DIVESCALE CASE STUDY · 2026

PROJECT 01

AWS INFRASTRUCTURE & CLOUD MIGRATION

FinTech Serverless Migration

Migrated transaction sync infrastructure from EC2 to serverless. Lambda, SQS, and Step Functions now handle millions of financial transactions with zero silent data loss.

AWS Lambda | SQS | Step Functions | FinTech

Client: US FinTech Startup

THE PROBLEM

EC2 Infrastructure Was a Liability

A US-based FinTech startup was processing transaction syncs on EC2 instances. Patching, scaling, and monitoring consumed engineering hours. When a server crashed, transactions were silently lost. When traffic spiked, the system choked.

The integration with Yodlee for account aggregation was brittle. Network timeouts killed sync jobs. Failed transactions had no retry mechanism. Staff manually reconciled discrepancies at month-end.

9.6 hrs

ANNUAL DOWNTIME

EC2 patching windows, OS crashes, and capacity limits caused predictable outages.

Silent

DATA LOSS

Failed transactions were dropped without a dead-letter queue. No one knew what was missing.

5-8 hrs

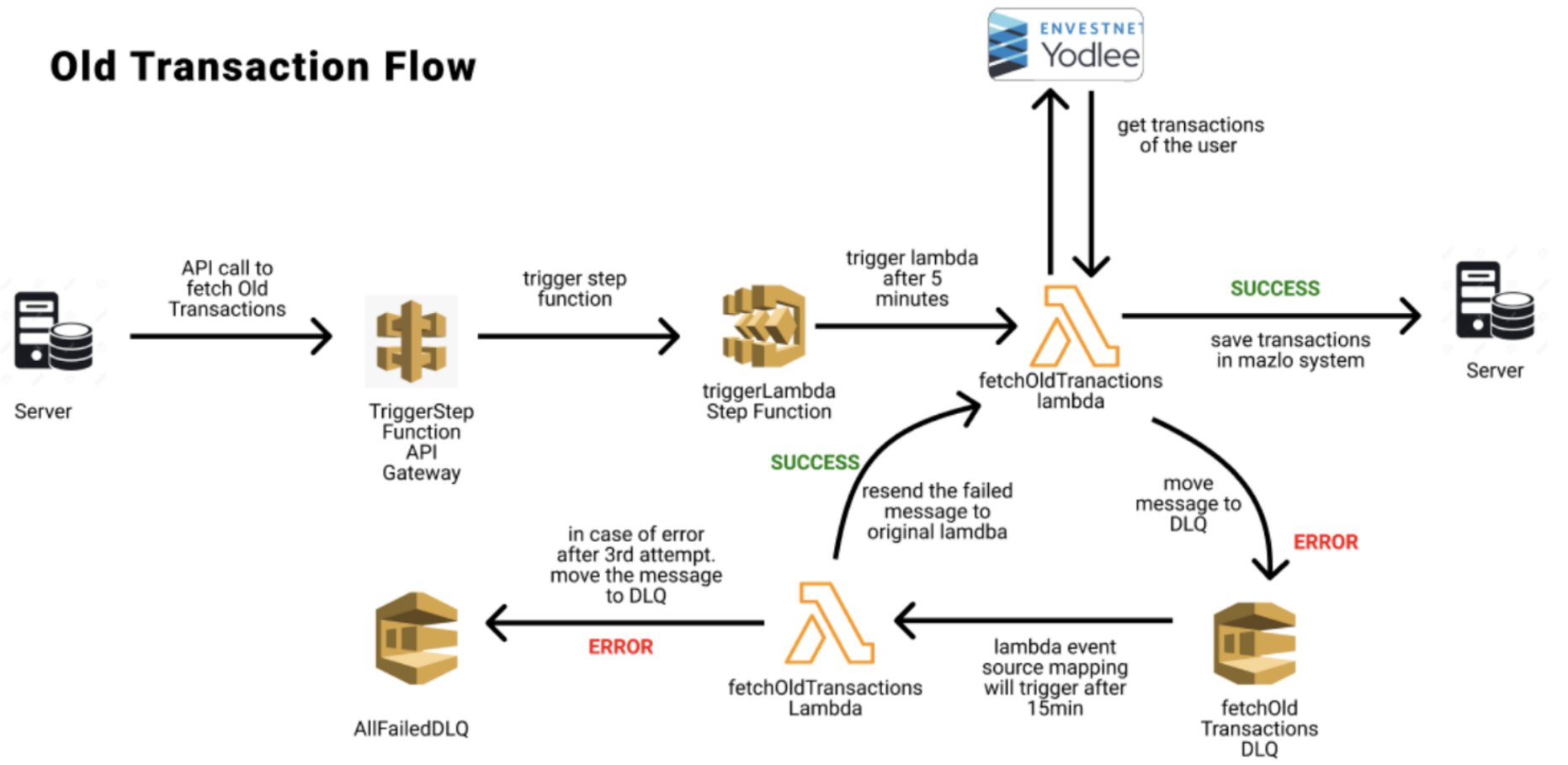
WEEKLY OPS TAX

Engineers spent time on patching, monitoring, and deployment coordination instead of product work.

