PLATFORM ENGINEERING

Real-time demand for a gas pipeline design: the architecture behind – Part 3

By Lucas Espindola, At Work Rio Solutions Ltda, Rio de Janeiro, Brazil

Part 2 of this series described what the new GasPipelineDesign Suite functionalities do for a pipeline engineer. This companion note describes what it takes to build them: a stateless API, a queue-backed solver fleet on Azure, a scenario cache that removes 97 per cent of the arithmetic, and an availability model borrowed — almost line for line — from the compressor stations the application simulates.

The two functionalities introduced in Part 2 — the compression-system availability study and the capacity ramp-up — are, to the engineer who has to build them, the same problem wearing different clothes: a compute-bound, embarrassingly parallel workload whose answer has to arrive while the user is still looking at the screen.

THE WORKLOAD, NOT THE DOMAIN

A Monte Carlo availability study draws 100,000 iterations. Each iteration is a state vector — every compressor unit in the system either running or failed — and each *distinct* state vector needs a steady-state thermohydraulic solve to say what capacity the pipeline can still deliver. On the reference case study, one solve takes a shade over a second. Multiplied out, that is twenty-seven hours of arithmetic sitting behind a button that has to respond in under two minutes.

Three levers close that gap, and only one of them is hardware: do not solve inside the request thread; do not solve the same thing twice; and solve whatever is left in parallel. The architecture described here is little more than those three sentences, implemented carefully.

THE ARCHITECTURE IN ONE PARAGRAPH

A TypeScript single-page application talks to a stateless REST API on Azure App Service. The API validates the model, persists it as an immutable, hash-addressed document in blob storage, puts a small job envelope on a Service Bus queue, and returns a job identifier — never a result. A scale set of solver workers competes for those messages, runs the numerical kernel and writes partial results back. The client watches the study fill in over a WebSocket channel. Azure SQL holds project metadata, Redis holds the scenario cache, and Application Insights holds the evidence.

STATELESS BY CONSTRUCTION

The numerical kernel is a pure library: gas composition, equation of state, pressure drop, compressor and driver maps in; capacity, power and fuel out. It performs no I/O, holds no session and knows nothing about HTTP, queues or tenants. That discipline is what makes every other property in this note possible — any worker can take any message, any instance can be destroyed mid-flight, and the identical kernel runs in the unit-test suite, on a developer's laptop and across sixty-four cloud instances.

THE QUEUE IS THE CONTRACT

Service Bus delivers at least once, which means a worker will occasionally see the same scenario twice and a study will occasionally lose a worker to a platform update. Both are ordinary. Work is keyed by job identifier and scenario hash, so a duplicate delivery is a cache write of a value already present, and an evicted worker costs one scenario rather than one study. Messages that fail repeatedly land in the dead-letter queue alongside the exact input document that produced them — in a numerical application, that is the difference between a bug report and a mystery.

REDUNDANCY IS A DESIGN PARAMETER — ON BOTH SIDES OF THE FENCE

Part 2 argued that stand-by compressor units are an economic decision: buy availability until the marginal unit costs more than the penalty it avoids [5,6]. The platform that hosts the argument is sized by identical reasoning. Every Azure service on the request path publishes a service-level agreement, and in series those agreements multiply. Table 1 shows what the request path is worth under three topologies, and where the cheap availability actually is.

	Composite SLA (%)	Downtime (h/year)
Single region; cache, queue and object store on the synchronous path	99.64	31.5
Single region; asynchronous tiers moved off the request path	99.94	5.3
Paired regions, active/active behind Traffic Manager	99.99	0.9

TABLE 1: Composite availability of the request path under three deployment topologies, computed by multiplying the published Azure service-level agreements for the services each topology places in series [1]. This is the software counterpart of Table 1 in Part 2, which set out the reliability and availability of the different compressor-station arrangements surveyed by EPRI [4]: taking the cache, queue and object store off the synchronous path buys more availability than any single tier upgrade, and a second active region buys most of the rest.

A study runs as follows:

1. The API validates the model, writes it to blob storage under the hash of its contents, and enqueues a job envelope of a few hundred bytes. The HTTP call returns in milliseconds.
2. Autoscale reacts to queue depth rather than CPU, and the worker scale set grows from zero towards its ceiling within one scaling interval.
3. Each worker draws a state vector, hashes it, and asks the cache. On a miss it runs the steady-state solve and writes the capacity under failure back under that hash.
4. Partial results — iterations completed, and the distribution of capacity under failure so far — are pushed to the client's WebSocket channel, so the graph fills in while the study is still running.
5. When the iteration budget is exhausted, an aggregation step compiles the capacity/frequency groups, writes the report, and marks the job complete. The workers scale back to zero.

