worldskills

Test Project

Module Architecting 1

Cloud Computing

Submitted by:
Amazon Web Service, Independent Test Project Designer

WSC2026_TP53_en

Contents

This Test Project consists of the following documentation/files:

1. WSC2026_TP53_WellArchitect_1_actual_enn (this document)
2. Architecture diagram
3. Game Assets (provided via S3)

Use of AI Tools and Models

You may use the AI model provided in the competition environment to assist you while building your solution. You may not use any external or third-party AI model or service — from any device, personal account, or self-hosted on competition infrastructure. Your deployed solution may invoke only the Amazon Nova and Amazon Titan models provided in the environment; where a task specifies a model, use that model.

AI Intelligent Transformation of Insurance Claims Settlement

Task Description

XYZCompany is a London-headquartered international insurance group with branch offices in over 30 countries, processing more than 2 million insurance claims annually. As business volume continues to grow and fraud techniques become increasingly sophisticated, the company has decided to undertake a comprehensive AI-driven transformation of its traditional manual claims processing workflow.

After six months of requirements analysis and architecture design, the AI product team has completed the system blueprint — an intelligent processing platform covering the full claims lifecycle. The platform consists of four core modules:

- **Intelligent Document Processing Pipeline:** Automatically ingests claim materials, extracts text via document processing and applies AI models for risk classification and assessment.
- **Knowledge Base**: Vectorizes and indexes insurance policy documents and claim materials, supporting natural language retrieval.
- **Agent Multi-Dimensional Fraud Detection**: AI Agents analyze claims in parallel across multiple dimensions — text anomalies, amount anomalies, relationship networks, and historical case matching — aggregating evidence to produce a comprehensive determination.
- **Intelligent QA Web Service:** Serves global branch office staff, providing knowledge-base-powered policy queries and claims Q&A, with CDN acceleration, edge firewall protection, and content safety filtering.

The product team has provided architecture design documents, sample datasets, a pre-compiled Web application binary, and detailed configuration specifications. The project is now formally handed off to the cloud architecture team for implementation. As the cloud architect receiving this handoff, you are responsible for building the entire infrastructure from scratch on AWS, deploying all components, establishing event-driven communication between modules, and ensuring end-to-end system reliability.

CAUTION

Please use the **us-east-1** region.

Background

XYZCompany's claims business covers five major insurance categories: auto, medical, property, life, and travel. Customers submit claim materials (PDFs, scanned images, handwritten forms, and other formats) via email, mobile app, or partner offices. The traditional process relies on manual reviewers processing claims one by one, with an average review cycle of 5–7 business days and a fraud detection rate below 40%.

The manual workflow creates three strategic risks: customer dissatisfaction from slow resolution, financial losses from undetected fraud, and inability to scale with growing policy volume across 30+ international branches.

The new platform aims to reduce per-claim automated processing time from days to minutes while increasing the fraud detection rate from below 40% to above 85% through multi-dimensional AI analysis. It provides branch office staff across 30+ countries with instant, AI-powered policy and claims Q&A via a low-latency web service. Content safety filtering ensures that inappropriate model outputs never reach staff or persist in records. The system maintains encryption at rest, comprehensive audit trails, and point-in-time data recovery capabilities to satisfy insurance regulators across all operating jurisdictions.

The platform adopts an event-driven, loosely coupled architecture where modules communicate through asynchronous events — storage notifications and database change streams trigger downstream processing without direct inter-module coupling. This design enables independent scaling — document processing can burst during peak submission hours without impacting the QA web service. Failures in one module do not cascade to others; if fraud detection encounters an error, document ingestion and QA continue uninterrupted. New modules (e.g., future compliance engines, SLA monitors) can subscribe to existing event streams without modifying producers. Every state transition is an observable event, supporting end-to-end traceability from document upload through fraud determination.

See Technical Details below for precise implementation requirements.