ARCHITECTURE COMPARISON

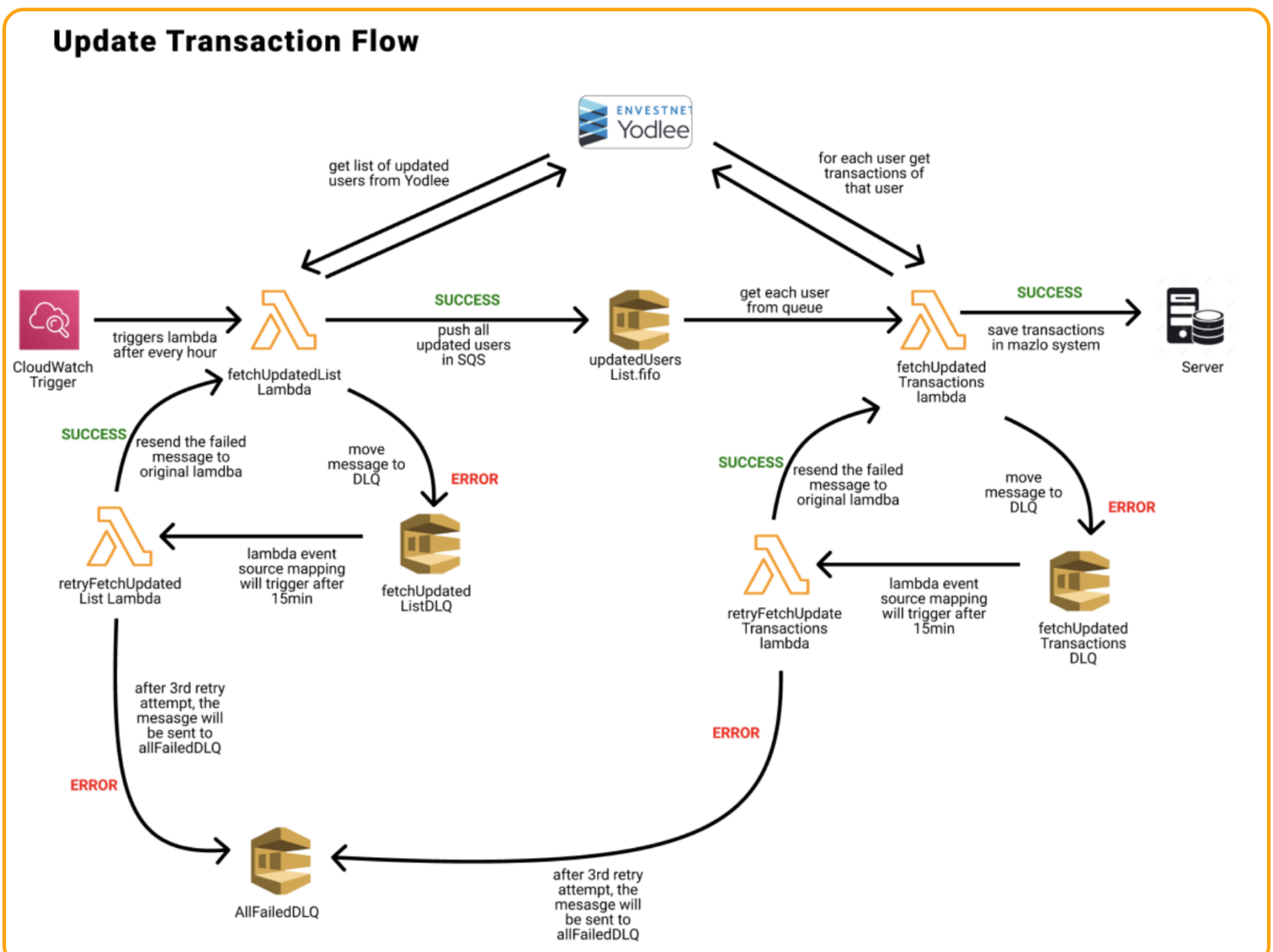
From Fragile to Fault-Tolerant

BEFORE — EC2 MONOLITH



- Single point of failure
- No automatic retry on failure
- Manual scaling required

AFTER — SERVERLESS



- Auto-scaling from 0 to thousands
- Dead-letter queue captures all failures
- Step Functions auto-retry with backoff

THE SOLUTION

Serverless Transaction Sync

Divescale designed and implemented a fully serverless architecture. Yodlee account aggregation feeds into Lambda functions. SQS queues buffer transactions. Step Functions orchestrate retries. API Gateway exposes endpoints. Nothing is provisioned. Nothing is patched.

Lambda Compute

Transaction processing functions run on demand. No idle servers. No capacity planning. Scale is automatic.

SQS + DLQ

Every transaction enters a queue. If processing fails, it moves to a dead-letter queue. Zero silent data loss.

Step Functions

Orchestrates multi-step sync workflows with built-in retry logic, exponential backoff, and error handling.

API Gateway + CloudWatch

Managed endpoints with 99.99% SLA. CloudWatch triggers periodic sync jobs and surfaces anomalies instantly.

CHAPTER 03 · RELIABILITY

Built for Financial-Grade Uptime

In FinTech, downtime is not an inconvenience. It is a compliance risk and a revenue loss. The serverless architecture delivers measurable reliability improvements.

SLA COMPARISON

COMPUTE	QUEUE	ORCHESTRATION
99.95%	**99.99%**	**99.99%**
Lambda vs 99.9% EC2	SQS guarantees delivery	Step Functions managed

Dead-Letter Queue

Failed transactions are captured, not dropped. Engineers can inspect and replay without data loss.

Auto-Retry with Backoff

Step Functions handle Yodlee API timeouts automatically. No custom retry code needed.

Fault Isolation

One failed Lambda does not crash the system. Each transaction is an independent execution.

Zero-Downtime Deployments

Lambda deployments are blue/green by default. No maintenance windows. No sync interruptions.

Infrastructure Cost Reduction

Moving from provisioned EC2 to pay-per-use serverless eliminated idle compute waste. The startup only pays for transactions actually processed.

AWS INFRA SAVINGS

$6K-25K

Per year vs EC2 + RDS provisioned instances

ENGINEERING TIME

$23K-37K

Per year reclaimed (5-8 hrs/week at $88/hr)

BUG PREVENTION

$10K-50K

Per year avoiding reconciliation and compliance issues

TOTAL ANNUAL VALUE

CATEGORY	ANNUAL VALUE
Engineering time saved (5-8 hrs/wk × $88/hr)	**$23K-$37K**
AWS infra cost reduction (serverless vs EC2)	**$6K-$25K**
Bug / incident prevention (1-2 failures/mo avoided)	**$10K-$50K**
Reliability improvement (fewer outages, zero silent data loss)	**$5K-$20K**
Total estimated value	**~$44K-$132K**

Bugs That No Longer Happen

The old architecture had predictable failure modes. The new architecture prevents them by design. Here is what used to break, and what now protects against it.

BEFORE

Failed Transaction Silently Dropped

No DLQ meant failed syncs disappeared. Reconciliation errors surfaced only at month-end.

AFTER

DLQ captures every failed message. Engineers inspect and replay. Zero silent data loss.

BEFORE

Yodlee API Timeout Kills Sync

A single network timeout would crash the entire sync batch. No retry mechanism existed.

AFTER

Step Functions retry with exponential backoff. SQS preserves the message. Sync continues.

BEFORE

EC2 Out of Disk / Memory

Full sync outage requiring manual restart. Capacity limits hit during traffic spikes.

AFTER

Lambda auto-scales. No server to fill up. Traffic spikes are handled transparently.

BEFORE

Traffic Spike Overloads Server

Requests dropped during high-volume periods. Transactions were lost before they could be processed.

AFTER

Lambda scales to thousands of concurrent executions. The queue absorbs spikes without dropping data.

AWS Services & Stack

The architecture is entirely serverless. Node.js Lambda functions handle business logic. PostgreSQL persists transaction state. NGINX serves as reverse proxy where needed.



Lambda
Serverless Compute



SQS
Message Queue



API Gateway
Managed Endpoints



CloudWatch
Monitoring & Logs



Node.js
Runtime



PostgreSQL
Database



NGINX
Reverse Proxy

RESULT

From provisioned infrastructure to serverless resilience.

The FinTech startup no longer worries about servers. Transactions sync automatically. Failures are captured, not dropped. Costs scale with usage, not with uptime. The engineering team focuses on product, not on patching.

$44K-132K	99.89%	65%
ANNUAL VALUE	UPTIME	COST REDUCTION

DIVESCALE CASE STUDY · 2026

PROJECT 02

AWS MICROSERVICES ARCHITECTURE

MASAF Restaurant Chain

Serverless microservices for a KSA restaurant chain. Automated Revel POS sync, QuickBooks accounting, TAP invoices, WhatsApp delivery, and SMS notifications.

