SECURITY & DEPLOYMENT

What runs where, and *what leaves.*

Written to be forwarded. If you are the person who was asked to review this, everything you need is on this page — no call required, no sales pitch in the middle of it.

Most reporting tools answer “is our data safe?” with a policy. The honest version is structural: it depends entirely on what actually runs, and where. This page describes all three shapes of work I deliver, including the one where the answer to “does our data leave our environment?” is *not* a flat no.

> **SCOPE OF THIS DOCUMENT**
>
> This is a one-person consultancy. Everything below is what I actually do, stated so you can check it, not a set of aspirations. Where a claim is verifiable, I have said how to verify it. Where I do not hold a certification, I have said so plainly rather than implying one.

The three shapes

Which one applies is decided with you before anything is built, and written down. I do not let a project drift from the first into the second without saying so, because the security answer changes when it does.

SHAPE	WHAT IT IS	DOES DATA LEAVE YOUR ENVIRONMENT?
Self-contained file	One HTML file you open from your own file share. No server, no install.	**No.** There is nothing for it to leave through.
Automated refresh	A scheduled job reads your data, aggregates it, and republishes the file.	**No, when it runs in your tenant** — the arrangement I argue for. If it has to run on my infrastructure instead, that is a yes, and it is contracted and registered as such.

SHAPE	WHAT IT IS	DOES DATA LEAVE YOUR ENVIRONMENT?
Explorable dataset	Larger data, still queried entirely in your browser rather than on a server.	**No.** The query engine runs in the browser tab.

Shape one — the self-contained file

The deliverable is a single HTML file. It carries its own charting library and its own fonts. It has no script tags pointing anywhere, no stylesheet links, no images loaded from a server, no analytics, no error reporting, no telemetry. Opening it makes zero network requests of any kind.

- **No infrastructure.** Nothing to provision, patch, or pay for.
- **No credentials.** There is no account, no login, no session, no password to manage or revoke.
- **No network egress.** Not restricted egress — none.
- **Works offline,** from a local download, a network drive, or SharePoint.
- **Nothing to breach on my side,** because I hold nothing.

DON'T TAKE MY WORD FOR IT

Every chart, filter and drill-down still works. Or open it with the browser devtools network tab recording and watch it make no requests at all.

That is not a promise about my conduct. It is a property of how the file is built, and you can falsify it in about thirty seconds.

What is inside the file

Usually the data itself, pre-aggregated. That matters: **the file *is* the data**, so wherever it gets forwarded, the data goes with it. Before anything is embedded I check whether it is personal data, whether it is commercially sensitive, whether every embedded column is actually read by a chart, and whether the grain is finer than the questions need. If a view is weekly-by-team, I embed weekly-by-team — not per-person-per-minute.

One file, one audience. If two teams should not see each other's numbers, they get two files with two slices of data, not one file with a filter someone can clear.

After delivery, distribution and access control are yours — your file share, your permissions. I say that explicitly at handover so it is never an assumption.

Values are text, never markup

Every value that comes from your data is written to the page as text, never as HTML. A spreadsheet cell containing a script tag is a realistic accident, not a hypothetical attack, and it renders as visible characters rather than executing. Before delivery I test with a deliberately hostile row and confirm it. Dependencies are pinned to exact versions, taken from the official release artifact, and recorded in the file header so you can review them if an advisory lands later.

Shape two — the automated refresh

This is where the honest answer changes, so it gets the most detail.

Where the code runs

In order of preference:

1. **Inside your tenant** — your Azure subscription, your scheduled job, your storage. Data never crosses a boundary, you can see it and kill it, and there is no bus-factor problem if I disappear. *This is the default I argue for.*
2. **My EU-region infrastructure, with credentials you own** — used only when you have nowhere to run it. Data does cross to me, so it goes in the Article 30 register, a data-processing agreement has to be signed first, and retention is short by design.
3. **A VPS I administer** — only when neither of the above fits.

Access — the exact permission to ask for

The pipeline reads. It does not write back to the source system and it does not hold a credential that could.

On Microsoft Graph, the permission to grant is `Sites.Selected`, as an application permission on an app registration **you own**. It is worth knowing why that specific one:

- `Sites.Selected` grants **no access at all** on its own. Your administrator then grants read on exactly the one site I need. Nothing else in the tenant is reachable, by construction.
- `Files.Read.All` or `Sites.Read.All` would be easier to set up and would let me read **every file in every site collection** in your tenant. If a vendor asks you for either of these to read one folder, that is worth pushing back on — including if the vendor is me.
- If the data genuinely lives in personal OneDrive rather than SharePoint, there is no per-folder application permission available. The right answer there is a dedicated account with access to one shared folder, not a tenant-wide grant.

The app registration belongs to you. You can revoke it in a click, without asking me, and without breaking anything else you own. Credentials are per-project and never reused across clients. After consent I check the *granted* scopes in the console, because what was requested and what was granted are not always the same thing — and I recommend you check them too.

Secrets

Where you are on Microsoft: Azure Key Vault in your tenant, EU region, with access granted to the job's managed identity rather than to a person. Otherwise, that platform's native secret manager in an EU region.

Secrets are injected at runtime as environment variables. Never committed, never logged, never pasted into a chat message or a ticket. Rotation never requires a code change — if rotating a key means editing code, the design is wrong. Keys are rotated at handover and whenever anyone with access leaves.

