

TECH & AI CASE STUDY - LOGISTICS / 3PL - 2025

Containerizing a .NET Monolith That Survives Peak Season for a 3PL

Based on a delivery engagement with LoadHarbor Logistics

6x
PEAK VOLUME HANDLED

-41%
INFRASTRUCTURE COST

2 to 0
PEAK ORDER-INTAKE OUTAGES

**monthly to
weekly**
RELEASE CADENCE

A fillable practice edition: the full story, 8 actions,
6 reflections and a 90-day plan - work it with your team.

Free - no email required
emersontechnology.com

MAKE THIS PRACTICE BOOK YOURS

Work it, do not just read it

This practice book turns the LoadHarbor Logistics case study into a working session for your team. Read the full story first - what happened, what was built and what it changed. Then tick the 8 actions you can commit to, write short answers to the 6 reflections, and finish with a 90-day plan. Every field in this PDF is fillable: open it in Adobe Acrobat Reader, Chrome, Edge or macOS Preview, type directly into the boxes and save.

YOUR NAME	ROLE
COMPANY	TEAM / DEPARTMENT
DATE	EMAIL (OPTIONAL)

MY GOAL FOR THIS PRACTICE SESSION

- 01 The story** - What happened at LoadHarbor Logistics - before, during and after the project.
- 02 The results** - The numbers that changed, plus industry benchmarks from public research.
- 03 The 8-action checklist** - Tick boxes as your organization completes each step. Owner and date fields included.
- 04 The 6 reflections** - Fillable questions that pressure-test your own roadmap.
- 05 My 90-day plan** - Three fillable phase cards that turn this case study into your roadmap.

RUN IT AS A 90-MINUTE TEAM SESSION

25 min READ Read the story together. Ask everyone to note the one number that surprised them most.	40 min PRACTICE Work the 8-action checklist and the 6 reflections. Assign an owner and a date to every action you commit to.	25 min COMMIT Fill in the 90-day plan. One goal per phase, one number that tells you it worked. Book a review date before you leave the room.
--	--	---

THE STORY

What happened, and why it worked

01 The Company

LoadHarbor Logistics is a national third-party logistics provider running 12 warehouses that fulfill orders for e-commerce brands. The company's customers outsource their warehousing, pick-pack fulfillment, transport coordination and billing to LoadHarbor, which means its systems sit directly in the path of every parcel its clients sell — especially during the holiday quarter, when client order volumes multiply five to six times and the fulfillment operation runs at full tilt for weeks. In 3PL economics, the peak season is the business: the revenue concentration is enormous, the SLAs are strict, and the margin for operational error is nearly zero. Adobe Analytics reportedly measured recent U.S. holiday e-commerce seasons above \$220 billion, compressing 3PL fulfillment pressure into a few intense weeks — and LoadHarbor's software had to survive every one of them.

The company's technology estate reflected a decade of organic growth. Its warehouse management, transport and billing systems lived in a 15-year-old .NET Framework monolith on manually managed Windows Server VMs, an estate that worked well enough in the quiet months and fell apart precisely when the money was being made. LoadHarbor's leadership understood the situation with unusual clarity, because their customers did: when fulfillment slips, the brand that sold the parcel is the one that hears about it, and enterprise clients audit their 3PLs' incident history as carefully as their pricing.

02 Where Things Stood

The 2023 season made the risk concrete. Scaling for peak meant ordering hardware months in advance, based on forecasts the business did not fully trust — and even with that lead time, the season still brought two multi-hour outages during order intake at the worst possible moments, missed-SLA penalties and cure notices from two enterprise customers. Order intake is the most sensitive pathway in a 3PL: if orders cannot enter the warehouse management flow, nothing downstream — allocation, waves, picking, shipping, billing — has anything to process. Both outages traced back to the same structural cause: the monolith scaled as a unit, so a surge in one workload dragged every subsystem toward the same ceiling.

The release process carried its own chronic costs. Releases happened roughly monthly through a manual, after-hours process that touched every subsystem at once, so even small fixes waited weeks for a window and every deployment bet the entire warehouse operation on a single coordinated event. The engineering team had evolved defensive habits around the monolith: certain modules had been frozen so long that nobody touched them voluntarily, and institutional knowledge about their internals lived with a handful of veterans. Deloitte and industry analyses have reportedly found that large organizations spend well over half of IT budgets — figures often cited above 70% — simply operating legacy systems, and LoadHarbor's budget told the same story: most of its technology spend bought maintenance and peak-season survival rather than new capability. MHI's annual industry report, produced with Deloitte, reportedly ranks cloud and scalable supply-chain technologies among logistics operators' top adoption and investment priorities — LoadHarbor was about to find out why.

