Sharif Sule Ndlovu

SOFTWARE DEVELOPER · DEVOPS ENGINEER · DEVBRIDGE

Pretoria, South Africa

Documentation of engineering work across three production Django systems and one ongoing social programme, written as proof of work rather than summary. Each case study records the project as built, the specific problems encountered, how they were resolved, and what the resolution taught — including the mistakes and what they cost.

CONTENTS

AT A GLANCE

2
government platforms live in production

10
provincial departments on one Django deployment

96
government submission forms maintained

3
AWS environments owned end-to-end

CONTACT

sharifndlovu@gmail.com
knowledgebase.devbridge.co.za
portfolio.devbridge.co.za

PROJECT OVERVIEW

A Django e-commerce and content-delivery platform commissioned by a private client to sell access to workplace compliance training — public accountability, policy, finance and employment modules. It carried recurring subscription billing via Paystack, a session-based cart, Google OAuth2 login, content gating by subscription status, and async email and task processing on Celery and Redis.

The client cancelled before go-live and pivoted to a book and blog model. The engagement was used to go well beyond the brief — raw WebSockets, async Django, low-level payment integration — and became the technical foundation for every government system that followed.

KEY CHALLENGES & RESOLUTIONS

Paystack webhooks — the payment lifecycle from first principles

Coming from a .NET background where payment libraries hide the flow, this demanded a ground-level model of webhooks: the provider calls your server, not the reverse. Every subscription event was implemented and reasoned through — *charge.success*, *invoice.payment_failed*, *subscription.disable*, *subscription.expiring_cards* — with correct state transitions, proactive card-expiry mail before the next billing date, and soft failures (retryable decline) distinguished from hard cancellations. Every inbound webhook was signature-validated to block spoofed events.

Docker container networking — service names, not localhost

An early runtime failure: services could not reach each other inside Compose. The cause was Docker's network model — containers on the same network resolve each other by service name, so a Django app dialling *localhost:5432* never finds PostgreSQL in a separate container. Bridge mode (isolated network, DNS-resolved names) versus host mode (shared host stack) became foundational for every containerised project after it.

Migration surgery — integer primary keys to UUIDs, mid-project

With the data model already live, privacy and API safety required moving off auto-increment integers, which expose record counts and allow enumeration. Changing a primary key referenced by foreign keys across tables requires a precise sequence: add the UUID column, backfill, drop the old constraint, promote the column, repoint every foreign key — learned through PostgreSQL constraint syntax, transaction safety and *RunSQL* inside Django migrations. The ORM abstracts SQL; production schema changes do not.

STACK

Django 5 · Celery · Redis · PostgreSQL · Paystack · AWS S3 · Docker Compose · Google OAuth2 · Django Channels

ROLE

Sole developer — application, payments, infrastructure and deployment.

STATUS

Cancelled before go-live; client pivoted to publishing. Retained as the technical groundwork for e-PGLUM and OTP.

Channels, Redis and async — closing the gap deliberately

Realtime notifications required Django Channels, which introduces ASGI and Python's async/await model alongside the synchronous request cycle. The difficulty was conceptual: knowing when code runs sync versus async, and why mixing the two wrongly causes deadlocks or dropped connections. Redis served two roles at once — Channels layer for message passing between consumers, and cache backend — which had to be configured so the uses stayed isolated. The gap was closed through systematic study, and the time spent on raw WebSockets built a concurrency intuition that paid off directly on OTP.

WSGI versus ASGI, and the first AWS infrastructure

Static and media files moved off the local filesystem into S3 — the first real AWS engagement — which meant understanding IAM roles, bucket policies, presigned URL generation and how Django's storage backends wire into boto3. Alongside it, the difference between WSGI (synchronous, one request per worker) and ASGI (asynchronous, long-lived connections) became concrete, as did why the development server is not production and what Gunicorn provides. Cloud storage and production server selection became core competencies carried into both government systems.

IMPACT / OUTCOME

The platform was never deployed — the client cancelled and pivoted to publishing. The professional outcome was significant: every major concept underpinning the later government systems — async task queues, containerisation, cloud storage, payment webhooks, OAuth, raw database migrations and async Django — was first encountered, broken and understood here. This is the project that turned framework familiarity into engineering depth.

