AWS CLOUD MODERNIZATION

Cloud-Native Booking Report

Orchestration with AWS Step Functions

How a leading gaming and hospitality chain retired its on-premises batch job framework and migrated end-to-end booking report orchestration to AWS Step Functions — achieving 99.2% execution success rate, 82% reduction in operations overhead, and 38% infrastructure cost reduction in 6 weeks.

82%

Ops Overhead Reduction

15–18 hrs/week → under 3 hrs/week

99.2%

Execution Success Rate

Across all Step Functions workstreams

38%

Infrastructure Cost Reduction

Fully-loaded annual TCO vs on-prem

Engagement Summary

A leading gaming and hospitality chain migrated its tightly coupled on-premises batch booking report orchestration framework to AWS Step Functions — replacing manual job monitoring, custom failure handling, and fixed-capacity infrastructure with a fully managed, serverless, event-driven orchestration platform. The on-premises batch framework was fully decommissioned following sequential workstream cutover. The engagement was completed in 6 weeks with zero report delivery disruption to business, finance, and operational stakeholders.

The Challenge

Customer Background

A leading gaming and hospitality chain operated a portfolio of properties requiring continuous, accurate reporting across booking, reservation, revenue, and operational data streams. Business, finance, and operational teams depended on timely, consolidated booking reports generated daily from multiple application databases and third-party reservation systems to drive occupancy management, rate decisions, and financial reconciliation.

Visibility Constraints

The on-premises framework provided limited visibility into job execution state. Operations teams had no real-time view of which steps were running, which had failed, or how far along the pipeline had progressed at any given moment. Diagnosing failures required manual log inspection across multiple systems, with no centralised execution history or state-level traceability. This opacity significantly extended mean time to resolution and made it impossible for business and operational stakeholders to have confidence in report delivery timelines during critical periods.

Scalability Constraints

Fixed on-premises infrastructure sized for average reporting load could not accommodate the surge in data volumes and report generation requests that characterised peak booking periods. During major events and holiday windows — the highest-value operational periods for a gaming and hospitality operator — the reporting infrastructure was under its greatest strain precisely when accurate, timely reporting was most critical to revenue management and operational decision-making.

Operational Constraints

The on-premises batch orchestration framework depended on tightly coupled job dependencies, manual monitoring, and custom-built failure handling logic that had accumulated significant technical debt over time. When individual jobs failed — due to data source unavailability, schema changes, or infrastructure issues — the absence of structured retry logic and automated recovery meant that operations teams had to manually diagnose the point of failure, assess downstream impact across dependent jobs, and restart affected processes by hand. During peak booking periods — major events, holidays, and high-occupancy windows — this manual intervention model was incompatible with the volume and velocity of reporting demand.

Technical Constraints

Tight coupling between batch jobs meant that a failure in any single extraction, transformation, or aggregation step could cascade across the entire reporting pipeline — leaving downstream jobs in an indeterminate state and requiring full or partial restarts that extended report delivery timelines. The absence of native branching, conditional logic, and independent step management made it structurally difficult to isolate failures, implement targeted retries, or route errors to appropriate handling paths without rewriting significant portions of the custom orchestration code.

The Core Challenge

A tightly coupled on-premises batch framework generating booking reports across multiple data sources — with no automated error recovery, no real-time execution visibility, and fixed infrastructure unable to scale during peak events. A serverless, cloud-native orchestration platform was required to deliver reliable, observable, and elastic report generation at any scale.

Why Amazon Web Services?

AWS Step Functions was selected as the orchestration platform based on four strategic criteria that directly addressed the structural limitations of the legacy batch framework:

Native Error Handling	Full Execution Visibility	Elastic Serverless Scale	Parallel Execution
Built-in retry policies, exponential backoff, and Catch states replace thousands of lines of custom failure handling code in the legacy framework.	State-level execution history, real-time status tracking, and CloudWatch Logs Insights replace manual log inspection across disparate on-premises systems.	Lambda functions and Step Functions scale automatically with workload demand — no pre-provisioning, no manual capacity management during peak events.	Step Functions parallel branches execute independent report workstreams simultaneously — eliminating the sequential bottlenecks inherent in the legacy batch architecture.