AWS Lambda | Revel Systems | QuickBooks | WhatsApp API

Client: KSA Restaurant Chain

THE PROBLEM

Manual Operations at Scale

A multi-location restaurant chain in Saudi Arabia was running on manual workflows. POS sales from Revel were exported by hand into QuickBooks. Invoices were created manually for TAP payments. Customer notifications were sent one by one via WhatsApp. Delivery orders were managed through phone calls.

The staff spent 30-50 hours per week on operations that could be automated. Errors were common. Revenue leakage went unnoticed. Customer satisfaction suffered from delayed responses.

30-50 hrs

WEEKLY MANUAL WORK

Across accounting, invoicing, SMS, and delivery coordination.

2-5%

REVENUE LEAKAGE

From unsynced sales, missed orders, and invoice errors.

132K+

KSA RESTAURANTS

Competing in a SAR 42.5B market. Manual ops are a liability.

THE SOLUTION

Two Microservice Architectures

Divescale built two distinct serverless systems for the restaurant chain. One handles accounting and payment sync. The other manages delivery and customer communication. Both run on AWS Lambda without servers to manage.

Revel Sync Microservice

Syncs orders and payments from Revel POS to QuickBooks. Creates TAP invoices. Sends SMS via Unifonic.

Lambda | RDS | API Gateway

Yallow Delivery Microservice

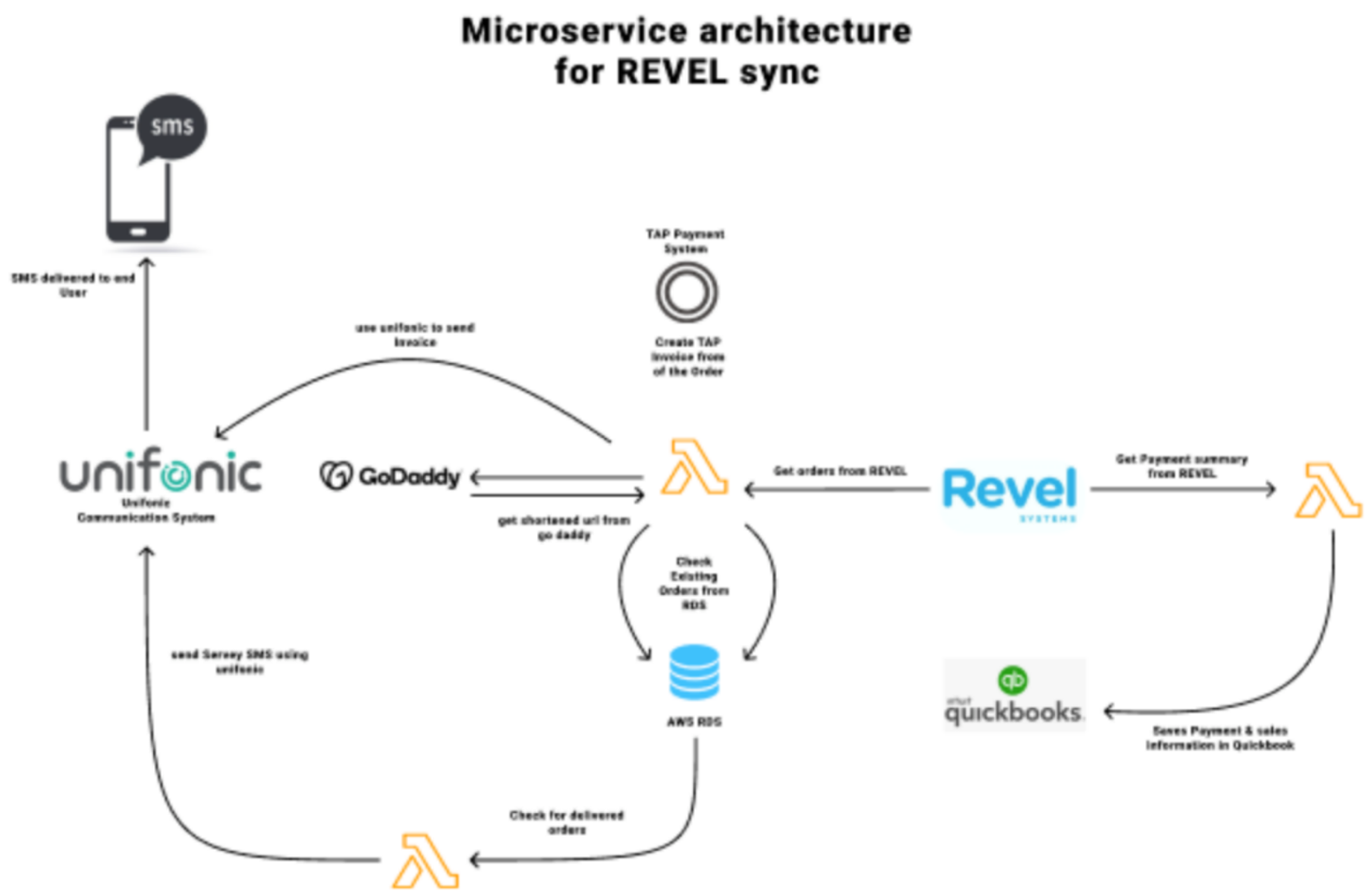
Integrates with Yallow delivery management via WhatsApp Business API. Handles order updates and location sharing.

Lambda | WhatsApp API | RDS

ARCHITECTURE 01 · REVEL SYNC

POS to Accounting Pipeline

This architecture synchronizes order and payment data from Revel Systems into QuickBooks, creates TAP invoices, and sends customer notifications via SMS.



BEFORE

Staff exported Revel sales manually. QuickBooks entries were keyed by hand. Invoice creation took 30 minutes per batch.