The second lever is worth more than the third. Twenty-one compressor units across seven stations admit two million possible states, but availability figures near 0.99 concentrate the draw heavily: on the reference case, 100,000 iterations produced 2,847 distinct state vectors. Hashing the vector and consulting the cache before solving turns a 100,000-solve study into a 2,847-solve study — a 97.2 per cent hit rate, and the single largest factor in the timings charted below.

Every study carries a seed. The same model and the same seed produce the same distribution, in the same order, whether the work is spread over one worker or sixty-four, in São Paulo or in Texas; partial sums are combined in a defined order rather than in the order they happen to arrive. Reproducibility is not a nicety here. The output of these studies ends up in a redundancy decision or a tariff filing, and a number that cannot be reproduced is an opinion.

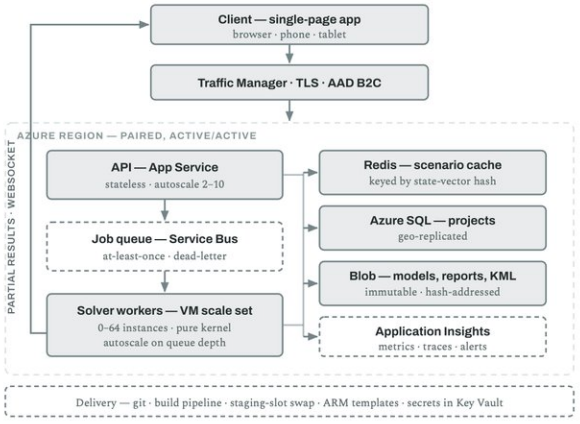
What the platform watches in production is not CPU: solves per second, cache hit ratio, convergence failures per thousand solves, queue age, and time from admission to first partial result. A hit ratio that falls says the model has changed shape; convergence failures that rise say the physics has been handed something it cannot resolve, which is almost always an input-validation bug upstream.

Cost falls out of the same design. The reference study bills roughly fourteen US cents of compute; the same fleet left running around the clock would bill about US$4,500 per month. Elasticity is not a line item — it is the reason a 100,000-iteration availability study is a routine click rather than an overnight job on somebody's workstation. »

A hundred thousand Monte Carlo iterations contain only a few thousand distinct compressor states. The rest is arithmetic we have already done.

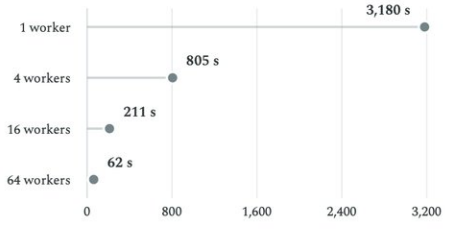
REFERENCE ARCHITECTURE — REQUEST AND FAN-OUT PATHS

Stateless API, queue-backed solver fleet, shared state and telemetry, results streamed back as they accumulate.



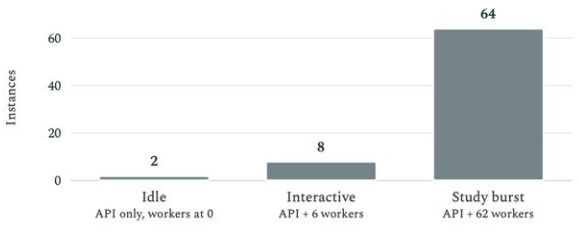
STUDY WALL-CLOCK BY FAN-OUT WIDTH

Seconds to complete a 100,000-iteration availability study. 51× speed-up at 64 workers — 80 per cent parallel efficiency.



ELASTIC COMPUTE FOOTPRINT

Instances held by the platform through a working day. The capacity ramp-up, rewritten as a billing meter.



ABOVE: The request and fan-out paths of the GasPipelineDesign Suite (left); wall-clock time for a 100,000-iteration availability study against the width of the worker fleet (right, top); and the compute the platform actually holds across a working day (right, bottom) — the software counterpart of the capacity ramp-up study charted in Part 2, with capacity added on demand and released the moment the study ends.

Performance and cost figures are modelled from the architecture described here, the case-study workload published in Part 2, and 2018 Azure list prices; they illustrate the engineering argument rather than reproduce a single production trace.