The Solution

Migration Methodology

The AWS Partner designed and delivered a cloud-native serverless booking report orchestration platform using a structured four-phase methodology — replacing the legacy batch framework with production-ready AWS Step Functions state machines within a six-week engagement timeline, with zero disruption to live reporting operations throughout.

Phase 1	Phase 2	Phase 3	Phase 4
Assessment	**Build**	**Parallel Run**	**Cutover**
Workflow mapping, dependency graph, data source inventory	State machine design, Lambda development, RDS & API integration	Validation against live data, peak period coverage, parity testing	Sequential workstream cutover, batch job retirement, decommission

Phase Detail

Phase 1 — Assessment

A comprehensive inventory of the existing on-premises batch job framework was conducted —

Phase 3 — Parallel Run

Each Step Functions state machine was validated against live production data volumes before

mapping all extraction jobs, transformation dependencies, aggregation steps, report generation processes, and downstream delivery targets across business, finance, and operational reporting workstreams. Each job dependency chain was documented as a directed workflow graph, forming the blueprint for Step Functions state machine design. Data source connectivity requirements — spanning Amazon RDS databases and third-party booking and reservation system APIs — were catalogued alongside failure modes, retry requirements, and report delivery SLAs for each workstream.

Phase 2 — Build

Documented workflow graphs were translated into AWS Step Functions state machines — one per major reporting workstream — with each state machine defining the complete sequence of extraction, validation, transformation, aggregation, and delivery steps as independently managed states. AWS Lambda functions were developed for each discrete processing step, with dedicated functions per third-party API data source ensuring independent observability, retryability, and deployability. Amazon RDS connections were managed through VPC-bound Lambda execution with least-privilege IAM policies.

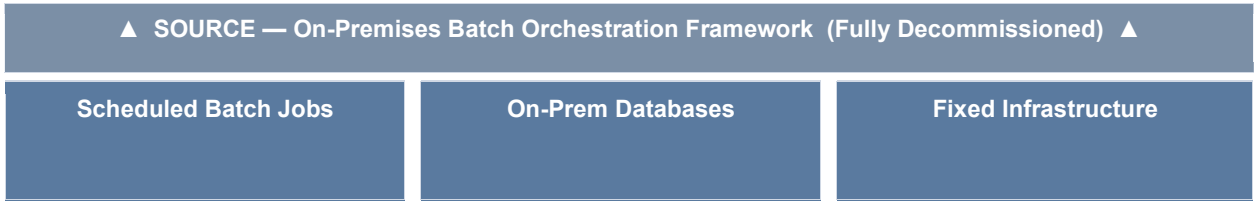
legacy batch jobs were retired — confirming data integrity, report output parity, and end-to-end execution timing against established delivery SLAs. The parallel run was extended to cover a peak weekend period for the highest-volume reporting workstreams, following the discovery that average load testing was insufficient to validate Lambda timeout thresholds under peak data volumes. Two Lambda functions were optimised following peak period parallel run findings before cutover was approved.

Phase 4 — Cutover & Decommission

Individual state machines were cut over sequentially by reporting workstream, with the legacy batch jobs for each workstream retired only after successful parallel run validation. Cutover was executed workstream by workstream to eliminate the risk of simultaneous multi-workstream transition and maintain continuous report delivery to business and operational stakeholders throughout the migration. The full on-premises batch orchestration framework was decommissioned following completion of the final workstream cutover within the six-week engagement timeline.