03 The Challenge

The challenge was to modernize a revenue-critical fulfillment platform without ever leaving the warehouse unprotected, and to have the work finished and proven before the next peak season arrived. The calendar was the unforgiving constraint: modernization work could proceed through spring and summer, but November and December were untouchable — no mid-season migrations, no untested deployment machinery, no experiments during the weeks when every order mattered. That deadline compressed the program into a single coherent push with a hard, seasonal examination date that no product launch could reschedule.

The technical challenge decomposed into three risks. First, the unknown-behavior risk: a 15-year-old monolith contains business rules that exist nowhere except in code, and rewriting without pinning those rules first is how modernization programs destroy order data. Second, the scaling risk: even containerized, the platform had to scale on the right signals — warehouse load arrives in predictable waves that CPU metrics detect far too late, and scaling on the wrong trigger produces outages that look like architecture failures. Third, the release risk: warehouse operations cannot tolerate even a few minutes of a bad deploy during shift peaks, so the new deployment machinery needed validation against live volumes before cutover, with instant rollback. The program's promise to the business was blunt: peak season must be boring, and the two cure-notice customers must have no reason to issue a third notice.

04 Discovery and Diagnosis

Emerson Technology began by instrumenting the monolith before touching it. Application performance monitoring went onto every subsystem, and automated contract tests were written around the interfaces between order intake, inventory allocation, wave planning and billing — pinning down what the 15-year-old system actually did, including the undocumented rules that existed only in code. That instrumentation served double duty: it de-risked every subsequent change, and it gave the team its first honest visibility into where time and capacity actually went during order surges. You cannot modernize safely around behavior you cannot see, measure or verify, and the contract-test suite became the program's safety net for the year that followed.

The seam analysis came next. Version-control history and incident records were analyzed to find which modules changed most often and failed most often, because those become your first services — release pain points, not architecture diagrams, should draw the roadmap. The analysis converged quickly: order intake and inventory allocation were simultaneously the hottest, the most fragile and the most business-critical, with wave planning and billing close behind. The diagnosis also quantified the scaling economics: the VM estate was sized against an untrusted peak forecast, meaning the company paid for capacity most of the year to protect against a surge it could neither predict nor shed. That finding reframed the program's financial goal — not merely cheaper infrastructure, but capacity purchased in minutes when waves arrive instead of months ahead of forecasts nobody believes.

05 The Solution

We lifted, refactored and containerized in stages, with the sequencing chosen so that each stage delivered value on its own and reduced the risk of the stage after it.

Pinning behavior before cutting

The monolith's interfaces were pinned with contract tests before any decomposition began, so every extracted service could be validated against the behavior the business actually depends on rather than against documentation nobody trusted. Order intake, inventory allocation, wave planning and billing were then decomposed along those tested seams into separate services, while the remaining core continued to run unchanged. This split — extract the hot paths, leave the stable core — kept the program's risk proportional to its reward: the modules that needed to scale and change fastest gained independence first, and the modules that rarely changed kept their stability until the economics of moving them improved.

Containers first, decomposition second

In parallel, the entire application — including the not-yet-decomposed core — was brought up inside Windows containers on Amazon ECS, even with unchanged code. Containerization delivered autoscaling and repeatable deployments immediately, while gradual decomposition continued safely behind it, which is the essential trick: the business got elastic capacity and weekly releases months before the refactor finished. Terraform provisions

everything, so environments are reproducible and every capacity change is a code review rather than a purchase order. Reporting moved to a read replica feeding Power BI dashboards, giving operations peak-readiness tracking without analytical queries ever touching the transactional path.

Scaling and release machinery

Autoscaling policies tie to order-queue depth rather than CPU alone, adding capacity in minutes when waves arrive instead of in procurement cycles that took months. A Kafka backbone decouples order intake from downstream systems, so marketing-driven spikes queue rather than cascade — backpressure is dramatically cheaper than outage, and a durable queue between intake and fulfillment turns a traffic surge into a scheduling problem instead of an incident. GitHub Actions pipelines with automated regression suites cut releases from monthly to weekly, and blue-green deployments across warehouse regions let operations validate new versions against live volumes before cutover, with instant rollback ready — because warehouse operations cannot tolerate even a few minutes of a bad deploy during shift peaks.

06 The First 90 Days

The first month was entirely about visibility and pinning: APM went live across every subsystem, the contract-test suite was built around the four priority seams, and the container baseline was proven on ECS with the unchanged application. Week five brought the first meaningful cutover of the new era — not a decomposed service, but the whole containerized platform taking live production traffic, with the old VM estate still warm behind it. The win was immediate and visible: deployment went from an after-hours manual ritual to a pipeline run, and the first post-containerization release shipped in week six without an after-hours session.