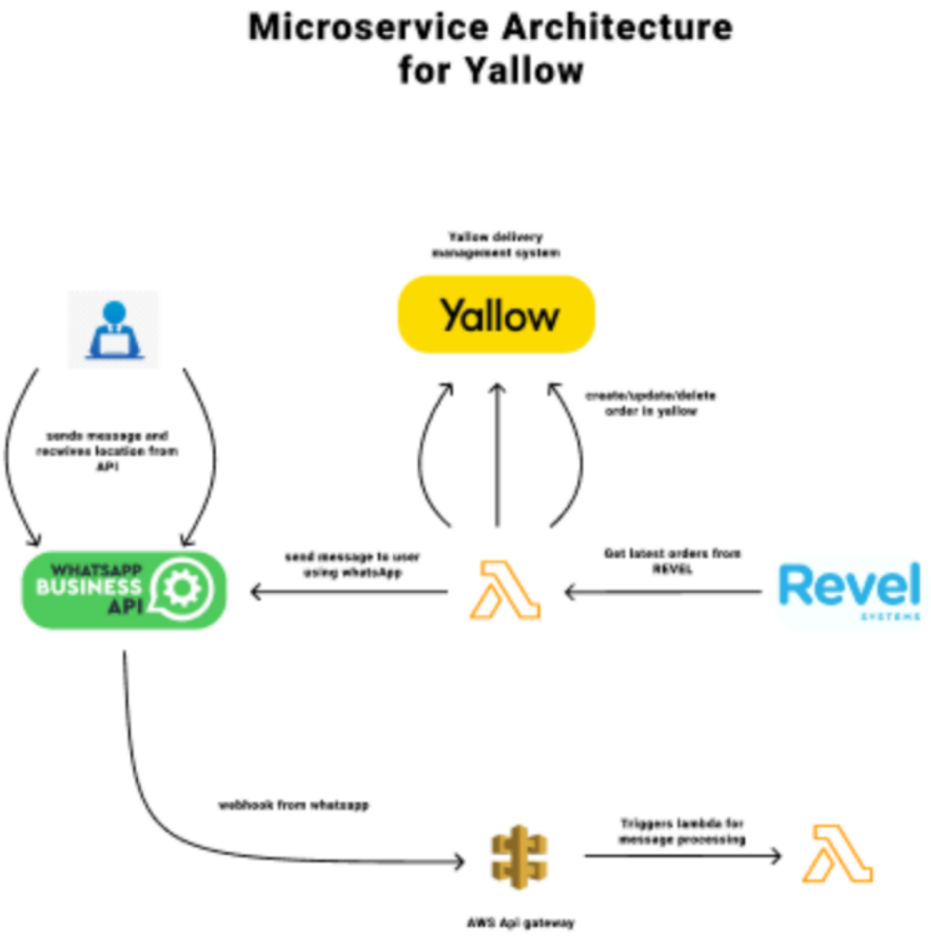
AFTER

Lambda auto-syncs every X minutes. TAP invoices created programmatically. Unifonic SMS triggers on payment confirmation.

ARCHITECTURE 02 · YALLOW DELIVERY

WhatsApp-Driven Delivery

This architecture integrates with Yallow delivery management using the WhatsApp Business API. Customers receive order updates, share locations, and communicate with drivers via WhatsApp.



BEFORE

Delivery orders managed via phone calls. Customer updates required manual texting. Address verification was error-prone.

AFTER

WhatsApp webhook captures orders. Lambda processes messages. Structured location data feeds Yallow directly. No manual handoff.

CHAPTER 03 · RELIABILITY

Zero Manual Handoffs

Every step that used to require a person now runs automatically. Lambda functions handle the workflow. RDS prevents duplicate processing. API Gateway manages webhooks reliably.

No Missed Revel Syncs

Lambda runs on schedule. If it fails, it retries. RDS tracks last sync timestamp so nothing is processed twice.

No Lost Invoices

TAP webhook confirms payment before the flow continues. Failed payments are flagged, not silently ignored.

No Dropped WhatsApp Messages

API Gateway receives every webhook. Lambda processes reliably. 22M+ WhatsApp users in KSA are reachable instantly.

Complete Audit Trail

Every order flows through: Revel → Lambda → RDS → TAP → Unifonic. Disputes are winnable. Chargebacks are preventable.

Annual Value Summary

The serverless microservices architecture delivers recurring value across staff time, infrastructure cost, and revenue protection.

STAFF TIME SAVED

$25K-75K

30-50 hrs/week reclaimed across operations and engineering

REVENUE PROTECTED

$15K-50K

From unsynced sales and missed orders prevented

AWS SAVINGS

$5K-18K

Serverless vs traditional EC2 instances

TOTAL ANNUAL VALUE (KSA RESTAURANT CHAIN)

CATEGORY	ANNUAL VALUE
Staff time saved (30-50 hrs/wk ops + engineering)	**$25K-$75K**
AWS infra cost savings (serverless vs traditional)	**$5K-$18K**
Revenue leakage prevention (unsynced sales, missed orders)	**$15K-$50K**
Invoice / payment error reduction	**$3K-$10K**
Chargeback & dispute prevention (audit trail)	**$3K-$8K**
Total estimated value	**~$51K-$161K**

Errors That Disappeared

Manual restaurant operations have predictable failure modes. The microservices architecture prevents them systematically.

BEFORE

POS Sales Not in QuickBooks

Staff forgot to export/import. Month-end reports were delayed. Revenue was understated.

AFTER

Lambda auto-syncs every X minutes. RDS deduplication prevents double-entry. Reports are always current.

BEFORE

Invoice Sent with Wrong Amount

Manual data entry errors created customer disputes and delayed payments.

AFTER

TAP invoice created programmatically from order data. Amount is always accurate. Payment is verified.

BEFORE

Customer Not Notified

Staff forgot to send SMS. Customers called to check order status. Support load was high.

AFTER

Unifonic SMS auto-triggered on payment confirmation. Survey follows automatically after delivery.

BEFORE

Lost WhatsApp Order

Messages buried in chat. Orders missed. Customers frustrated. Revenue lost.

AFTER

API Gateway webhook captures every message. Lambda processes into RDS. No order slips through cracks.

AWS Services & Integrations

The architecture runs entirely on AWS serverless. Python Lambda functions integrate with Revel, QuickBooks, TAP, Unifonic, and WhatsApp Business API.

Lambda
Serverless Compute

RDS
Managed MySQL

API Gateway
Webhook Endpoints

Python
Runtime

QuickBooks
Accounting

WhatsApp API
Customer Communication

RESULT

A KSA restaurant chain automated what used to consume 50 hours a week.

Sales flow into QuickBooks automatically. Invoices create themselves. Customers get SMS updates without staff lifting a finger. Delivery orders route through WhatsApp. The architecture pays for itself in months.

$51K-161K	30-50 hrs	2 Microservices
ANNUAL VALUE	WEEKLY SAVINGS	SERVERLESS SYSTEMS

MASAF Restaurant Chain · Divescale · 2026

DIVESCALE CASE STUDY · 2026

PROJECT 03

AWS SERVERLESS MICROSERVICES

Cheflivery

Automated food delivery pipeline from order to payment to notification. Typeform intake, Lambda processing, TAP payment, and SMS confirmation -- all serverless.

AWS Lambda | API Gateway | RDS | Food Delivery

Client: Cheflivery Food Delivery

THE PROBLEM

Manual Orders, Manual Errors

A food delivery chain was processing orders through phone calls and emails. Staff took orders by hand, typed them into spreadsheets, created invoices manually, and called customers to confirm payment. Every step was a point of failure.

Order errors cost 2-5% of revenue annually. Chargebacks drained profit. Customers were frustrated by delays and mistakes. The business was growing, but the operations were not.

2-5%

REVENUE LOST TO ERRORS

Wrong items, wrong addresses, missed orders. Manual entry has predictable error rates.

5-8 min

PER ORDER PROCESSING

Phone calls, manual entry, invoice creation, payment checking, and customer notification.

$15-100

PER CHARGEBACK

Without audit trails, disputes were unwinnable. Repeat offenders faced additional fines.