Solution Architecture

The target architecture replaced the monolithic on-premises batch framework with a distributed, serverless orchestration layer built around AWS Step Functions as the central workflow coordination engine. Each reporting workstream is managed as an independent state machine — providing native branching, conditional logic, parallel execution paths, configurable retry policies with exponential backoff, and structured Catch state error routing. AWS Lambda provides the independently scalable compute layer for each discrete processing step. Amazon S3 stores report outputs in structured prefix hierarchies aligned to downstream business intelligence and operational tooling consumption patterns. Amazon CloudWatch provides unified execution observability with SLA-aligned alarm configuration and Logs Insights query capability across all state machines.



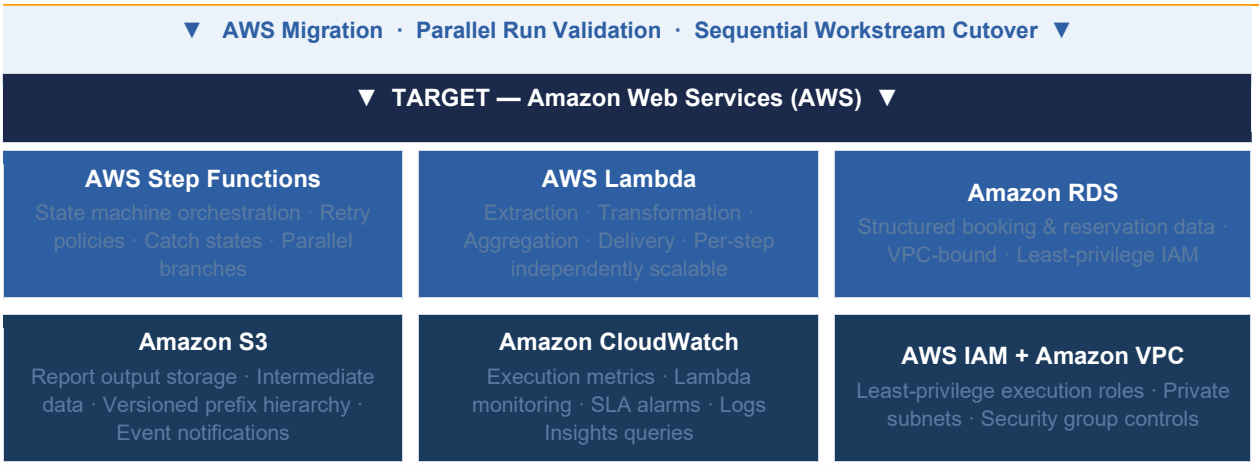


Figure 1: Serverless AWS Target Architecture — On-Premises Batch Framework Fully Decommissioned

AWS Services Used

AWS Step Functions	**AWS Lambda**	**Amazon RDS**
Amazon S3	**Amazon CloudWatch**	**AWS IAM**
Amazon VPC		

Business Benefits & Outcomes

Operational Outcomes

Automated retry logic, structured error handling, and CloudWatch-driven observability eliminated the primary labor categories driving the 15–18 hour weekly operations burden of the legacy framework. Operations engineering time dedicated to booking report orchestration management was reduced from approximately 15–18 hours per week to under 3 hours per week — an 82% reduction — recovering an estimated 630–780 engineering hours annually for redirection to higher-value data engineering and analytics initiatives. Mean time to detect a pipeline failure dropped from 35–45 minutes under manual monitoring to under 2 minutes through CloudWatch alarm notification. Mean time to diagnose a failure fell from approximately 90 minutes of manual log triage to under 12 minutes using Step Functions execution history.

Reliability & Report Delivery Outcomes

Under the legacy framework, partial pipeline failures during peak booking periods resulted in delayed or incomplete report delivery an estimated 3–4 times per month — with finance and revenue management teams regularly receiving incomplete booking consolidation reports during morning reporting windows on high-demand days. Following migration, Step Functions state machine execution success rates across all reporting workstreams exceeded 99.2%, with report delivery consistency improving to 98.6% on-time delivery across the first 90 days of production operation, compared to an estimated 84% under the legacy framework.

