Skaraz Data

The Climate Pipeline Reality Check

5 lessons + 12-point checklist from parametric insurance pipelines

By Stéphane Karasiewicz

Climate Pipeline Partner · skarazdata.com

May 2026

Why this PDF exists

If you're running a parametric insurance product or a climate risk platform, your pipelines are probably in one of three states:

1. **Working but fragile** — you ship perils, but each new geography or model update is a 6-week project.
2. **Bolted together** — code duplicated between perils, datasets that quietly change calibration, dashboards that look fine until they don't.
3. **Genuinely modular** — but you only get here after 18+ months of iteration, and most teams don't.

I've spent the last 2 years building production climate pipelines across drought, heat stress, precipitation, and water balance — covering 10+ countries in LatAm, SEA, and Africa. This PDF is the distillation of what I'd tell my past self.

No sales pitch. No call required. If you want to discuss your specific context after reading, the Discovery option is at the end.

The 5 Lessons

Lesson 1 — Ship the MVP peril before the modular architecture

The trap. You join a parametric insurtech early-stage and immediately see the need for a "real" modular architecture to scale to 10+ perils. You want to do things right. You propose a big refactor.

The reality. You'll spend 3 months designing an elegant system that serves nobody. Meanwhile, the 3 deals your sales team is trying to close depend on the next peril shipping in 6 weeks.

The lesson. Ship the MVP peril in "code duplication assumed" mode in 6 weeks. Then when you have 3-4 perils in production, refactor with the knowledge of their actual differences. Premature modular code is more expensive than temporary duplication.

> TAKEAWAY
>
> ***Modular architecture is what you extract from 3 working pipelines, not what you design before the first.***

Lesson 2 — Calibration drift is the silent killer

The trap. You use CHIRPS for precipitation, ERA5-Land for temperature, IMERG for near-real-time. Everything works at first deployment. You ship. You move to the next peril.

The reality. 4 months later, your drought model starts under-estimating triggers in Colombia. You haven't changed anything. But ECMWF released ERA5-Land v2 with a different calibration in tropical zones. Your coefficients are now calibrated on data that no longer exists.

The lesson. Pin your dataset versions explicitly (ERA5-Land v1.x.y), version your calibration pipelines, and run a monthly regression test on known historical dates. If your outputs drift, you'll know before your client does.

> TAKEAWAY
>
> ***In climate pipelines, the dataset is part of the code. Version it.***

Lesson 3 — Production is 3am on a Sunday, not a Jupyter notebook

The trap. Your data scientist has a notebook that works. The model is validated on backtest. You move it to production via a Docker container + a cron that runs a Python script every night. Done.

The reality. One Wednesday at 4am, the GFS server serving satellite data is 6 hours late. Your cron runs, finds nothing, writes an empty file to S3. At 8am, the client receives their daily report: "Drought risk: 0% everywhere." Disaster.

The lesson. Production climate pipelines need retry logic, data freshness checks, alerts on missing inputs, graceful degradation, and a human notified if something drifts. 80% of production code isn't the model — it's everything around the model.

> TAKEAWAY
>
> ***Production is the model + everything that prevents it from waking you up at 3am.***

Lesson 4 — Climate scientists and DevOps don't speak the same language

The trap. You have a brilliant climatologist who builds an SPEI model with remarkable precision. You have a senior DevOps who deploys microservices on Kubernetes. They're on the same team but don't understand each other.

The reality. The climatologist says "just run this notebook daily." The DevOps hears "let's orchestrate it with Airflow." 3 months later, the Airflow runs but the model no longer reflects what the climatologist validated — someone "optimized" the data prep along the way.

The lesson. Someone on the team must be able to challenge model choices through comparative studies **AND** read a Dockerfile **AND** debug PostgreSQL in production. If nobody bridges these worlds, every release becomes a game of broken telephone.

> TAKEAWAY
>
> ***The expensive role isn't the climate scientist or the DevOps. It's the person who bridges them.***

Lesson 5 — A new geography should be a config change, not a new project

The trap. You shipped drought in Mexico. Sales closes Chile. Someone says "easy, we duplicate the Mexico code." 4 weeks later, Chile is in production. Except it's now a parallel codebase, with its own tweaks, its own validation logic, its own bugs.

The reality. When the 4th country arrives, you have 4 divergent codebases. Every bug fix must be done 4 times. Every new peril takes 4× the work. Gross margin collapses.