THE SOLUTION

Automated Order Pipeline

Divescale built a serverless microservice that handles the entire order lifecycle. The customer fills a Typeform. A Lambda function receives the webhook. The order is stored in RDS. A TAP invoice is created. Payment is confirmed via webhook. A Zapier workflow sends an SMS. The customer is notified. No human touches the order in between.

Typeform Intake

Customers enter their own order details. Structured data. No misheard phone orders. No typos from staff.

Lambda Processing

Webhook triggers Lambda. Order is validated, stored in RDS, and passed to the payment step. Auto-retry on failure.

TAP Payment

Invoice created programmatically. Payment webhook confirms before fulfillment. Idempotent. No double-charges.

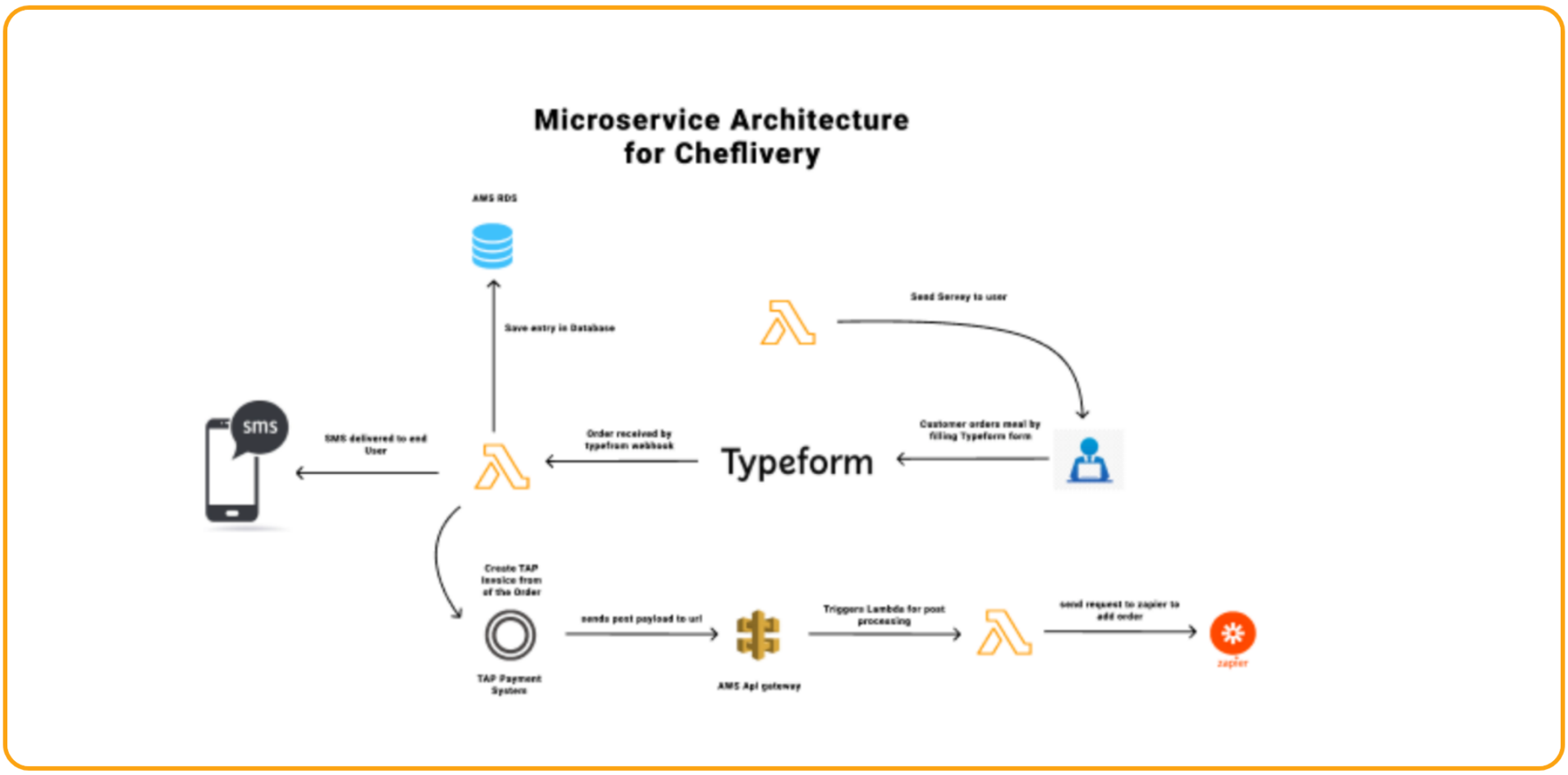
Zapier + SMS

Payment confirmation triggers SMS automatically. Customer knows immediately. Staff does nothing.

ARCHITECTURE OVERVIEW

Order-to-Notification Pipeline

The architecture is a linear pipeline. Each stage is a Lambda function triggered by the previous. RDS persists state. API Gateway exposes webhooks. No server is provisioned.



STEP 1	STEP 2	STEP 3	STEP 4	STEP 5
Typeform	**Lambda**	**RDS**	**TAP**	**SMS**
Customer order	Process webhook	Store order	Payment	Notification

Built for 99.9% Uptime

A food delivery business cannot afford downtime during lunch rush. The serverless architecture guarantees availability without manual intervention.

SERVICE-LEVEL GUARANTEES

API GATEWAY	LAMBDA	RDS
99.99%	**99.95%**	**99.95%**
Managed endpoints	Auto-scaling compute	Multi-AZ database

No Server Crash Takes Orders Offline

Lambda + API Gateway runs on AWS infrastructure. Lunch rush is handled transparently. No capacity planning needed.

Payment Confirmation Never Lost

TAP webhook + API Gateway ensures payment confirmation is received. Lambda retries on transient failures. No manual checking.

RDS Multi-AZ Prevents Data Loss

Automated backups, patching, and failover. No database administrator required. Order history is always safe.

Lambda Auto-Retry on Failures

3 built-in retry attempts for every function. Zapier rate limits or SMS glitches are handled automatically.

Annual Value Summary

The serverless order pipeline saves staff time, reduces infrastructure cost, and prevents revenue loss from errors.

STAFF TIME SAVED

$15K-75K

4-20 hrs/day of order entry eliminated

ENGINEERING TIME

$23K-37K

5 hrs/week reclaimed on ops

AWS SAVINGS

$5K-18K

Serverless vs EC2 instances

TOTAL ANNUAL VALUE

CATEGORY	ANNUAL VALUE
Staff order entry time saved (4-20 hrs/day × $15-25/hr)	**$15K-$75K**
DevOps / engineering time saved (5 hrs/wk × $88/hr)	**$23K-$37K**
AWS infra cost reduction (serverless vs traditional)	**$5K-$18K**
Order errors & chargebacks prevented	**$5K-$20K**
Total estimated value	**~$48K-$150K**

Errors That Vanished

The pre-automation order flow had predictable failure modes. The serverless pipeline prevents them by design.