Peak Period Scalability Outcomes

The serverless architecture delivered its most visible operational impact during the first major peak event period following go-live — a high-occupancy holiday weekend driving simultaneous booking report demand across multiple properties and business units. Lambda functions scaled automatically to accommodate the surge in extraction, transformation, and aggregation workload without manual intervention. Step Functions parallel branches executed independent report workstreams simultaneously, delivering peak period report generation throughput approximately 3.4x above the legacy framework's demonstrated peak capacity — with full on-time SLA compliance and zero manual escalations.

Observability Outcomes

Step Functions execution history provided state-level traceability for every report generation cycle — showing exactly which state succeeded, which failed, what error was returned, how many retries were attempted, and the final execution outcome — accessible through the AWS console without log query or script execution. Business and finance stakeholders given read-only access to Step Functions execution history reduced inbound status enquiries to the operations team by an estimated 60–70% in the first month post-launch, as stakeholders self-served report delivery status without raising support requests.

Cost Outcomes

The consumption-based pricing model of AWS Lambda and Step Functions replaced fixed batch job server infrastructure costs with variable spend scaling directly with actual report generation workload. Total orchestration infrastructure cost was reduced by approximately 38% on a fully-loaded annual basis, inclusive of Lambda, Step

Functions, S3, and CloudWatch costs against the prior on-premises server, licensing, and operational labor baseline. A post-migration optimization review — covering Lambda memory tuning, Step Functions execution path consolidation, and S3 lifecycle policy configuration — delivered a further estimated 12% reduction in ongoing running costs relative to the initial post-migration baseline.

Before vs. After

Metric	Before (On-Premises Batch)	After (AWS Step Functions)
Report Delivery Failures	3–4 per month during peak periods	**97% reduction — near-zero with automated recovery**
On-Time Report Delivery Rate	~84% across peak and off-peak periods	**98.6% — first 90 days post-migration**
Execution Success Rate	Indeterminate — no state-level tracking	**99.2% across all Step Functions workstreams**
Mean Time to Detect Failure	35–45 minutes — manual log inspection	**Under 2 minutes — CloudWatch alarm**
Mean Time to Diagnose Failure	~90 minutes — manual log triage	**Under 12 minutes — Step Functions execution history**
Ops Management Effort	15–18 hrs/week — monitoring & restarts	**Under 3 hrs/week — 82% reduction**
Transient Failure Recovery	Manual restart — ops team required	**Automated — Step Functions retry policies**
Peak Throughput Capacity	Fixed — sequential batch execution	**3.4x legacy peak — elastic parallel execution**
Infrastructure Cost Reduction	—	**~38% on fully-loaded annual basis**
Engagement Timeline	—	**6 weeks — discovery to decommission**
Source Environment	—	**Fully decommissioned post-cutover**

TCO Analysis

The AWS Partner conducted a structured Total Cost of Ownership analysis comparing the fully-loaded on-premises batch orchestration framework against the AWS Step Functions target state over a three-year horizon across four cost categories.

Cost Category	On-Premises Baseline	AWS Step Functions Target
Infrastructure & Licensing	Fixed batch job servers, OS licensing, hardware maintenance, data center overhead — provisioned for peak, paid year-round	**Eliminated — Step Functions and Lambda pricing is consumption-based, scaling with actual workload demand**
Operational Labour	15–18 hrs/week — manual monitoring, log inspection, failure diagnosis, pipeline restarts; dedicated peak-period coverage	**Under 3 hrs/week — 82% reduction; 630–780 engineering hours/year recovered for data engineering initiatives**
Report Failure Risk	3–4 delivery failures/month during peak; escalation labor, stakeholder communication overhead, delayed revenue and ops decisions	**97% reduction in delivery failures — automated recovery eliminates escalation overhead and business impact of delayed reporting**
Scalability Overhead	Manual capacity planning before peak events; sequential execution bottlenecks structurally unresolvable without architecture redesign	**Eliminated — Lambda and Step Functions scale automatically; parallel branch execution delivers 3.4x legacy peak throughput**
3-Year TCO Outcome	Baseline — full cost model including infrastructure, labour, failure risk, and scalability overhead	**~38% total cost reduction on a fully-loaded annual basis**

Lessons Learned

The engagement delivered successful outcomes across all defined success criteria. Six key lessons were identified that strengthen the repeatable serverless orchestration framework applied to future batch modernisation workloads.

