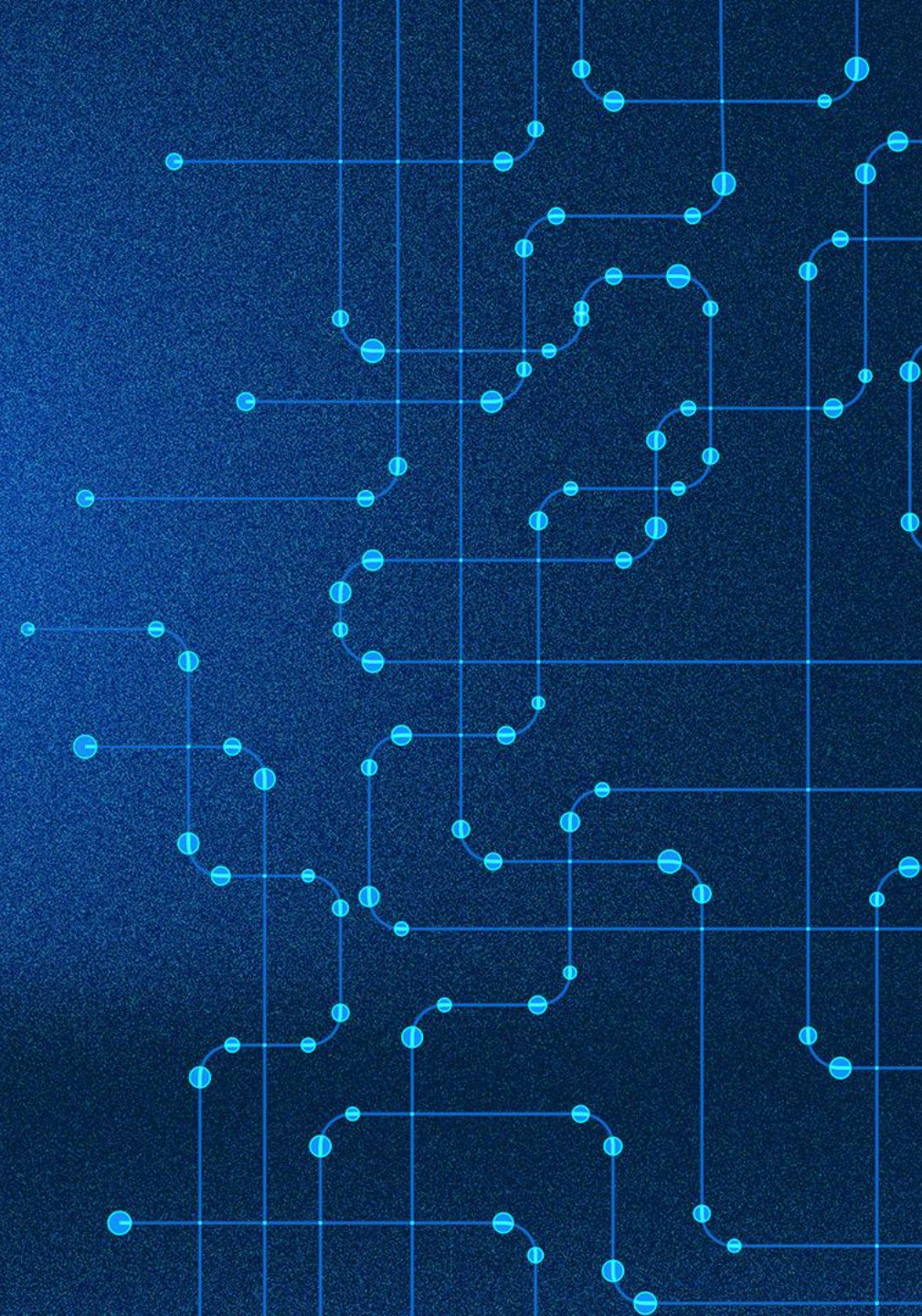
COOi
Co Open Innovation
studios

Enterprise Application Migration & Modernization

Amazon Web Services Playbook

Table of Contents



Introduction

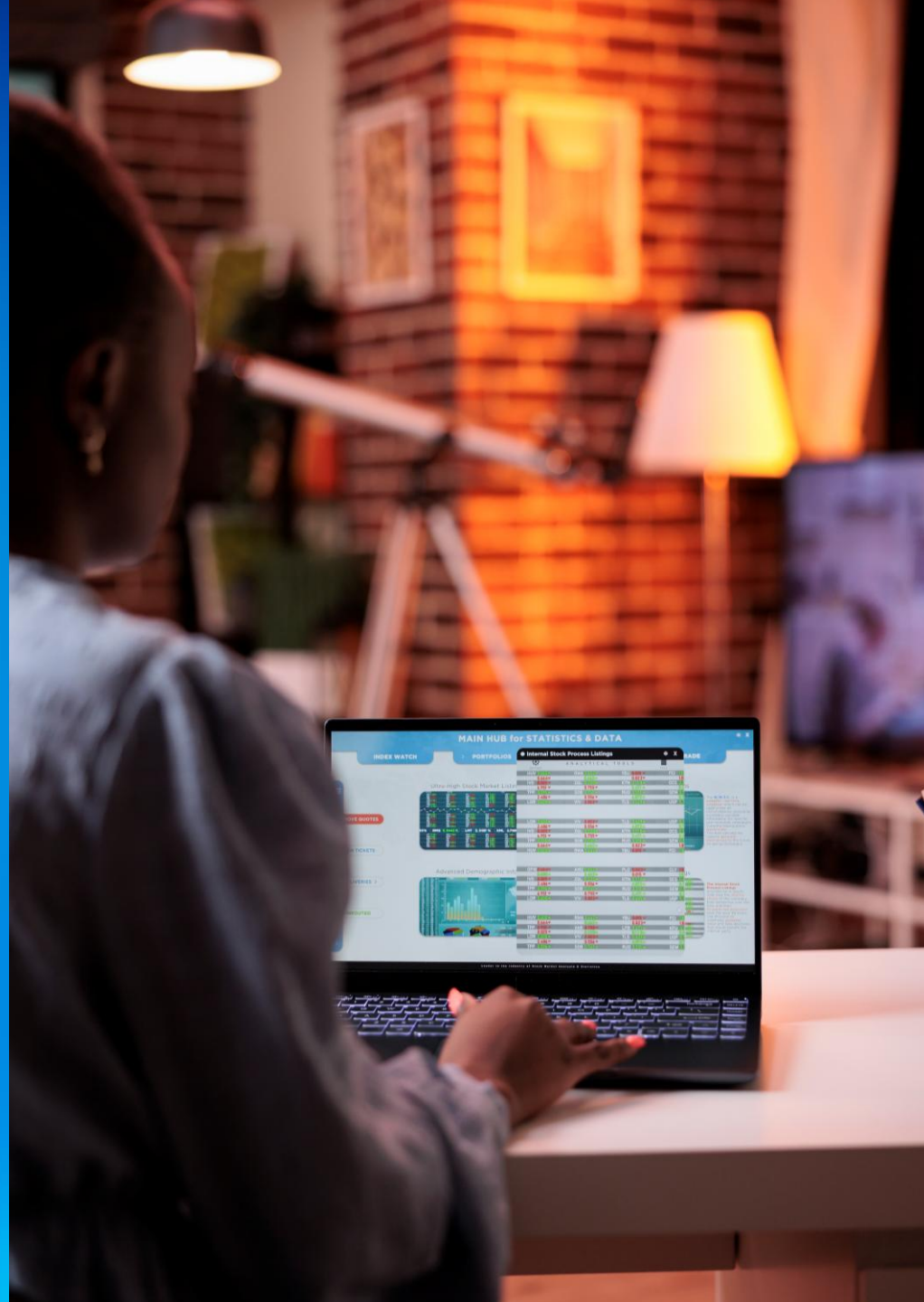
An authoritative transition blueprint designed to bridge the gap between technical execution and business strategy, ensuring every local asset adopts enterprise-grade security and automated operational patterns.