BEFORE

Wrong Order Items

Phone orders were misheard or mistyped. Customers received wrong meals. Remakes cost time and money.

AFTER

Customer enters directly in Typeform. Structured data. No transcription errors.

BEFORE

Missing Orders

Paper orders were lost. Emails were missed. Customers waited for food that was never prepared.

AFTER

Every Typeform submission triggers Lambda. Stored in RDS. No order slips through.

BEFORE

Payment Not Collected

Invoice forgotten. Payment not verified. Food delivered without confirmation. Revenue at risk.

AFTER

TAP invoice auto-created. Webhook confirms payment before fulfillment. No delivery without payment.

BEFORE

Chargebacks Without Proof

No documentation trail. Disputes were unwinnable. $15-100 per chargeback. Repeat offenders fined.

AFTER

Typeform response + payment webhook + Lambda logs = complete audit trail. Every dispute is winnable.

AWS Services & Stack

The architecture is entirely serverless. Python and JavaScript Lambda functions integrate with Typeform, TAP, Zapier, and SMS services.

Lambda
Serverless Compute

API Gateway
Webhook Endpoints

RDS
Managed MySQL

Python
Runtime

JavaScript
Runtime

MySQL
Database Engine

Zapier
Workflow Automation

RESULT

From phone orders to a fully automated serverless pipeline.

Cheflivery now processes orders without human intervention. Customers enter their own details. Payments are verified automatically. Notifications are sent instantly. The business scaled without adding operations staff.

$48K-150K	99.9%	0
ANNUAL VALUE	UPTIME	SERVERS MANAGED

DIVESCALE CASE STUDY · 2026

PROJECT 04

AWS CLOUD PLATFORM & AI INTEGRATION

Pet Health Care System

Multi-tenant SaaS platform for veterinary clinics with AI-powered chatbot, automated CI/CD, secure file storage, real-time billing, and credential rotation.

AWS Bedrock | ECS Fargate | AI Agent | Multi-Tenant

Client: Veterinary SaaS Platform

THE PROBLEM

Building a Veterinary SaaS from the Ground Up

We were building a multi-tenant veterinary SaaS platform from scratch. The goal was to serve hundreds of clinics with automated operations, not manual work. The platform needed to handle appointment scheduling, billing per clinic, secure file uploads, automated credential rotation, and AI chat support.

Deployments were manual and risky. Security was an afterthought. Cost allocation across tenants was impossible. The platform needed to scale, but the infrastructure was being built piece by piece without a unified architecture.

Hours

PER DEPLOYMENT

Manual builds, database migrations, and rollbacks consumed the entire team.

Unknown

PER-TENANT COST

No visibility into which clinic consumed what resources. Billing was guesswork.

Manual

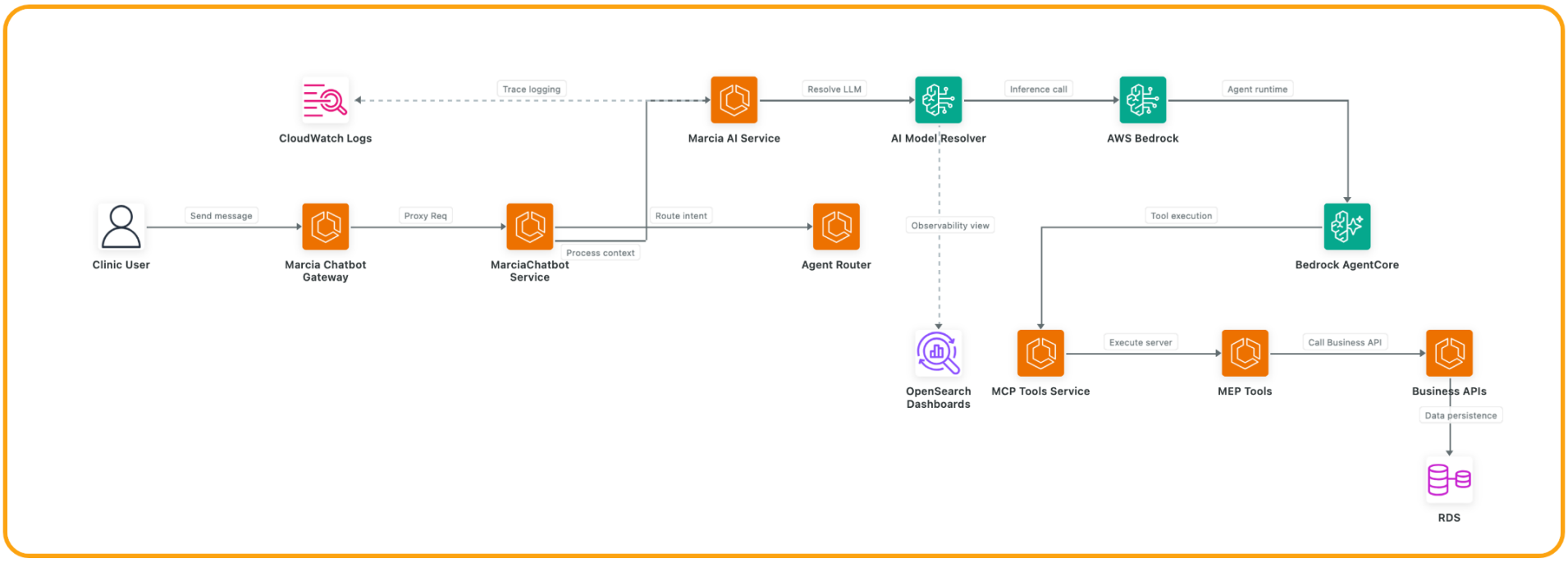
CREDENTIAL ROTATION

Database passwords and Redis keys were rotated by hand. Risk of exposure was high.

ARCHITECTURE 01 · AI PLATFORM

Marcia Bedrock Agentic AI

A clinic-facing AI chatbot built on AWS Bedrock. The agent routes veterinary queries, resolves LLM models, executes tools, and calls business APIs. All interactions are logged and observable.



BEFORE

Clinics called support for every question. Staff answered the same queries repeatedly. No self-service existed.

