

The 4-Week AWS Migration That Paid for Itself

From On-Prem to AWS in 4 Weeks

Case Study

Siyakha Consulting

Executive summary

What we did, and what it changed

Siyakha maintains an educational platform for a growing South African user base. It lived on a single virtual private server, and it had reached the ceiling of what one machine could serve.

We moved the applications, their databases and every byte of user data and file storage onto AWS in four weeks, with the old environment serving users until the final week. The platform now runs as containers across two availability zones, scales with load, and deploys from a pipeline instead of a shell session.

MIGRATION WINDOW

4 weeks

Kickoff to cutover
Old environment live throughout

HOSTING COST

-40% /mo

vs the previous bill
Before further optimisation

PEAK LOAD

8x

Concurrent users
Verified in load testing

The challenge

Four problems, one root cause

A bigger server answers none of these, because all four follow from there being exactly one server.

The upgrade treadmill

Every growth spurt meant paying for a larger machine, and the next spurt meant paying again.

One failure domain

Every application and the database shared a machine. Anything that took it down took everything down.

Capacity sat idle

The server had to be sized for the busiest week, so most of the time it was oversized and still costing full price.

Deploys were manual

Shipping a change meant a shell session on the production server, with no way to roll it back cleanly.

Why four weeks

The preconditions that make the timeline honest

Four weeks is what it takes when three things are true

The applications containerise without a rewrite

If week one shows the application needs restructuring first, the timeline changes and we say so before week two.

Someone on the client side decides quickly

Most slow migrations are not slow because of engineering. They are slow because decisions have to go through several approval processes.

The data tolerates replication-based migration

We stand up the new databases, replicate continuously, and cut over once the data has caught up. No long maintenance window.

When it is not four weeks

Tightly coupled legacy components, data that cannot be replicated live, or compliance sign-off can extend the migration window. Better to find that out in week one than in week three.

The starting point

Start on credits, not spend

AWS funds new accounts through several programmes. The credits are applied before anything is migrated.

AWS Activate credits

Tiers run from \$1,000 for a self-funded founder up to \$200,000 for companies backed by an approved accelerator, VC or AWS partner.

Proof-of-concept funding for AI and ML

AI and ML pilots are funded separately from Activate, which covers GPU and inference spend while a model is still being proven.

\$200 for finishing the new-account checklists

AWS grants a further \$200 in promotional credit for completing its new-account setup checklists. The steps are highlighted here: github.com/JoeyAlpha5/aws-new-account-credit-booster

Enough runway to prove the architecture

Together these cover a small-to-mid migration and its first months of running, so the design is validated against real traffic on credits rather than against invoices.

Baseline first, then right-size

The first month after cutover, still inside the credit window

Sizing decided on day one is a guess. The month after cutover turns it into a measurement, and the account is corrected before it ever bills.

Let it run before tuning it

The first full month carries real traffic on the new architecture, so every resource has a measured demand behind it instead of an assumption.

Right-size against the numbers

Compute, database and storage are resized against what the month actually showed, which is where the bulk of the saving comes from.

Sunbird Insyte audits the account

Insyte reads the account and generates FinOps recommendations: idle resources, oversized instances, storage classes worth moving, and commitments worth taking.

All of it before a card is charged

The work happens while credits still cover the account, so the first paid invoice lands against an already-optimised environment.

Approach at a glance

What is live by the end of each week

Sizing decided on day one is a guess. The month after cutover turns it into a measurement, and the account is corrected before it ever bills.

Week	Theme	Live and signed off by end of week
1	Foundation and networking	VPC, subnets across two availability zones, NAT, security groups, VPC endpoints, IAM roles, secrets store, Terraform state backend. No application yet.
2	Data and storage	Managed MySQL on RDS, Multi-AZ, databases migrated and verified. Shared file storage and object storage ready. Production configuration loaded.
3	Application	Container images built and stored. Web service serving HTTPS behind the load balancer, worker service running scheduled jobs and the queue. Autoscaling proven under load.
4	Cutover and hardening	Production live on AWS at the real domain with real data. Pipeline deploys on push. Monitoring and alarms live. Load test passed. Old environment drained.

Week 1 – Free Assessment

Build the empty, secure environment everything else lands in

Nothing user-facing ships this week. That is the point, the platform keeps running untouched while the destination is built.