Initial State

You will be provided with:

- IAM Roles: Pre-configured IAM roles with isolated permissions for each component
- Game Asset Bucket: Insurance policy documents and claim material files stored in the designated S3 bucket s3://wsc-tp53-2026-game-asset-day1
- Config Template: Configuration template for the Web application
- Fraud Cases: Historical fraud case files for knowledge base population
- Web Binary: A pre-compiled Web application binary (Linux x86_64)

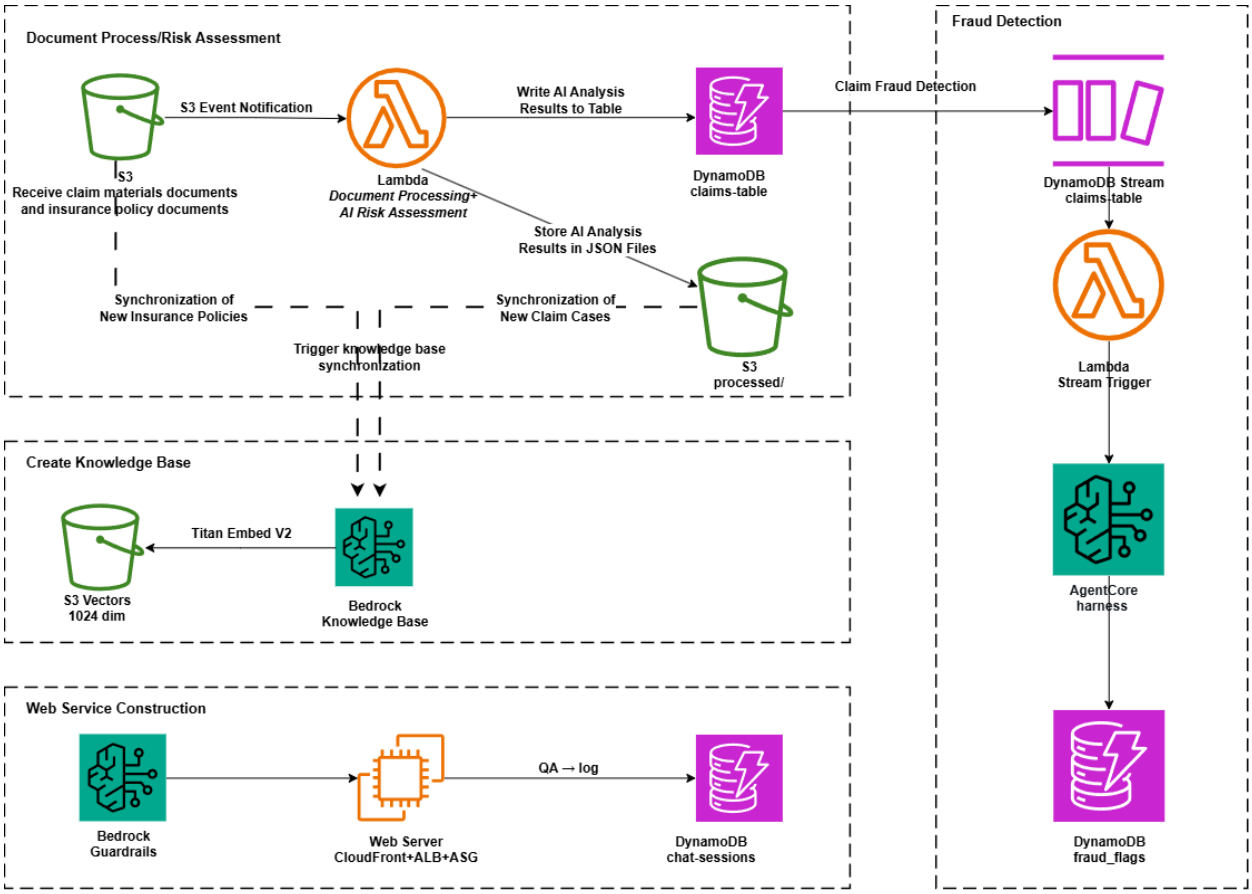
The Web application is a binary file. You cannot view or modify the source code. Configuration is done by placing a config.json file in the same directory as the binary.

Game Assets

Download the necessary game assets from the S3 bucket provided in your account:

RESOURCE PATH	DESCRIPTION
s3://wsc-tp53-2026-game-asset-day1/claims-ai	Web application binary (Linux ELF executable)
s3://wsc-tp53-2026-game-asset-day1/config.json	Web configuration template
s3://wsc-tp53-2026-game-asset-day1/samples/*	Insurance policy and claim material documents
s3://wsc-tp53-2026-game-asset-day1/fraud-cases/*	Historical fraud cases for knowledge base
s3://wsc-tp53-2026-game-asset-day1/DynamoDB_Table_Definitions.md	DynamoDB table structure definitions
s3://wsc-tp53-2026-game-asset-day1/Bedrock_Guardrails.md	Bedrock Guardrails configuration guide
s3://wsc-tp53-2026-game-asset-day1/Case_Evaluation_Indicators.md	Risk assessment and fraud detection field guidelines
s3://wsc-tp53-2026-game-asset-day1/AI_Output_Samples_One.json	Document recognition output sample 1
s3://wsc-tp53-2026-game-asset-day1/AI_Output_Samples_Two.json	Document recognition output sample 2

System Architecture



Note: The architecture above is for reference only. You may freely choose services and design your solution within the available permissions.