Weeks seven through ten decomposed order intake, the hottest and most fragile seam, behind the contract tests, and put the Kafka backbone between intake and downstream systems. The timing was deliberate: with the queue in place by early summer, the team had an entire off-peak season to observe backpressure behavior under real marketing spikes from client campaigns. Weeks eleven and twelve delivered inventory allocation extraction, tuned the queue-depth autoscaling policies against observed wave patterns, and stood up the first full-volume rehearsal for the coming peak — a load test against a realistic order mix in early autumn rather than December, because virtually every peak-season failure is predictable in a proper rehearsal. By day 90, weekly releases were routine, the first two services were live, the scaling machinery had been exercised under load, and the peak-readiness dashboard in Power BI had become the leadership team's shared scoreboard for the season ahead.

07 The Results

The following peak season absorbed a 6x volume surge with zero order-intake outages, as ECS scaled warehouse workloads in minutes ahead of each wave instead of months ahead of forecasts. Infrastructure costs fell 41% in the first year, driven by retiring idle Windows VMs and buying capacity on demand instead of ahead of the season. Weekly releases replaced the monthly cycle, cutting release effort per change by roughly 80%, and the two customers that had issued cure notices renewed multi-year contracts after the season closed without incident.

METRIC	BEFORE	AFTER
Peak volume handled	Baseline	6x
Infrastructure cost	Baseline	-41%
Peak order-intake outages	2	0
Release cadence	Monthly	Weekly

The operating rhythm of the business changed with the platform. Capacity conversations shifted from "what did we forecast in July" to "what does the queue say this morning," and the finance team started modeling infrastructure as a variable cost that tracks volume rather than a fixed estate that tracks optimism. Client-facing teams gained a new proof point: peak-readiness dashboards, rehearsed load tests and a clean season are now part of how enterprise accounts are won and renewed. The engineering team, no longer spending its year preparing for a December it feared, began shipping improvements continuously through the year — including into the peak itself, behind blue-green switches, which would have been unthinkable under the old release ritual.

08 What Your Business Should Do

LoadHarbor's modernization is the practical template for any legacy system that faces seasonal cliffs. Instrument the monolith before touching it: add application performance monitoring and automated contract tests to the existing system before refactoring anything, because you cannot modernize safely around behavior you cannot see, measure or verify. Find seams by change frequency — analyze which modules change most often and fail most often, because those become your first services; let release pain points draw the roadmap, not architecture diagrams. Containerize before decomposing: get the entire application running in containers first, even with unchanged code, since containerization delivers autoscaling and repeatable deploys immediately while gradual decomposition happens safely behind it. Scale on business signals, tying autoscaling to order-queue depth or shipment rates rather than CPU alone, because logistics load arrives in predictable waves that infrastructure metrics detect far too late.

Build the season-defense mechanisms that turn peaks into ordinary work. Buffer spikes with a message queue — put a durable queue between order intake and downstream systems so marketing-driven surges wait in line instead of cascading; backpressure is dramatically cheaper than outage. Rehearse peak season like a launch: run full-volume load tests against realistic order mixes in early autumn rather than December, since virtually every peak-season failure is predictable in a proper rehearsal. Ship releases behind blue-green switches, validating new versions against live volumes before cutover with instant rollback ready, because warehouse operations cannot tolerate even a few minutes of a bad deploy during shift peaks. And sunset hardware on a schedule: convert each data-center milestone into freed capacity and retired assets, reporting savings per quarter, since visible wins keep modernization funded through the slow middle phases of any migration. As you plan, measure your own exposure with the questions this program raised: what last peak season cost you in penalties, overtime and goodwill; which system would fail first if volume doubled next month and when you would know; how many months ahead you must buy hardware to cover a forecast you do not trust; and which subsystem has been frozen so long nobody dares touch it. Work through these eight steps with your team using the action checklist in this practice book, and revisit the reflection questions each quarter — peak seasons reward the operators who rehearse, and punish the ones who hope.

REAL-WORLD PROOF & BENCHMARKS

Deloitte and industry analyses have reportedly found that large organizations spend well over half of IT budgets — figures often cited above 70% — simply operating legacy systems. - Deloitte

MHI's annual industry report, produced with Deloitte, reportedly ranks cloud and scalable supply-chain technologies among logistics operators' top adoption and investment priorities. - MHI Annual Report, 2023

Adobe Analytics reportedly measured recent U.S. holiday e-commerce seasons above \$220 billion, compressing 3PL fulfillment pressure into a few intense weeks. - Adobe Analytics, 2023

THE PRACTICE PAGES

The 8-action checklist

Work through the list with your team. Tick each box as your organization completes the step, and fill in an owner and a target date so every action has a name and a deadline.

