

Nomad Universe and Gentoo Tickets: running technology for a three-company group

Three affiliated UAE ventures share one commerce platform: a sports-tourism company, a ticket-distribution business, and a multi-tenant travel-commerce product that sells official Rugby World Cup 2027 packages under several...

Role: Group CTO and Senior Technical Consultant, sole technology owner

Scope: NOMAD Events LLC, Gentoo Tickets, Nomad Universe. Internal IT, cloud infrastructure, the shared commerce platform, third-party ecosystem, agency governance, security and compliance.

Stack: Next.js 16 (multi-tenant storefronts), Go with GraphQL (API), React and MUI (admin), PostgreSQL on AWS RDS, ECS Fargate, S3 and SES, Cloudflare (DNS, R2, Pages, Workers, Access), GitHub Actions, Notion and Jira for delivery.

Live: nomad-universe.com, kt.nomad-universe.com, groups.nomad-universe.com, campaign.kuonitumlare.com/rwc2027



Context

Three affiliated UAE ventures share one commerce platform: a sports-tourism company, a ticket-distribution business, and a multi-tenant travel-commerce product that sells official Rugby World Cup 2027 packages under several brands, including a partnership with Kuoni Tumlare. Day-to-day engineering is delivered by an external agency, TMA. My job is to own everything the agency does not: the architecture, the cloud estate, the vendors and integrations, the governance that makes the agency ship reliably, and the compliance posture the clients ask about. When something is urgent or sits outside the agency's remit, I fix it myself.

This is the most thoroughly evidenced project in my portfolio because it is the current one. Everything below is from work done in the last few months, not reconstructed from an old repository.

What I built and run

A multi-tenant storefront platform

One Next.js 16 codebase serves several branded storefronts. Tenant scoping is done with `basePath` routing, so the KT consumer site, the KT ICC site and the KT B2B corporate-packages site are separate products to a visitor and one deployable to us. A custom image loader was needed because the built-in optimizer does an internal self-fetch that ignores `basePath`; that bug took down every image on the B2B site after the `basePath` change and I traced it through the ECS logs to the optimizer, then fixed it with an explicit `BASE_PATH` prefix on every local asset. Locale-aware routing uses `next-intl` v4. The API is Go with GraphQL, shared by the storefronts and a React admin.

The Kuoni Tumlare partnership site

The RWC 2027 partnership site runs origin-served from our infrastructure and edge-proxied through KT's own CloudFront, at campaign.kuonitumlare.com/rwc2027. The B2B corporate-packages site lives in its own smaller repository so I can hot-fix it directly without the agency cycle. In one week that path handled a full UAT punch list from the client: a double-submit bug that had created a duplicate enquiry in production, missing security headers, SEO metadata that was computed in code but never rendered, five legal pages ported from the consumer site, a cookie-consent banner that did not exist, and three rounds of design feedback on those pages. Six pull requests, each merged and deployed the same day, each verified against the live site before I reported it done.

Per-tenant transactional email

Every brand sends from its own identity. I designed the pattern that makes that work: SES credentials derived with SigV4 per tenant, stored in AWS Secrets Manager, resolved at send time so a tenant can never send as another tenant. It powers all transactional email across the group.

Infrastructure decisions

Moving the front door to Cloudflare

I ran the group's migration from an AWS-only estate to a hybrid one: nameserver cutover with no downtime, S3 uploads moved to R2, and the frontends handed progressively to Cloudflare Pages and Workers with a GitHub Actions pipeline that deploys through Wrangler and pulls its API token from Secrets Manager rather than a repository secret. Cloudflare Access sits in front of the staging admin. When the agency later needed extra token scopes to automate their own deploys, I checked the request against what Wrangler actually needs before granting it, and granted exactly that.

Separating staging from production

Staging and production had shared one RDS instance and one uploads bucket since launch. I split them. The target database sits in a private subnet with no bastion, so I ran the migration from a disposable Fargate task in the same subnets, with a security-group rule opened only for the migration window and an IAM role created only for that task, and tore all of it down afterwards. Production was never touched. The one real mistake along the way was mine: a shell-quoting bug let bash expand placeholder text inside a JSON override before Python saw it, which produced an empty connection string on the first attempt. I rewrote the step to read from disk with no interpolation and it ran clean.

Governance that makes an agency ship

I set the delivery workflow the agency works inside: specifications written in Notion using a PRD template, a "Ready for Jira" gate that hands work to execution, code review on GitHub with CODEOWNERS enforcement, and a separate hot-fix path so small urgent changes do not have to wait for a full cycle. I administer Jira, Confluence and Notion for the group and I am the single approver for agency scope, sprint contents and architectural changes. Every piece of work, including work I hand to the agency, is logged in one planner so nobody has to ask what is in flight.

The most useful rule in that system is knowing which side of the line a change belongs on. A UAT checklist for the consumer storefront produced eighteen findings that live in the agency-owned monorepo; I filed those as GitHub issues for TMA with reproduction steps and suggested fixes. The same audit's findings for the B2B site were mine to fix, and I fixed them the same week.

Security and compliance

A client asked us to review the agency's access to production data. I audited the account directly rather than reading the policy names. The policy called `ReadOnlyStaging` was scoped to every resource in the account, because until the RDS split there was no staging to scope it to. One engineer held an `iam:PassRole` that could hand any role in the account to S3 Batch Operations, which is a bulk-export path. There was no CloudTrail trail at all. I wrote a three-phase remediation plan, executed the parts that were safe to execute immediately (scoping the `PassRole` to the one legitimate execution role, standing up an account-wide CloudTrail trail with S3 data-event logging on the production uploads bucket), and sequenced the rest behind a maintenance window and a heads-up to the two engineers whose access it changes, because pulling access mid-project would have broken their in-flight work.

Separately: two rounds of a security audit across the Gentoo Tickets backend, more than eighty tracked findings covering webhook replay protection, secret rotation, admin-domain validation, GraphQL introspection lockdown, CSP and SRI headers and structured audit logging. An EAA and WCAG 2.1 AA assessment delivered to KT's IT team. A Schedule 4 data-security plan authored for a Mubadala engagement.

Incidents, and how they were diagnosed

When the cookie-consent banner vanished from the storefront, the code was fine: the script tags were present and the initialiser was correct. The consent vendor's CDN was returning an S3 `AccessDenied` error for our project bundle, because the free plan had lapsed. I confirmed it with a direct fetch, the account was upgraded, and I verified the fix on both affected domains before closing it. While I was in that code I also found and documented a real race condition in how the banner initialises, unrelated to the outage, so it would not bite later.

When the admin showed stale asset domains after a deploy, the first theory was a Cloudflare traffic-split misconfiguration. It was not. I compared the chunk hashes in the GitHub Actions build log with what the browser had actually loaded and proved the deploy was correct all along; the edge was serving a cached `index.html`. One cache purge fixed it. Diagnosing by evidence rather than by the most convenient theory is most of the job.

Outcomes

- Zero-downtime cloud migration and a zero-downtime staging/production split.
 - One deployable serving multiple brands, with a partner site live on the partner's own domain.
 - A delivery system an external agency ships through predictably, with clear ownership rules.
 - A compliance posture backed by real controls (least-privilege IAM, audit logging) rather than policy names.
 - Production issues diagnosed and closed the same day, with the evidence written down.
-

Evidence

Live sites listed above. Pull requests and issues on the Gentoo-Tickets GitHub organisation. Architecture diagrams for the migration and the storefront routing are on asadaliawan.com, drawn from the code.