Services

SERVICE	RESPONSIBILITY	AWS SERVICE
Document Ingestion	Unified entry point for insurance policy and claim material documents	S3
Intelligent Document Processing Pipeline	Text extraction, claims classification, risk assessment	Lambda, Bedrock, S3 Event Notifications
Knowledge Base	Semantic search of insurance policies and claim materials	Bedrock Knowledge Base, Amazon S3 Vectors
Fraud Detection	Multi-dimensional parallel fraud analysis	Bedrock AgentCore Harness, Lambda, DynamoDB Streams
Web Service	Intelligent QA for global branch offices	EC2, Auto Scaling group, Application Load Balancer, CloudFront, WAF

Data Layer

COMPONENT	DATA STORE
Claims Data Table	DynamoDB claims-table (PK: claim_id)
Fraud Flags Table	DynamoDB fraud_flags (PK: claim_id)
Chat Sessions Table	DynamoDB chat-sessions (PK: session_id)
Document Storage	S3 bucket (create your own, e.g. claims-processing-1231123)
Vector Storage	Amazon S3 Vectors (1024 dimensions)

Tasks

- Before you begin: Access the Cloud Raiser platform and thoroughly review the document guidelines.
- Earn points based on system functionality as outlined in Score Events and Scoreboard.
- **TASK 1.** Build the intelligent document processing pipeline: S3 storage, event-driven automation, document recognition, AI classification, durable failure handling, and claim-object observability.
  Submission: S3 Bucket URI
- **TASK 2.** Create a knowledge base from insurance policy documents and claim materials, with support for real-time, automatic incremental synchronization.
  Submission: Knowledge Base ID
- **TASK 3.** Implement an Agent multi-dimensional fraud detection system: supervisor/router multi-agent collaboration with specialist agents and DynamoDB Stream trigger.
  Submission: Fraud AgentCore Harness ARN
- **TASK 4.** Deploy a highly available AI QA Web service: Guardrails safety controls shared across fraud and customer-facing agents, multi-AZ compute behind an Application Load Balancer, global acceleration, WAF protection, HTTPS, and origin trust via the CDN.
  Submission: HTTPS URL

Technical Details

This section specifies all mandatory requirements. Every item uses imperative language (must, shall, prefer) and represents a non-negotiable constraint on the implementation.

1. Data Protection and Encryption

- Sensitive claim objects and case records must be encrypted at rest using customer-managed KMS keys (CMK). This applies to both the primary object store (S3 bucket for raw/ and processed/) and the claims system-of-record (DynamoDB claims-table).
- The fraud_flags table must also be encrypted at rest (same sensitivity class as claims-table).
- Object version history must be enabled on the intake bucket so that mistaken overwrites of claim evidence can be rolled back.
- Deletion protection must be enabled on the claims system-of-record table to prevent accidental table deletion.
- CloudFront distribution must redirect all HTTP requests to HTTPS; no plaintext content delivery allowed

2. Resilience and Recovery

- Investigators must be able to restore structured claim rows to any second within roughly the last 35 days after a bad write (point-in-time recovery).
- Failed asynchronous Lambda invocations must land in a durable failure destination (Dead Letter Queue or On-Failure Destination) rather than being silently discarded.
- Lambda function timeouts must reflect real document processing size and model inference latency — do not use default 3-second timeouts for AI workloads.
- Raw objects older than approximately 90 days must transition to lower-cost storage via lifecycle rules.
- Application compute must run behind an Application Load Balancer spanning multiple Availability Zones. Unhealthy instances must be replaced using load-balancer health signals, and fleet size must track measured request pressure.

3. Observability and Tracing

- Enable end-to-end distributed tracing for each claim object from ingestion through parsing to table writing .
- Retain CloudWatch logs for all processing functions (document pipeline, fraud orchestration, stream triggers).
- Publish Bedrock model invocation logging so that model calls used in claim decisions remain auditable and reviewable.
- Maintain a CloudWatch Dashboard with at least four widgets covering pipeline health, fraud processing, and Web service metrics.
- Configure at least three CloudWatch Alarms on critical pipeline or Web service signals.
- Create at least one SNS topic for alarm delivery.

