting store	
F2	Timeouts, breakers		Outbox pipeline	
F3	Keys, inbox, outbox			
F4		Tests, trunk, rollback		
F5		Tenant queues, workers		
F6	Tracing			SLO review
F7				Rightsizing, schedules
Architecture			Module boundaries	Capacity, extraction

Dependencies that set the order

- **F6 first:** tracing measures every later change.
- **F1 before F7:** rightsizing is safe only once reporting has left the primary.
- **F3 before F2 part 2:** the outbox pipeline reuses the inbox and outbox tables.
- **F4 before any extraction:** more deployable units need a faster, safer release.

Decision at month 12

Extract a first service only if a module needs independent scaling or release. The capacity model and the SLO review provide the evidence; if neither shows the need, the modular monolith stays.

RECOMMENDATIONS

What we recommend not doing

A full rewrite

It would freeze feature work for a long time and bring back problems the current code has already solved. Every finding in this report can be fixed in place.

A microservices migration now

It adds network calls and releases before timeouts, idempotency and delivery are fixed. Revisit it in months 10 to 12, with data.

Replacing RabbitMQ

The duplicates come from handlers that are not idempotent, not from the broker. Any broker can redeliver a message.

Kubernetes before delivery is fixed

It changes how software runs, not how safely it ships. With a 46-minute build and a 22% change failure rate, it would add risk rather than remove it.

Next steps

1. **Presentation session (90 minutes):** walk through the findings with engineering leadership and agree priorities.
2. **Owners:** name an owner for each stage 1 item and set up the baseline dashboard.
3. **Stage 1:** timeouts and circuit breakers, idempotency, tracing, reports on the read replica.
4. **Re-measure** at the end of month 3 against the baseline on page 7.

The roadmap is written so that Client A's teams can run it on their own. All outputs, including the diagram sources, belong to the client.

Glossary

p95 and p99
The value that 95% (or 99%) of requests stay under. A p95 of 2.8 s means one request in 20 takes longer than 2.8 s.

Transactional outbox
A table written in the same database transaction as the business change. A relay publishes its rows as messages, so no message is lost and none is sent for a change that did not happen.

SLO (service level objective)
A measurable target for a customer journey, such as the share of orders submitted within an agreed time.

Idempotent
Safe to process twice: a repeated message has no additional effect.

Change failure rate
The share of releases that cause a failure in production and need a rollback, hotfix or incident response.

CDC (change data capture)
Streaming row changes from the database log into another store, here the reporting store, without querying the primary.

Circuit breaker
A guard that stops calling a failing dependency for a while, so one slow system cannot hold up the rest.

C4 model
A way of drawing software architecture at four levels of zoom: context, containers, components and code.