Why a Modernisation Playbook

Standardized Guardrails
Eradicates configuration drift, guarantees container optimization, and ensures all migrated assets align directly with the target stack.

Enterprise Scalability
Replaces isolated, low-availability local environments with dynamically scalable container pods backed by Multi-AZ persistent relational engines.

Early Risk Integration
Shifts security validation left, embedding automated checks, secrets masking, and mandatory compliance validation into early stages of engineering.



Purpose & Strategic Value Pillars

Standardized Delivery

Translates cloud engineering mechanics directly into business goals, keeping metrics, tracking logs, and audit pathways transparent across all tiers.

Rigid Compliance

Hardens the host border from Day 1, ensuring IAM authentications, certificate tracking, and secret synchronization pipelines are natively unified.

Operational Parity

Ensures immediate code execution parity across all development, testing, and staging ranges by standardizing multi-stage build pipelines.

Strategy and Code Assessment

Aligning client business objectives with technical feasibility to ensure the transition path supports growth, speed-to-market, and institutional security

What Are We Doing?

Performing deep-dive discovery sessions to define project requirements and assessing the legacy footprint to determine the most cost-effective and optimal transition model.

Roles

DevOps Engineer

Cloud Engineer

How It’s Done

Infrastructure as Code: Using Terraform templates to build identical environments reliably every time

Version Control: Leveraging Git as the "Time Machine" for all system and code changes to ensure absolute version control

Continuous Assembly & Auditability: Testing every developer contribution for security and bugs before it reaches users, ensuring every change is recorded, reviewed, and reversible for audits.

Key Outputs

- Terraform IaC templates
- GitOps automated assembly line
- Reversible audit trails

Application Preparation, Modernization and Migration

Deconstructing application code structures and translating data persistence engines to support stateless, horizontal scaling within containerized cluster resources.

What Are We Doing?

Decoupling existing application layers from underlying OS bindings and packaging code into high-security, minimal-footprint container images tailored for orchestration.

Roles

Application Developer

DevOps Engineer

Tools Needed

How It's Done

Multi-Stage Build Pattern: Separating build-stage dependencies from final runtime containers using minimized ECR-mirrored base slim images.

Environment Sanitization: Declaring environment-specific runtime configurations strictly outside of built container binaries.

Orchestrated Local Parity: Utilizing Docker Compose during early development phases to replicate multi-service production network.

Key Outputs

- Optimized Multi-Stage Dockerfile
- Local compose configurations
- Minimized runtime container

What Are We Doing?

Migrating localized, file-based data stores (such as SQLite) to managed AWS RDS instances configured for high availability.

Roles

Data Engineer

Database Administration (DBA)

Tools Needed

How It's Done

Zero-Static-Password Mandate: Decommissioning static database connection strings in favour of short-lived AWS IAM database tokens generated on runtime.

High-Availability Configurations: Deploying database engines inside Multi-AZ configurations across physical availability boundaries.

Automated Protection: Enforcing institutional 30-day automated backup retainment parameters and point-in-time recovery points.

Key Outputs

- Data schema refactoring scripts
- AWS IAM DB connection policies
- Managed RDS production instances

CI/CD Pipeline and Deployment

Standardizing the continuous verification of codebase assemblies and mapping automated delivery routes using verified pipeline configurations.

What Are We Doing?

Implementing a modern delivery pipeline with GitOps integration, ensuring that the cloud environment automatically syncs to match the approved state in the Git repository

Roles

Business Domain Owners

Technology Analysts

How It’s Done

Constraint & Persona Analysis: Analyzing legacy infrastructure constraints while identifying primary user personas and load profiles.

Migration Path Selection: Choosing between (Lift & Shift, Re-platform, or Refactor based on ROI and scalability.