CARRIED FORWARD

- Celery + Redis async task patterns
- Containerisation and network modes
- S3, IAM and presigned URLs
- Webhook verification and state machines
- Raw SQL migrations
- ASGI, Channels and async Django

PROJECT OVERVIEW

e-PGLUM is a nationwide electronic land-use management platform serving South African municipalities, live at epglum.co.za. It digitises a previously paper-based submission and assessment process for land-use applications, building plans, outdoor advertising, objections and appeals, deployed across QA, UAT and Production on ECS Fargate over RDS PostgreSQL with PostGIS, ElastiCache Redis and S3.

I was the AWS infrastructure owner for the full project — designing and running the cloud environment across all three tiers — while contributing to the Django backend alongside three developers, a frontend developer, testers and a project manager.

KEY CHALLENGES & RESOLUTIONS

96 submission forms — mixins as a survival strategy

The platform required 96 distinct form classes covering land-use applications, building plans, outdoor advertising, objections, appeals, rezoning, township establishment, subdivision and consolidation. They overlapped heavily — applicant identity, property location, municipality references, attachment handlers, near-identical validation — so avoiding duplication meant composing shared behaviour with Python mixins. A new form type assembles from existing mixins in minutes rather than hours of copy-paste; the same pattern extended to models and class-based views.

Report generation — visual analytics at scale

Reports had to produce graphs, summary tables and statistics scoped by municipality, submission type, date range and status — the first production use of Matplotlib, openpyxl/xlsxwriter and WeasyPrint. Generation initially ran inside the HTTP request cycle and began timing out as real data accumulated, so the pipeline moved to Celery: the view triggers a task, returns a job ID immediately, and the frontend polls for completion. Anything longer than a few seconds does not belong in the request cycle.

WAF blocking in production — the cost of environment mismatch

At go-live, AWS WAF managed rules blocked legitimate requests that had passed in lower environments: large submissions tripped *SizeRestrictions_BODY*, certain field content tripped XSS rules. Nothing had surfaced earlier because WAF was configured leniently there. Resolution meant reading CloudFront and WAF logs, identifying which managed rule group matched which request, and writing byte-match exclusions scoped to specific URI paths. Production security controls belong in every environment from the start — a lesson that directly shaped OTP.

STACK

Django 5 · ECS Fargate · RDS PostgreSQL · PostGIS · ElastiCache · CloudFormation · AWS WAF · S3 · EFS · Celery · WeasyPrint · Matplotlib

ROLE

AWS infrastructure owner across QA, UAT and Production; backend contributor in a team of seven.

STATUS

Live in production — epglum.co.za

Large file uploads — tuning timeouts across the whole path

Building plans, title deeds and site development plans regularly exceeded 20 MB, and at that size defaults across the stack became the problem: CloudFront's origin response timeout, the ALB idle timeout and Gunicorn's request timeout each failed silently when crossed. Fixing it required treating browser → CloudFront → ALB → ECS task as one path and tuning every layer consistently — and raised presigned POST URLs for direct-to-S3 uploads as the pattern for future projects.

First serious AWS estate — full ownership from scratch

A complete production environment designed and owned end-to-end: VPCs with public and private subnets, least-privilege security groups, RDS PostgreSQL in a private subnet, ElastiCache Redis, ECS Fargate services for web and Celery workers, an ALB with health checks, S3 for static and media, Route 53 and ACM. Every component had to be wired correctly — a wrong security group means no database; a wrong subnet means ECS tasks cannot pull images. CloudFormation was adopted so the three-environment setup was repeatable.

Nightly statistics — post-deployment broker integration

Two months after go-live, e-PGLUM had to publish nightly statistics to the same RabbitMQ middleman used by the OTP mobile integration. A Celery Beat task aggregates application counts by submission type, registrations, active versus static users, stalled applications and daily and weekly totals into a payload matching the broker's schema exactly. The constraint was efficiency: poorly written aggregations lock rows or trigger slow scans on a live database, and the task had to finish inside a narrow nightly window.

Copilot CLI deprecation — migrating live infrastructure

With AWS announcing Copilot CLI end-of-life for June 2026, both e-PGLUM and OTP were migrated to standalone CloudFormation managed by DRY bash scripts — reverse-engineering the generated stacks, extracting reusable patterns, and rebuilding them as templates with per-environment parameter files. The result is a deployment system split by concern (*push-env*, *push-image*, *deploy-env-addons*, *deploy-all*, *release*), completed across two codebases while both remained in active development.