1. Workflow Graph Documentation Must Precede State Machine Design — No Exceptions

Complete directed workflow graph documentation — mapping every extraction job, transformation dependency, aggregation step, and delivery target — is a mandatory gate before state machine design begins. Two of the reporting workstreams contained undocumented dependencies implicitly sequenced by shared file system state. Had state machine design begun concurrently with documentation, the resulting architecture would have introduced race conditions difficult to diagnose in production. Timeline pressure is not a valid reason to compress this gate.

2. Error Classification Is a Design Activity — Not a Configuration Detail

Retry policies and Catch states serve fundamentally different purposes and must be assigned based on explicit error classification before build begins. Treating all Lambda execution failures as retry candidates would have resulted in persistent failures being retried to exhaustion before routing to diagnostic paths. A formal error classification exercise — categorising every anticipated failure type as transient, persistent, or data quality — must be conducted before retry and Catch configurations are finalised.

3. Third-Party API Extraction Must Be Isolated in Dedicated Lambda Functions

Co-locating API extraction logic and transformation logic within the same Lambda functions creates fragile, difficult-to-debug functions where API authentication failures and rate limiting responses from third-party systems are mixed with transformation logic. Dedicated Lambda functions per third-party data source — independently observable, retryable, and deployable — is a non-negotiable architectural principle for any orchestration workload integrating external APIs.

4. Parallel Run Scope Must Include a Representative Peak Period

Average load validation is insufficient for gaming and hospitality orchestration workloads where peak booking periods drive materially higher data volumes and API call rates. The parallel run for the highest-volume workstreams must cover at least one representative peak period before cutover is approved. In this engagement, peak period parallel run testing revealed Lambda timeout threshold proximity that was invisible under average load — and was resolved before go-live.

5. Step Functions Execution Visibility Is a Stakeholder Tool, Not Just an Operations Tool

Providing read-only Step Functions console access to business and finance stakeholders — not just operations and engineering teams — reduced inbound status enquiries by an estimated 60–70% in the first month post-launch. Positioning execution history as a stakeholder-facing transparency capability, with appropriate read-only access configured at go-live, should be a standard practice in all future orchestration migration engagements.

6. S3 Output Structure Must Be Designed From the Downstream Consumer's Perspective

S3 prefix design organised by execution metadata — logical from the orchestration platform's perspective — created friction for downstream business intelligence and operational tooling that expected outputs organised by business unit, property, and report type. A downstream consumption mapping exercise conducted during Assessment, alongside workflow graph documentation, must be a mandatory input to S3 architecture design for all future orchestration engagements.

About the AWS Partner

Agateverse is an AWS Partner specializing in cloud modernization, serverless architecture, and data engineering for enterprise customers across gaming, hospitality, and regulated industries. With deep expertise in AWS Step Functions, Lambda, and event-driven architecture, the partner delivers structured, low-risk batch-to-serverless migration frameworks that eliminate operational overhead while improving reliability, scalability, and observability. The partner's engineering team has executed complex orchestration migrations across hospitality, financial services, and public sector verticals, with a proven methodology validated against the most demanding mission-critical reporting workloads.

AWS Partner Engagement Summary

This engagement meets all competency criteria: workloads originated entirely from on-premises batch infrastructure outside AWS; the source environment was fully decommissioned post-migration; the solution addresses a unique multi-step orchestration workload (extraction, transformation, aggregation, report delivery) across a regulated gaming and hospitality customer; and quantified outcomes are documented across reliability, scalability, observability, operational efficiency, and cost.