What happens when it fails

A stale dashboard looks exactly like a fresh one. That is the failure mode that actually hurts, so it is designed against directly:

- Sanity checks run *before* publish — row count within an expected band, no all-null key column, totals within tolerance of the previous run.
- If a check fails, **the job stops and keeps yesterday's good output.** It does not publish something wrong.
- Every published file carries a visible **“data as of”** timestamp, so a stale copy is obvious to the person reading it and not just to me.
- Failures alert me the same day. **A missed run alerts too** — silence is treated as a failure, not as success.
- Logs record events, not payloads: what ran, when, how many rows, what failed. No personal data in log lines.

During setup I trigger a deliberate failure and confirm the alert actually arrives, rather than assuming it would.

Data handling, whatever the shape

Minimisation

I take the columns I need and leave the rest. If a file has names, emails, salaries or free-text notes the dashboard never shows, they get dropped at the first processing step rather than carried around. Where an identifier is needed only to join or count, it gets hashed or replaced. I verify this by diffing the columns in the source extract against the columns the deliverable actually uses — anything in the first list and not the second has to be justified or removed.

Residency

Processing and storage stay in the **EU/EEA**. EU regions for any cloud service, EU-hosted repositories and vaults, no “wherever it lands” defaults. For each service I record the actual configured region string from the console, not the vendor's marketing claim. Any transfer outside the EEA is flagged in the Article 30 register before it happens, not after.

Subprocessors

Any third party that touches your data — hosting, vault, error tracking, email — is listed by name and purpose in the Article 30 register. I do not add one mid-project without telling you first. Adding a service is a two-step change: register row first, integration second.

Anything hosted

Where a deliverable is hosted rather than handed over as a file, it sits behind **Entra ID single sign-on** using your existing identity provider and your existing conditional access policies. No separate user list for someone to forget to deprovision. HTTPS everywhere, HTTP redirects to HTTPS, and I verify the certificate and the redirect before a URL is sent to you.

Separation

Development never points at production data or production credentials — separate app registrations, separate secrets, separate storage. Test data is synthetic or a minimised sample. Anything I publish publicly, including every demo on this site, is **synthetic-only**, generated by a seeded script; no real extract ever becomes a demo, not even lightly edited.

If I disappear

The bus-factor answer has to be structural too, or it is worth nothing.

- It runs in **your** tenant, on **your** credentials.
- You get **full source rights** and the actual source — not a compiled artifact, not a licence to use something I keep.
- Code is documented and the delivered version is tagged in a repository you hold.
- You get a runbook covering what to do when an alert fires after I am gone.
- At least once per project, before handover, I rebuild the pipeline from backup into a scratch environment and run it end to end — then write down the date, what I restored, how long it took, and what was missing. A backup nobody has restored is a guess, not a backup.
- There is no platform to stay subscribed to, no per-viewer licence, and nothing that stops working if you stop paying me.

End of the engagement

When an engagement ends: credentials are rotated, my access is removed, and your data is deleted from my machines and from any service I stood up — the working folder, the backups that contain it, and the trash, with the specific backups named.

Ideally I am holding nothing to begin with, because the self-contained shape never required me to keep a copy. Where I did hold data, deletion is confirmed in a handover document that you countersign. The engagement is not finished until that is signed and every row in the access register has a revocation date against it.

What I do not claim

> READ THIS PART FIRST, IF YOU ARE SHORT OF TIME
>
> I am **not ISO 27001 certified** and I am **not SOC 2 audited**, and I do not imply either. My internal runbook is organised loosely along the ISO/IEC 27001:2022 Annex A control themes so you can map my answers onto your own framework — that is a self-assessed alignment by a one-person consultancy, and nothing more than that.
>
> I also do not use the word “compliant” without naming what with. If you need a certified supplier, tell me and I will say so rather than talk around it.

Professional indemnity insurance is in place — RC Professionnelle, held in Belgium where I am established. You can have a copy of the certificate for your file; it names the insurer, the policy number and the period of cover. On the data processing agreement: **I sign yours**. Most legal teams already have a controller-to-processor template they would rather use than review someone else's, and I do not draft clauses myself. Where you have no paper of your own I start from a standard controller-to-processor model — or the European Commission's standard contractual clauses where they apply — reviewed by a qualified professional before I sign. Where I am processing personal data on your behalf, that paperwork is signed *before* the first extract, not retro-fitted. I do not accept a data file until it is in hand.

There is a one-page incident plan — detect, contain, tell you, assess the 72-hour GDPR question, write it up — filled in per project with your named contact and an agreed notification window. I walk through it once at kickoff so I am not reading it for the first time during an incident.

Questions you should ask any vendor

Including me. If a supplier cannot answer these quickly and specifically, that is information.

1. **Exactly which permission are you asking for, and what else does it reach?** “Read-only” is not a scope. Make them name it, then check what was actually granted in the console rather than what was requested.
2. **Who owns the credential — you or us?** If you cannot revoke it yourself, in a click, without asking them, you do not control your own data.
3. **Where does the compute actually run, and in which region?** Ask for the configured region string, not the marketing page.
4. **What happens when a refresh fails?** If the answer is not “it keeps the last good version and tells someone”, you will eventually make a decision on stale numbers without knowing it.
5. **Does a missed run alert, or only a failed one?** Most tools only catch the second. Silence is the expensive failure.
6. **Name every subprocessor that touches our data.** Hosting, vault, error tracking, email. If the list takes them a while to produce, it is not being maintained.
7. **What do you hold at the end, and how is it destroyed?** Ask for it in writing, countersigned.
8. **What happens if you disappear?** Do you hold the source, the credentials and the ability to run it — or a licence to something they keep?
9. **What are you certified for?** If the answer is a framework name without the word “certified” or an auditor attached, it is alignment, not certification. That can be fine — it just should not be sold as the other thing.