4. Network and Access Control

- Application compute must run in private network space (private subnets) behind the load balancer.
- Network exposure must follow least privilege — security groups allow only the ports the load balancer and required dependencies need.
- Public traffic must terminate at CloudFront. The application origin must reject requests that did not arrive through the CDN — enforce origin lockdown using a shared secret header.
- The CDN must include web application firewall (WAF) controls capable of absorbing abusive request floods against the QA endpoint (rate-based rules required, not only signature rule packs).
- Cache static assets aggressively at the edge while keeping dynamic QA responses appropriately fresh.

5. Compute and Scaling Constraints

- EC2 instance types must not exceed t3.medium (or t2.medium equivalent).
- Auto Scaling must use target tracking based on Application Load Balancer request metrics.

6. Content Safety and Guardrails

- A single Bedrock Guardrail must be created and applied on at least two generative paths: (1) the fraud orchestration/specialist Converse path, and (2) the customer-facing Web QA path.
- The Guardrail must include all four policy types: PII protection, topic filtering, word filtering, and contextual grounding.
- Do not use Amazon Bedrock Agents Classic (CreateAgent / agent guardrailConfiguration). You have access to AgentCore Harness. Apply Guardrails via the Fraud path and the Web application config.json.
- See Bedrock_Guardrails.md in Game Assets for exact policy configuration requirements.

7. Agent Architecture Constraints

- The fraud detection module must use a supervisor/router-style architecture: one AgentCore Harness coordinating multiple specialist Lambda tools.
- At least two specialist Lambda tools must be wired to the Harness.
- The fraud_flags output record must contain evidence from at least three different analysis dimensions (text_anomaly, amount_anomaly, network_analysis, historical_pattern) regardless of whether anomalies are found.
- The AgentCore Harness lifecycle must be configured with idleRuntimeSessionTimeout ≤ 120 seconds and maxLifetime ≤ 1800 seconds.
- Every InvokeHarness call must be followed by StopRuntimeSession in a finally block.
- DynamoDB Streams on claims-table must trigger the fraud automation path.

Service Details

IAM Roles

The following pre-configured IAM roles are available:

ROLE	PURPOSE
game-lambda-role	Claims processing pipeline
game-ec2-role	EC2 Web QA application
game-bedrock-kb-role	Knowledge Base service role
game-fraud-agent1-role	Text anomaly detection specialist
game-fraud-agent2-role	Amount anomaly detection specialist
game-fraud-agent3-role	Relationship network analysis specialist
game-fraud-agent4-role	Historical case matching specialist
game-fraud-harness-role	AgentCore Harness execution role
game-fraud-orchestrator-role	Fraud orchestration Lambda
game-fraud-stream-role	DynamoDB Stream trigger Lambda
game-fraud-gateway-role	AgentCore Gateway execution role

Database

For detailed table structures, refer to: s3://wsc-tp53-2026-game-asset-day1/DynamoDB_Table_Definitions.md

TABLE NAME	TYPE	PRIMARY KEY	PURPOSE
claims-table	DynamoDB	PK = claim_id	Store AI-processed claim data
fraud_flags	DynamoDB	PK = claim_id	Store fraud detection results
chat-sessions	DynamoDB	PK = session_id	Store QA conversation logs

Knowledge Base

Create a Bedrock Knowledge Base with the following configuration:

- **Data Sources**: s3://<bucket>/policies/ + s3://<bucket>/processed/
- **Embedding Model**: amazon.titan-embed-text-v2:0
- **Vector Storage**: Amazon S3 Vectors (1024 dimensions)
- **Chunking Strategy**: FIXED_SIZE, 512 tokens, 20% overlap

Intelligent Document Processing and Fraud Pipeline

- **Environment variables**: Externalize runtime parameters — model identifiers, guardrail identifiers, table names, bucket names, and resource ARNs — as AWS Lambda environment variables rather than hard-coding them.

Guardrails and Agents

- Follow the specification in s3://wsc-tp53-2026-game-asset-day1/Bedrock_Guardrails.md. Reference the same guardrail_id in both the AgentCore Harness environment and Web config.json.

Scoring

Deliverables

Each task's required submission is listed inline above. Monitor your Score Events and Scoreboard to track progress.

Traffic for each task will begin pushing progressively between 60 and 120 minutes after the competition starts.

Reference Documents

https://docs.aws.amazon.com/

https://docs.aws.amazon.com/bedrock/latest/userguide/

https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/

https://docs.aws.amazon.com/elasticloadbalancing/latest/application/

https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/

https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html