PLATFORM ENGINEERING

CASE STUDY: THE GASBOL RUN AS A WORKLOAD

The case study in Part 2 — the GASBOL main section redesigned under today's assumptions, 1,118 miles of 36-inch pipe, seven compressor stations of two operating units and one stand-by each, 1,100.76 MMcf/d — doubles as a convenient workload profile. What follows is the same run described from the platform's side of the glass.

(a) Workload:

Compressor units in the reliability model: 21

Monte Carlo iterations: 100,000

Distinct state vectors observed: 2,847

Scenario cache hit rate: 97.2 per cent

Median steady-state solve: 1.09 s

Solves per design sweep (five candidate diameters): 1,240

Input document: 84 KB, hash-addressed

Iteration results compiled into capacity/frequency groups: 9

(b) Platform:

Region: Brazil South, paired with South Central US

API: App Service S2, autoscale 2–10 instances

Workers: VM scale set, D2 v3, autoscale 0–64 on queue depth

Messaging: Service Bus Standard, dead-letter enabled

State:

- Azure SQL S2, geo-replicated
- Blob storage RA-GRS, immutable inputs
- Redis C1, scenario cache

Real-time channel: WebSockets, one topic per job

Identity: Azure AD B2C, per-tenant isolation; secrets in Key Vault

Telemetry: Application Insights, correlated by job id

Delivery: git, build pipeline, staging-slot swap, ARM templates

Results for the run:

(a) Platform:

API latency, p50 / p95 / p99: 82 / 240 / 610 ms

Admission to first partial result: 4.1 s

Study wall clock:

- 64 workers: 62 s
- 16 workers: 211 s
- single worker: 3,180 s

Speed-up at 64 workers: 51× (80 per cent efficiency)

Implied serial fraction (Amdahl [3]): 0.4 per cent

Solves avoided by the scenario cache: 97,153

Composite SLA of the request path:

- single region: 99.94
- paired regions, active/active: 99.99

Reproducibility: identical results across regions for a given seed

Regression suite: 214 golden cases, 0.1 per cent tolerance against the reference simulator

(b) Cost (in US$):

Worker compute per study: 0.12

Messaging, storage and egress per study: 0.02

Marginal cost per study: 0.14

Cost per 1,000 studies: 140.00

Platform baseline per month: 473.00

Equivalent always-on fleet per month: 4,485.00

CONCLUSION

The binding constraint in engineering computation is rarely hardware. It is what you choose not to compute. Twenty-seven hours of arithmetic became sixty-two seconds here not because sixty-four machines were thrown at it, but because 97.2 per cent of that arithmetic had already been done: a hundred thousand Monte Carlo draws across twenty-one compressor units contain only a few thousand distinct states, and recognising that was worth more than the entire fleet. The cloud primitives underneath are deliberately ordinary — competing consumers, queue-based load levelling, cache-aside and retry [2] — because the engineering that mattered lay elsewhere: in refusing to hold state anywhere it could be avoided, and in treating availability and cost as design outputs with arithmetic behind them rather than as operational afterthoughts. The physics in the kernel is Sidney's, and it has been right for thirty years. What this side of the fence adds is reach. A study that was once an overnight job on a specialist's workstation is now a click that costs fourteen cents and returns the same answer to anyone, anywhere, from the same seed. ■

REFERENCES

1. Microsoft, 2018. Service Level Agreements — App Service, SQL Database, Service Bus, Storage, Traffic Manager. Azure documentation.
2. Microsoft Patterns & Practices, 2017. Cloud Design Patterns: competing consumers, queue-based load levelling, cache-aside, retry, circuit breaker.
3. G.M.Amdahl, 1967. Validity of the single processor approach to achieving large scale computing capabilities. AFIPS Conference Proceedings, Vol. 30.
4. Electric Power Research Institute, 1999. Report No. RP 4CH2983.
5. S.P.Santos, 2009. Monte Carlo simulation – a key for a feasible gas pipeline design. Pipeline Simulation Interest Group, Galveston, TX, USA.
6. S.P.Santos, 2008. Availability and risk analysis effects on gas pipeline tariff making. International Pipeline Conference, Calgary, Canada.

About the developer

The GasPipelineDesign Suite is the work of the author, Lucas Espindola, a software engineer and expert in cloud-native architectures, data visualisation, real-time information processing and numerical optimisation, and in the patterns and practices that keep software maintainable. The numerical models and the engineering criteria behind them are the work of Sidney P. Santos, who retired from Petrobras in 2012 after more than 25 years as a Senior Consultant and gas-pipeline design engineer.