The lesson. A "new geography" should be a new entry in a config file: geographic zone, available datasets, locally calibrated thresholds, possibly a retraining. The application code should never change between countries. If adding a country takes more than 1 day, your architecture isn't modular — it's duplicated.

> TAKEAWAY
>
> ***If adding a country takes more than 1 day, you don't have a pipeline. You have N pipelines.***

The 12-Point Production Readiness Checklist

This is what I systematically verify before signing off "production-ready" on a climate pipeline. Print it. Run through it on yours.

Datasets & Calibration — the foundations

- ☐ **1. Dataset versions are pinned and documented.** Your CHIRPS is v2.0 or v3.0? Your ERA5-Land is v1.x.y? Versions are written somewhere (config, README, code) — not just "we take the latest."
- ☐ **2. Calibration was done on the exact same dataset version as production.** If you calibrated your drought model on ERA5-Land v1.3 and production uses v2.0, you have a latent bug. Verify alignment.
- ☐ **3. Monthly regression test on known historical dates.** You have 3-5 historical dates (e.g., Colombia drought 2018, Hurricane Maria 2017) with expected outputs. A monthly cron replays them and alerts if drift exceeds X%.

Production Infrastructure — the operational layer

- ☐ **4. Source data freshness is checked before pipeline runs.** Before your pipeline launches, it verifies that upstream data (CHIRPS, IMERG, etc.) is current. If not, it waits or notifies — it doesn't run on stale data.
- ☐ **5. Retry logic with exponential backoff for all external API calls.** Copernicus / CDS / NASA can timeout. Your pipeline retries 3-5 times with backoff before giving up. And it gives up **cleanly** (alert, no corrupted output).
- ☐ **6. Failed runs alert a human within 1 hour.** When the pipeline crashes at 3am, a human is notified (Slack, PagerDuty, email) before the client. Not the other way around.

Code Quality — the maintainability layer

- ☐ **7. New geography = config change, not code change.** You can add Peru or Bangladesh by editing 1 config file. If you have to fork a codebase, your architecture isn't modular.
- ☐ **8. Each peril is decoupled from the others.** You can re-ship drought without redeploying heat stress. No monolith where everything breaks together.
- ☐ **9. Deployment is reproducible from a fresh machine.** A new dev can clone the repo, run `make deploy` (or equivalent), and have a prod-like environment in under 30 minutes. If it requires 3 days of setup + 5 undocumented secrets, you have a problem.

Output & Governance — the transparency layer

- ☐ **10. Output files have embedded provenance metadata.** Each output CSV/JSON/NetCDF contains: pipeline version, source dataset versions, timestamp, geo bounding box. If a client says "your March 12 output looks weird," you can trace back.
- ☐ **11. Operational dashboards show pipeline health, not just model outputs.** Beyond the client dashboard (the drought score), you have an internal dashboard (pipeline lag, success rate, dataset freshness, compute time). You see problems before your clients do.
- ☐ **12. Documentation enables your client to run the pipeline without you.** A new data engineer on the client team can take over by reading your docs, without calling you. Otherwise you're a single point of failure — and a commercial risk.

How to use this checklist

1. **Print it.** Walk through your existing pipelines, item by item.
2. **Count your gaps.** Most teams I've audited check 5-8 out of 12 boxes. Below 6 means you're shipping risk to your clients.
3. **Prioritize by category.** Don't try to fix all 12 in parallel. Pick the weakest category and run a focused 2-week sprint on it.
4. **Repeat quarterly.** Pipelines decay. Datasets change. New perils stretch old architecture. This isn't a one-time audit.

What's next

If reading this raised more questions than it answered — or if you've checked 5 boxes and you're not sure which gap to close first — that's exactly the kind of context I help with.

The **Climate Pipeline Discovery** is a fixed 2-week paid engagement (€5-7k). I audit your existing pipeline, map risks and duplication, and deliver a 10-15 page technical note with a scoped remediation plan. No fuzzy consulting calls — concrete output you can use immediately.

📅 Book a 15-min Discovery Call: calendly.com/skaraz_data/15-minute-meeting

Or just hit reply on the LinkedIn DM where you got this PDF — happy to chat.

Stéphane Karasiewicz · Hauts-de-France · Remote worldwide

skarazdata.com · linkedin.com/in/stephane-karasiewicz · skaraz.science@gmail.com