☐**1. Instrument the monolith before touching it**

Add application performance monitoring and automated contract tests to the existing system before refactoring anything. You cannot modernize safely around behavior you cannot see, measure or verify.

OWNER

TARGET DATE

STATUS

--	--	--

☐**2. Find seams by change frequency**

Analyze which modules change most often and fail most often, because those become your first services. Let release pain points draw the roadmap, not architecture diagrams.

OWNER

TARGET DATE

STATUS

--	--	--

☐**3. Containerize before decomposing**

Get the entire application running in containers first, even with unchanged code. Containerization delivers autoscaling and repeatable deploys immediately while gradual decomposition happens safely behind it.

OWNER

TARGET DATE

STATUS

--	--	--

☐**4. Scale on business signals**

Tie autoscaling to business signals like order-queue depth or shipment rates rather than CPU alone. Logistics load arrives in predictable waves that infrastructure metrics detect far too late.

OWNER

TARGET DATE

STATUS

--	--	--

☐**5. Buffer spikes with a message queue**

Put a durable message queue between order intake and downstream systems so marketing-driven surges wait in line instead of cascading. Backpressure is dramatically cheaper than outage.

OWNER

TARGET DATE

STATUS

--	--	--



6. Rehearse peak season like a launch

Run full-volume load tests against realistic order mixes in early autumn rather than December. Virtually every peak-season failure we have seen was predictable in a proper rehearsal.

OWNER

TARGET DATE

STATUS

--	--	--



7. Ship releases behind blue-green switches

Validate new versions against live volumes before cutting over, with instant rollback ready. Warehouse operations cannot tolerate even a few minutes of a bad deploy during shift peaks.

OWNER

TARGET DATE

STATUS

--	--	--



8. Sunset hardware on a schedule

Convert each data-center milestone into freed capacity and retired assets, reporting savings per quarter. Visible wins keep modernization funded through the slow middle phases of any migration.

OWNER

TARGET DATE

STATUS

--	--	--

AFTER THE CHECKLIST: OUR TOP 3 PRIORITIES

From the eight actions above, write down the three you will start first - and the single number you expect each one to move. Three priorities with owners beat eight without any.

PRIORITY 1 NUMBER IT SHOULD MOVE	PRIORITY 2 NUMBER IT SHOULD MOVE	PRIORITY 3 NUMBER IT SHOULD MOVE
--	--	--

THE PRACTICE PAGES

Reflections for your team

Use these questions in your next planning session and type short answers into the boxes. The gaps you find are usually the fastest wins.

Q1. What did last peak season cost you in penalties, overtime and customer goodwill?

Q2. If order volume doubled next month, which system would fail first — and when would you know?

Q3. How many months ahead must you buy hardware to cover a demand forecast you do not trust?

Q4. Which subsystem changes weekly, and which has been frozen so long nobody dares touch it?

Q5. Could your team ship a warehouse fix on a Tuesday, or only in a release window?

Q6. What share of your IT budget buys maintenance instead of the capacity you actually need?

THE PRACTICE PAGES

My 90-day plan

Finish the session by committing to a plan. Three phases, one goal each, one number that tells you it worked. Bring this page to your next leadership review - or to a free consultation with Emerson Technology, and we will pressure-test it with you.

Days 1-30 Focus: audit and alignment

PRIMARY GOAL**KEY ACTIONS****HOW WE MEASURE SUCCESS****OWNER**

Days 31-60 Focus: build and prove

PRIMARY GOAL**KEY ACTIONS****HOW WE MEASURE SUCCESS****OWNER**

Days 61-90 Focus: scale and measure

PRIMARY GOAL

KEY ACTIONS

HOW WE MEASURE SUCCESS

OWNER

Tip: save a copy per team so each group keeps its own answers. Revisit the reflections every quarter - the questions stay the same even as the numbers move.

BOOK THE REVIEW BEFORE YOU CLOSE THE BOOK

Plans without a review date drift. Before you close this practice book, agree on when you will come back to these pages, who will run that session, and when you will check the numbers again. If you want an outside pair of eyes, bring this page to a free consultation with Emerson Technology and we will pressure-test the plan with you.

REVIEW DATE	REVIEWED BY	NEXT CHECK-IN
------------------------------------	------------------------------------	--------------------------------------

Ready for a transformation like LoadHarbor Logistics's?

Emerson Technology designs and builds the data, AI, cloud and automation systems behind results like these. On a free consultation call we will map this exact playbook to your business - your data, your constraints, your numbers.

Book a free consultation

support@emersontechnology.com

or start online: emersontechnology.com/contact