- Assess every application, database, scheduled job and external integration, and list the dependencies.
- Define the cutover criteria: what would have to be true before we were willing to move live traffic.
- Everything is configured in Terraform. VPC, public and private subnets across two availability zones, NAT, route tables.
- Configure every security group up front, referencing each other rather than open address ranges, so no tier is reachable except from the tier above it.
- Added VPC endpoints so the private tier reaches AWS services without traversing the public internet.
- Created IAM task roles and a secrets store, so no credential ever lives in a file on a server.

Week 2 – Move the data

The new environment quietly catches up with the old one

The databases and file storage migrated, while users carry on using the old environment.

- Configure the managed MySQL on RDS in a Multi-AZ configuration, so a zone failure is a failover rather than an outage.
- Started replication from the existing environment and let it run. Replication is what removes the long maintenance window.
- Moved user uploads and file storage to shared elastic file storage and object storage, with the new configuration pointed at the new locations.
- Encrypted storage at rest, enabled backups with a retention window, and turned on deletion protection.
- Verified repeatedly that what's replicated matched what left – row counts, checksums and spot reads, not a single import and a hope.

Week 3 – Modernise

Where migration stops being a move and becomes an upgrade

The applications run on AWS as containers, behind HTTPS, scaling on real load, reachable on a staging address, not yet on the real domain.

- Containerised both applications with production builds, and pushed the images to a private registry.
- Deployed a web service on ECS Fargate across private subnets in two availability zones, registered behind an application load balancer.
- Deployed a separate worker service for scheduled jobs and the queue, isolated from web traffic so a traffic spike cannot starve background work.
- Terminated HTTPS at the load balancer with a managed certificate and redirected all traffic to it.
- Configured autoscaling on both cpu utilisation and request count per target, so the platform responds to actual load.
- Configured autoscaling on both cpu utilisation and request c

Week 4 – Cutover

Cut over, harden, baseline

Production traffic on AWS at the real domain deploys automated monitoring live.

- Took the final data synchronisation from the old environment and verified it before moving any traffic.
- Moved DNS to the load balancer, moving the production traffic over to AWS.
- Replaced manual deploys with a pipeline that builds the image, registers a new task revision and updates the service – authenticating to AWS with short-lived credentials.
- Turned on container-level monitoring, dashboards and alarms for service utilisation, load balancer errors, unhealthy targets and database health.
- Load tested the new environment at several times the platform's previous peak, reproducing the exact conditions that used to take the old server down.
- Tightened egress rules, reviewed network access and finalised scaling policies.

How we de-risked the cutover

The parts that make week four boring

A cutover is the one irreversible-feeling moment in a migration. These five things are what make it reversible.

The old environment kept serving users

Right up to week four. Nothing about weeks one to three could affect a single user.

Rollback was a DNS change

The old environment stayed standing after cutover. Reverting meant pointing a record back, not restoring a backup.

Validated on a staging address first

The full platform ran on AWS, under load, before the real domain pointed anywhere near it.

Health checks gate every deploy

A build that fails its health endpoint stops rather than replacing a working one.

The load test reproduced the actual failure

We did not test a synthetic scenario. We recreated the conditions that used to bring the old server down, and watched autoscaling absorb them.

The results

Four weeks, measured



Sizing decided on day one is a guess. The month after cutover turns it into a measurement, and the account is corrected before it ever bills.

	Before	After
Hosting	A single virtual private server	Containers across two availability zones
Scaling	Buy a bigger machine, redeploy	Automatic, on processor load and request volume
Database	On the same machine as the applications	Managed, Multi-AZ, private, encrypted, backed up
Failure domain	One machine, everything on it	Zone-redundant; a zone loss is a failover
Deploys	Manual shell session on production	Pipeline on push, health-gated, no stored keys
Peak capacity	At its ceiling	8x the previous concurrent users
Monthly cost	Rising with every upgrade	40% lower, and falling

Is this your platform?

The four signals that this case study is about you

- 1 Your application runs on one server, and the answer to growth has been to buy a bigger one.
- 2 You pay for peak capacity all year and use it for a few weeks.
- 3 A deploy means someone connecting to production by hand.
- 4 You have been told a cloud migration is a six-month project.

If two or more of those are true, the free assessment is a week and it tells you whether four weeks is realistic for your platform. If it is not, we will say so.

Let's talk

Cloud migration, modernisation, and FinOps on AWS.

info@pyxis-hub.com

www.pyxis-hub.com