Key Outputs

- Terraform IaC templates
- GitOps automated assembly line
- Reversible audit trails

What Are We Doing?

Automating structural validation, vulnerability scanning, and secure build compilation workflows directly from corporate code repository platforms.

Roles

DevOps Engineer

Cloud Engineer

How It’s Done

Continuous Quality Verification: Executing unit test frameworks and static security analysis upon every codebase change.

Secure Compilation: Binding built container artifacts with dynamic, cryptographically verifiable Git SHA tracking variables.

Key Outputs

- GitHub Action pipeline definitions
- Automated security scan reports
- Version-controlled ECR artifacts

What Are We Doing?

Enforcing standardized delivery runbooks to deploy finalized container architectures into Kubernetes hosting runtimes securely.

Roles

DevOps Engineer

Cloud Engineer

How It’s Done

Policy-Driven Execution: Targeting Kubernetes clusters using Continuous Deployment (Octopus or Jenkins) worker pools assigned with explicit corporate execution contexts.

Pre-Deployment Database Migrations: Triggering automated hooks inside container environments to apply data schemas before routing live transactions.

API Normalization: Integrating Ingress controllers and proxy routing rules (like path-stripping layers) to align incoming traffic with containers.

Key Outputs

- Project release runbooks
- Raw Kubernetes manifest setups
- Active application target nodes

Multi-Environment Configuration and Certificates

Separating developer, testing, and production stages through network limits and securing endpoints.

METRIC / RESOURCE	DEVELOPMENT (DEV)	SYSTEM INTEGRATION TESTING (SIT)	USER ACCEPTANCE TESTING (UAT)
Cluster Allocation	Non-Prod Target A	Non-Prod Target B	Non-Prod Target C
Database Endpoint	Dev-Scoped Managed DB	SIT-Scoped Managed DB	UAT-Scoped Managed DB
Network Control Range	Internal Subnet Boundary	Dedicated Dev/QA Bridged Zone	Dedicated CIDR-restricted Range

Note: Firewall settings and network routing boundaries must be formally requested and authorized independently for each staging environment to prevent cross-tier transaction bleeding.

Resilience, Scale and Disaster Recovery

Building systems capable of surviving hardware failures with zero user impact and defining recovery targets based on criticality.

What Are We Doing?

Ensuring the application dynamically grows or shrinks based on transaction volume while maintaining acceptable institutional Recovery Time Objective (RTO) target.

Roles

Cloud Engineer

Infrastructure Engineers

How It’s Done

Fault Tolerance: Deploying across Multi-AZ (Availability Zones) so applications continue running automatically if an AWS data center fails.

Scalability: Utilizing Horizontal Pod Autoscaling (HPA) to add "clones" of the app during peak hours and delete them at night to optimize cost.

Tiered Disaster Recovery: Defining Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO) using strategies like Backup & Restore, Pilot Light, or Multi-Site

Key Outputs

- Project release runbooks
- Raw Kubernetes manifest setups
- Active application target nodes

Security, Access and Governance

Enforcing zero-trust configuration architectures across active pipelines, federating credentials via active directories, and satisfying formal change ticketing gates.

What Are We Doing?

Establishing an end-to-end defence-in-depth model that prevents plaintext database credentials, API keys, and administrative configurations from exposure.

Roles

Cloud Engineer

DevSecOps Engineer

How It’s Done

Development Boundary Rules: Enforcing isolated configurations inside developers' environments and blacklisting them globally via gitignore patterns.

Build Automation Isolation: Leveraging GitHub Secrets validation variables to compile dynamic, temporary deployment tokens.

Unified Cloud Storage: Anchoring critical production keys directly inside AWS Secrets Manager, mapped to Octopus dynamically at container runtime.

Environment-scoped sensitive variables to prevent exposed values.

Key Outputs

- AWS Secrets Manager mappings
- Git ignore exclusions
- Encrypted runtime configurations


Thank You