IMPACT / OUTCOME

Live in production at epglum.co.za, replacing a manual paper-based application process with a digital submission and assessment workflow — and establishing me as infrastructure owner on government-scale Django deployments across three tiers.

ENVIRONMENTS

QA — feature verification

UAT — municipal acceptance

Production — public traffic

DEPLOY SCRIPTS

push-env.sh
push-image.sh
deploy-env-addons.sh
deploy-all.sh
release.sh

PROJECT OVERVIEW

OTP — branded Livi Lemphakatsi — is a multi-tier government case management system serving the Office of the Premier, Mpumalanga, across 10 provincial departments. Citizens submit cases from a mobile app: service-delivery complaints, missing persons, vacancies, crime hotspots, community projects. The Django portal handles intake, routing, assessment, inter-departmental collaboration and resolution, with case workers operating under SLA enforcement.

The architecture is queue-driven: mobile app and portal speak through a RabbitMQ middleman broker rather than directly, giving loose coupling and a clear contract boundary. Each department runs on its own branded subdomain, and POPIA compliance is met by deploying production in af-south-1 (Cape Town).

KEY CHALLENGES & RESOLUTIONS

Architecture evaluation — simplicity over workflow engines

Before any application code, the team formally evaluated Camunda, Salesforce-adjacent tooling, django-viewflow, n8n and multi-tenant frameworks including django-tenants — and scrapped all external workflow engines. Their complexity outweighed their value for a system whose "workflow" is a case moving through statuses. The status-as-state-machine pattern from e-PGLUM was extended (NEW → IN_PROGRESS → RESOLVED / DUPLICATE / CANCELLED) with SLA escalation layered on via Celery Beat.

Ten departments, one application, ten identities

Each of the ten departments runs on its own subdomain with its own primary, secondary and accent colours and custom CSS, applied dynamically by *DepartmentThemeMiddleware* reading the Host header on every request. Architecturally it is one Django application and one database; visually it must feel like ten systems — so every view, template and email had to be theme-aware.

Thirty subdomains — no wildcards where it counted

Ten departments across three environments means 30 subdomains. A wildcard certificate covers TLS, but WAF rules, CSRF trusted origins and session cookie domains all needed explicit entries, because the theme middleware resolves department identity from the exact subdomain string. Any mismatch between a registered subdomain and the middleware's expectation fails silently, so every entry had to be set precisely in the database and kept in sync with DNS across all environments.

STACK

Django 5.2 · ECS Fargate · PostgreSQL · PostGIS · Amazon MQ · Redis · Celery · SSE · ASGI · CloudFormation · AWS WAF · Rekognition · pgvector

DEPARTMENTS

Agriculture · Safety & Security Liaison · Economic Development & Tourism · Co-operative Governance & Human Settlements · Culture, Sport & Recreation · Health · Office of the Premier · Public Works · Social Development · Social Services

STATUS

Live in production — livilemphakatsi.co.za · af-south-1

Queue architecture — from SQS to RabbitMQ on the middleman

SQS was implemented first; as the mobile integration matured it became clear the middleman ran RabbitMQ and expected both sides to publish and consume there. OTP publishes case updates, status changes and responses outbound, and consumes inbound queues (*q_livi_case_activities*, *q_livi_case_chat*, *q_livi_missing_persons*, *q_livi_vacancies*) for mobile submissions. Migrating meant absorbing the protocol difference — HTTP-polled SQS versus long-lived AMQP connections through pika — and re-implementing publisher and consumer logic against the middleman's schema. Every message is recorded in *RxQueueLog* and *TxQueueLog* for traceability and replay.

WAF parity — production-level firewall in QA from day one

The costliest infrastructure mistake on e-PGLUM was lenient WAF rules in lower environments. On OTP, QA carried the same managed rule groups as production from the start, so anything that would be blocked in production was blocked immediately during internal testing — with the team present. Turnaround dropped from days under production pressure to hours.