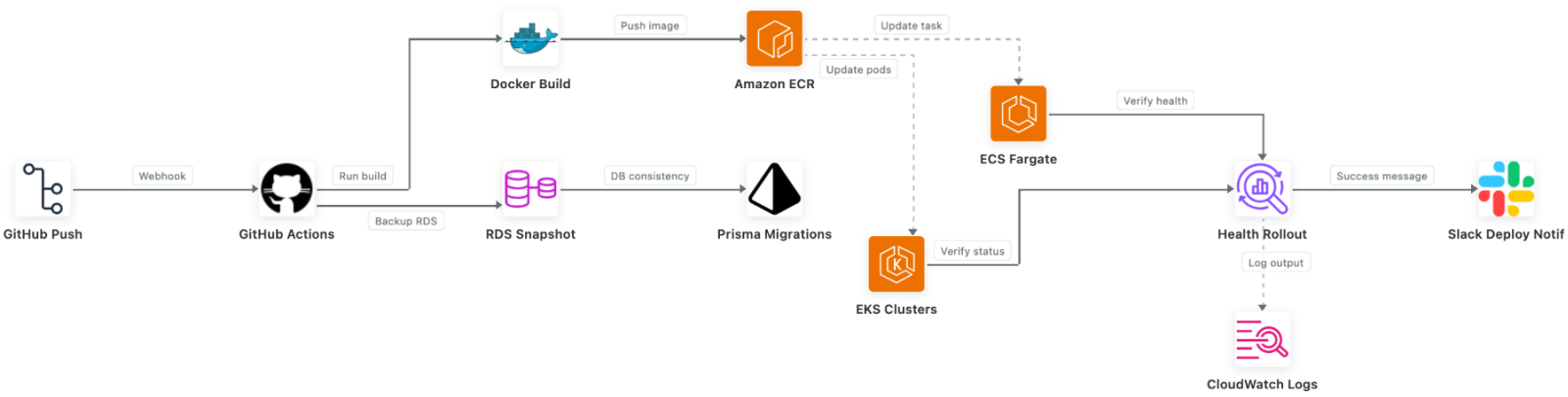
AFTER

Bedrock Agent handles queries 24/7. Intent routing, LLM resolution, and tool execution are fully automated. CloudWatch logs every interaction.

ARCHITECTURE 02 · CI/CD

Automated Deployment Pipeline

GitHub Actions trigger Docker builds, push to ECR, run Prisma migrations, update ECS and EKS clusters, verify health, and notify Slack. Zero-touch deployment from commit to production.



BEFORE

Deployments were manual and error-prone. Database migrations were skipped. Rollbacks took hours. No health checks.

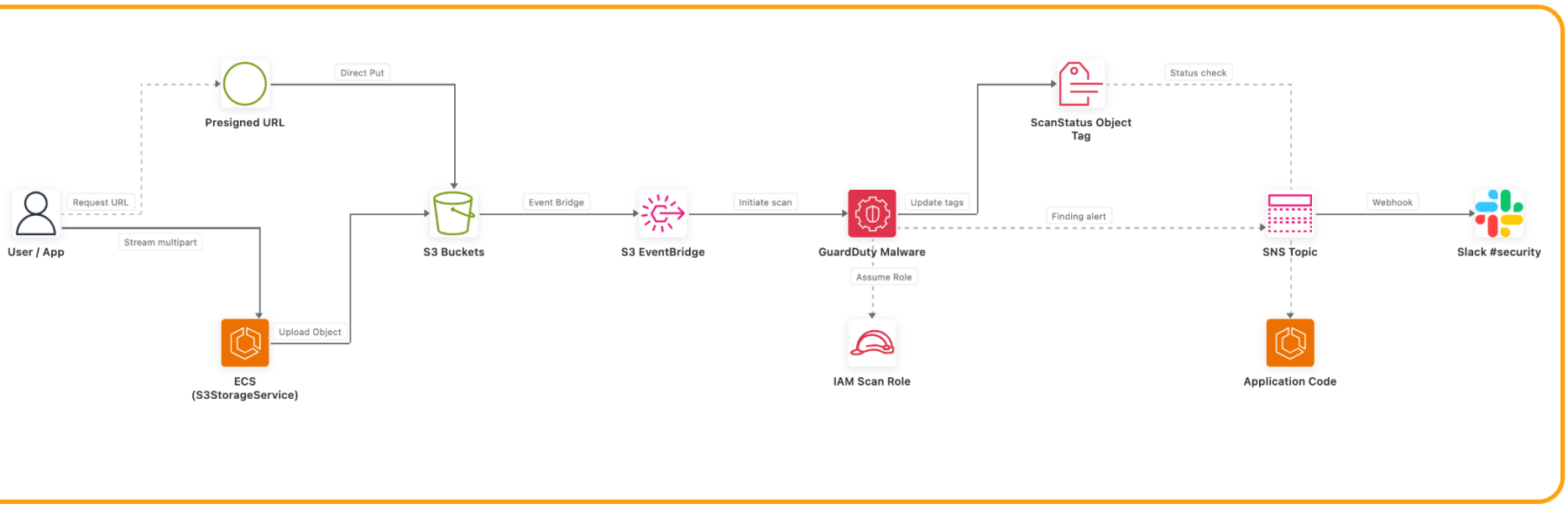
AFTER

Every push triggers a full pipeline: build, snapshot, migrate, deploy, verify, notify. Failed deploys are caught before they reach users.

ARCHITECTURE 03 · SECURITY

S3 File Upload & Malware Scanning

Every file uploaded by a clinic is scanned for malware before it is accepted. GuardDuty Malware Protection triggers on S3 events. Infected files are flagged. Clean files are tagged. Slack is notified.



BEFORE

Files uploaded to a simple storage bucket. No scanning. Malware could propagate across the platform. No audit trail.

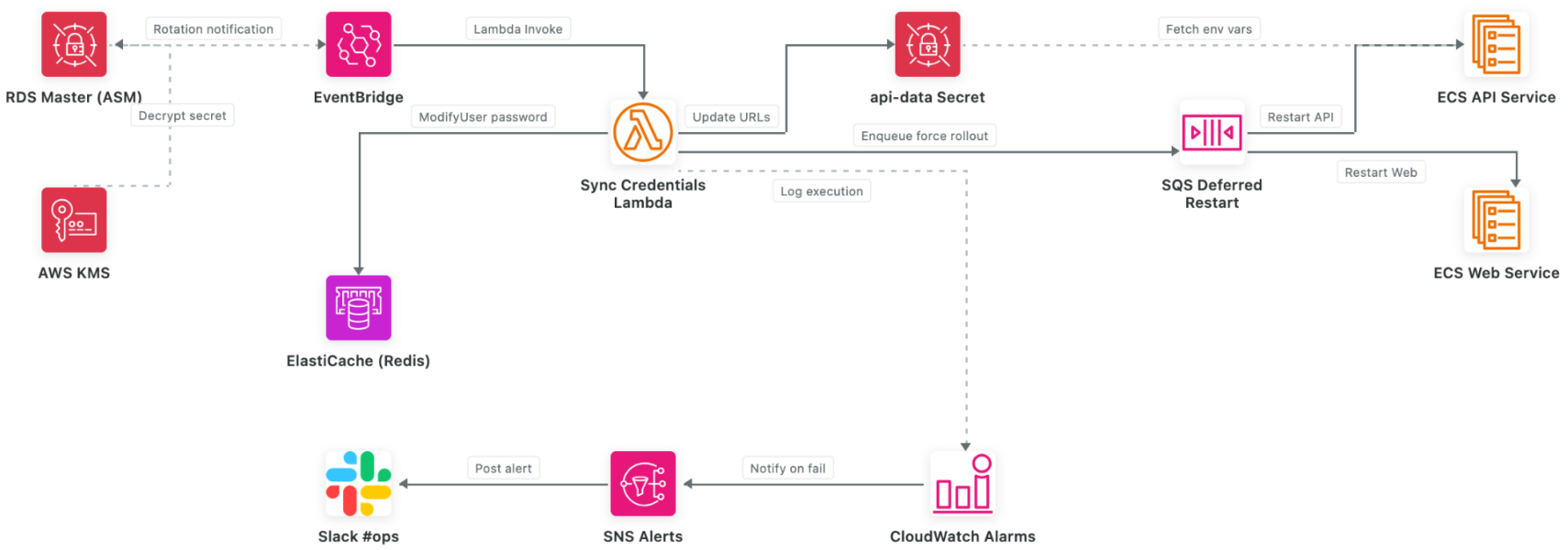
AFTER

GuardDuty scans every upload. EventBridge triggers the scan. SNS alerts Slack #security. Clean files get a ScanStatus tag. Malware never reaches the application.

ARCHITECTURE 04 · SECURITY

Database & Redis Credential Rotation

RDS master credentials and Redis keys are rotated automatically via AWS Secrets Manager. EventBridge triggers rotation. Lambda syncs credentials. SQS defers restarts. ECS services reload seamlessly. No manual intervention.



BEFORE

Credentials were rotated manually during maintenance windows. Downtime was required. Old passwords lingered in config files. Security audits failed.

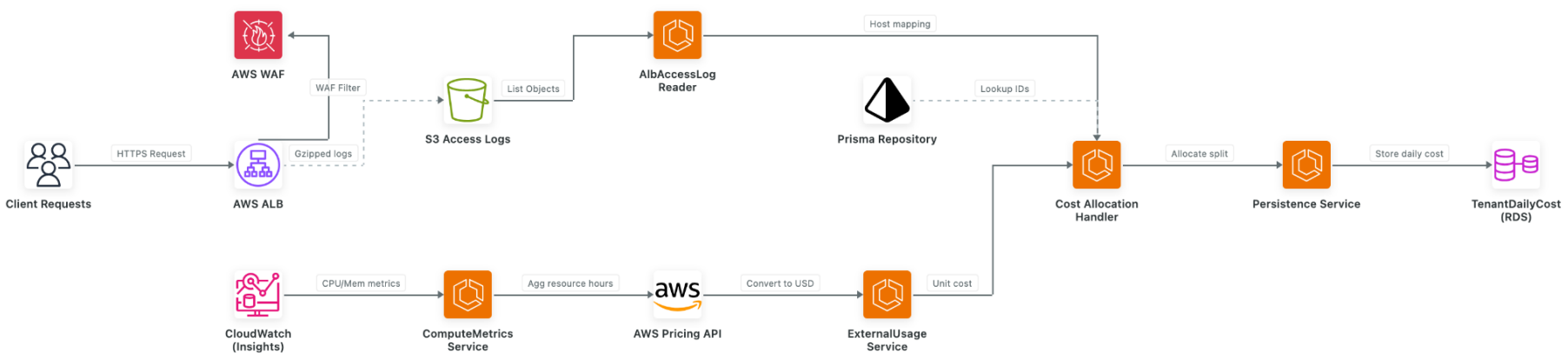
AFTER

Secrets Manager rotates automatically. Lambda syncs to ElastiCache. SQS defers ECS restarts. KMS encrypts at rest. CloudWatch alarms notify Slack on failure.

ARCHITECTURE 05 · BILLING

Per-Tenant AWS Usage Billing

ALB access logs and CloudWatch metrics are analyzed to attribute AWS costs to individual clinics. The Cost Allocation Handler maps hosts to tenants. Daily costs are persisted to RDS for accurate invoicing.



BEFORE

AWS costs were a single bill. No way to know which clinic used what. Billing was flat-rate guesswork. Margins were eroded by heavy users.

AFTER

ALB logs and CloudWatch metrics are processed daily. Prisma maps hosts to tenants. AWS Pricing API converts to USD. RDS stores daily costs per clinic. Usage-based billing is accurate.

THE SOLUTION

Integrated Cloud Platform

Divescale designed and built a unified AWS platform that connects AI chat, automated deployments, secure storage, credential management, and real-time billing. Each component is isolated, observable, and scalable. The platform is built on ECS Fargate with supporting services for every operational need.

Bedrock Agentic AI

Clinic users interact with Marcia, a Bedrock-powered agent. It routes intents, resolves LLMs, executes tools, and calls business APIs. CloudWatch traces every interaction.

CI/CD Pipeline

GitHub Actions automate builds, Docker pushes, RDS snapshots, Prisma migrations, ECS/EKS deploys, health checks, and Slack notifications. Zero manual deploys.

Security by Design

GuardDuty scans every file upload. KMS encrypts secrets. WAF filters traffic. Credentials rotate automatically. CloudWatch alarms surface anomalies instantly.

Usage-Based Billing

ALB logs and CloudWatch metrics are processed to allocate costs per tenant. AWS Pricing API converts to USD. Daily costs are stored in RDS for accurate invoicing.

Built for 99.95% Uptime

A veterinary SaaS platform serves hundreds of clinics during business hours. Downtime is not acceptable. The architecture guarantees availability across all critical paths.

SLA COMPARISON

ECS FARGATE	RDS MULTI-AZ	ALB
99.95%	99.99%	99.99%
Container orchestration	Automated failover	Managed load balancing

Health Rollout Checks

Every deployment is verified before traffic is shifted. Failed deploys are rolled back automatically. CloudWatch logs the reason.

RDS Snapshot Before Deploy

Every deployment creates an RDS snapshot first. If a migration fails, the database is restored. Zero data loss on bad deploys.

Auto-Restart on Credential Change

SQS deferred restart ensures ECS services reload gracefully when credentials rotate. No manual restart. No downtime.

CloudWatch Alarms to Slack

Every anomaly triggers an alarm. SNS routes to Slack #ops. The team knows before users do. Response time is measured in minutes.

AWS Services & Stack

The platform is built on ECS Fargate with a comprehensive AWS service suite. TypeScript and Python power the backend. Prisma manages the database layer.

Bedrock
AI / LLM

ECS Fargate
Container Compute

EKS
Kubernetes

ECR
Container Registry

RDS
Managed Database

S3
Object Storage

GuardDuty
Threat Detection

SNS
Notifications

CloudWatch
Monitoring & Logs

EventBridge
Event Bus

OpenSearch
Search & Analytics

ElastiCache
Redis Cache

KMS
Key Management

WAF
Web Firewall

ALB
Load Balancer

Docker
Containerization

GitHub Actions
CI/CD

Slack
Notifications

Prisma
ORM

Node.js
Runtime

Python
Runtime

RESULT

From manual operations to an integrated, automated cloud platform.

The veterinary SaaS platform now serves hundreds of clinics with AI chat, automated deployments, secure file storage, and usage-based billing. Engineering focuses on features, not on firefighting.

99.95%
UPTIME

5
INTEGRATED SYSTEMS