Realtime chat — SSE, Redis pub/sub and the ASGI transition

Case workers and citizens converse in a threaded chat per case. Live delivery uses Server-Sent Events over async Django views on Daphne, with Redis pub/sub decoupling persistence from streaming: the message is saved, a signal publishes it to a Redis channel, and the SSE view streams it out. A Last-Event-ID recovery scheme in ISO8601#UUID form handles reconnects without duplicates, and the feature shipped behind a *CASE_CHAT_REALTIME_ENABLED* flag so it could be toggled without redeployment.

Health checks — five per group, used as diagnostics

Each group runs five checks: database, Redis cache, S3, migrations applied and Celery worker availability. Passing all five consistently required careful ECS startup ordering, since a missing security-group rule or IAM permission fails the check and puts the task in a replacement loop. That turned the endpoint into a diagnostic: a cycling task means one specific dependency is unreachable, and the response says which.

IMPACT / OUTCOME

Live across ten government departments. Scrapping workflow engines for a status-driven case model produced a system case workers operate without specialised training, and the WAF parity practice established here is now standard across DevBridge deployments.

CASE LIFECYCLE

New

In progress

Resolved

Duplicate

Cancelled

HEALTH CHECKS

Database · Redis cache · S3 storage · migrations applied · Celery workers

Programme — Community Web Modernization

A DEVBRIDGE SOCIAL INITIATIVE

WHAT IT IS

A DevBridge initiative providing modern, secure, sustainable static websites to community organisations across South Africa, at low or no cost for qualifying recipients. It exists because many community institutions — estates, schools, churches, NGOs, sports clubs, local initiatives — have either no web presence or maintain sites that are slow, insecure and no longer representative of the communities they serve.

HOW IT WORKS

Every site ships as a modern static website — no server, no database, no CMS to patch — deployed to Cloudflare's edge network with automatic HTTPS, DDoS protection and sub-second global loads. The architecture is deliberate: removing the traditional server stack drops operating costs to near zero and keeps the site secure and functional indefinitely with minimal maintenance. Organisations receive full ownership of the files and choose between managed content updates or a self-service CMS integration. Delivery runs in four steps: initial conversation, design, build and deploy, handover with full file ownership.

WHY IT MATTERS

The programme sits outside the government contract work and shows a different dimension of it: product thinking in designing a repeatable delivery model, client acquisition and management across diverse organisations, frontend design and build, and a deliberate decision to apply the same infrastructure discipline used on e-PGLUM and OTP to community-scale problems. It is ongoing, not finished.

LIVE CLIENTS

- Reliance Pro
- Makgotoso Motloung
- Rehoboth Boerdery
- Sknn
- Dream Engineering
- I-Ntuthuko Enterprise

STACK

Static HTML/CSS/JS · Cloudflare CDN · Cloudflare DNS · Git deploy · GSAP · Lenis

Certifications

Certifications were pursued in parallel with live project work — not as prerequisites, but as structured consolidation of concepts already being applied in production. Each maps to a phase of the project timeline.

AWS Certified Cloud Practitioner

EARNED DURING THE WORKPLACE BASICS

Formalised the foundational AWS knowledge being built through the first S3 and IAM implementations.

AWS Solutions Architect — Associate

EARNED DURING THE WORKPLACE BASICS

Provided the architectural vocabulary for VPC design, multi-AZ deployment and service selection — applied directly on e-PGLUM.

AWS Developer — Associate

EARNED DURING THE WORKPLACE BASICS

Deepened understanding of ECS, Lambda, SQS and the deployment patterns used throughout the government systems.

CompTIA Network+

BEFORE E-PGLUM

Networking fundamentals — subnets, routing, DNS, TLS — underpinning the full VPC and domain work on e-PGLUM.

CompTIA Project+

BEFORE E-PGLUM

Project management fundamentals relevant to operating as infrastructure owner in a structured seven-person team.

Microsoft Azure Fundamentals (AZ-900)

EARNED DURING OTP

Broadened cloud platform knowledge beyond AWS; credential linked via Microsoft Learn.

Azure Developer Associate (AZ-204)

IN PROGRESS

In progress — exam scheduled for approximately the week of 30 June 2026.

VERIFY

Credentials and detail available at knowledgebase.devbridge.co.za

CONTACT

sharifndlovu@gmail.com
Pretoria, Gauteng
South Africa

ELSEWHERE

devbridge.co.za

portfolio.devbridge.co.za

github.com/sharifndlovu123

LinkedIn — Sharif Ndlovu