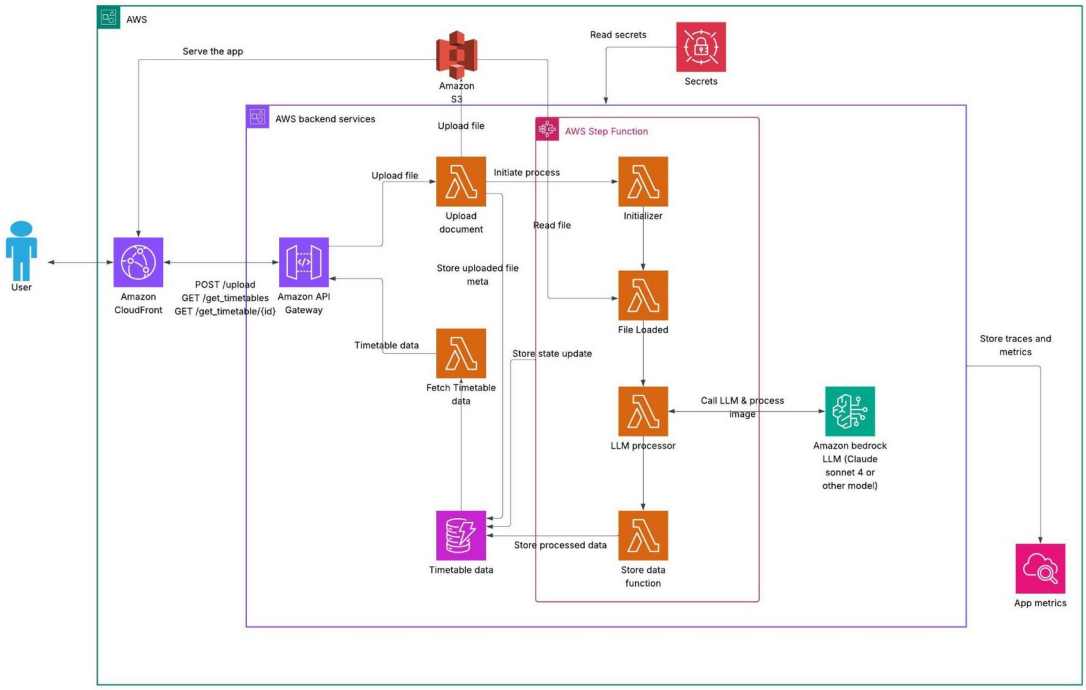
Teacher Timetable Extraction Platform – AWS-based Solution Plan

Teacher Timetable Extraction Platform – AWS-based Solution Plan



System architecture

1 Requirement overview

Teachers will upload their weekly timetables in diverse formats (scanned images, PDFs or Word documents). The system must extract time-block events (e.g., subjects such as “Maths”, registration periods, lunch breaks), detect start and end times/duration, preserve the original names and notes, and render the extracted timetable in the platform’s user interface. The extraction pipeline must handle typed or handwritten text, coloured tables and various layouts. This report proposes a set of planned activities and the AWS services that can be used to design, build and deliver such a solution.

2 Proposed AWS solution

2.1 Architecture overview

File upload & storage – A public-facing API (Amazon API Gateway) front-ends a POST endpoint for file uploads. The API triggers an AWS Lambda function that stores the uploaded

file in an Amazon S3 bucket. S3 is a fully-managed object storage service that can store virtually unlimited data and offers high durability (11 nines) and scalability (datacamp.com).

Workflow orchestration – An AWS Step Functions state machine orchestrates the extraction pipeline. Step Functions allow you to coordinate distributed components using visual workflows; they support task states that invoke services like Lambda, choice states for branching logic and parallel states for concurrent processing (medium.com). The service is serverless, automatically scales and provides built-in error handling and retries (medium.com).

Document processing – A Lambda task downloads the file from S3 and determines its type (PDF, image, DOCX). It can convert documents into images (e.g., using LibreOffice or Pandoc in a container) and then call Amazon Textract to extract text, handwriting, key-value pairs and tables from the document. Textract is a fully managed ML-powered OCR service that goes beyond traditional OCR by identifying relationships between data points such as key-value pairs and tables (cloudoptimo.com). It can extract raw text and handwriting, identify key–value pairs, extract tables while preserving rows and cells, handle multi-page documents and process documents synchronously or asynchronously (cloudoptimo.com).

LLM-based interpretation – The text and table data returned by Textract are unstructured. A Lambda function (or containerised microservice) uses Amazon Bedrock to interpret the extracted data and map them to a structured timetable. Bedrock is a fully managed generative-AI service that provides a unified API to access multiple state-of-the-art foundation models—including Anthropic Claude, Cohere Command/Embed, Jurassic, Llama 3, Mistral and Amazon's Titan models (caylent.com). Bedrock's diversity of models, scalable infrastructure and enterprise-grade security allow us to choose the model that best interprets timetable information (caylent.com). We can design prompts to ask the model to output a JSON list of events with day, start_time, end_time, subject_name, and notes fields, preserving names and notes.

2.2 Database schema suggestion

Each **TimeBlock** record can include fields: timetable_id (partition key), day_of_week, start_time (ISO 8601), end_time, duration_minutes, event_name, notes, teacher_id and optionally source_file_url. DynamoDB's flexible schema suits variable numbers of events per day. A separate **Timetable** entity stores metadata about the upload (teacher, upload timestamp, status) and is linked via timetable_id.

3 Planned activities and phases

The table below summarises the high-level phases and key activities. Detailed steps and deliverables are described in the following subsections.

Phase	Key activities (short)	Relevant AWS services
1 Requirement gathering	Capture functional & non-functional requirements, analyse sample timetables, define success criteria	–

Phase	Key activities (short)	Relevant AWS services
2 Architectural design	Draft architecture & data flow diagrams, select services & frameworks, create database schema	S3, API Gateway, Lambda, Step Functions, Textract, Bedrock, DynamoDB/RDS, SQS, CloudFront
3 Environment setup	Set up AWS accounts, IAM roles & policies, create S3 buckets, configure CI/CD	IAM, S3, CodeCommit/CodePipeline
4 Backend prototype	Build file-upload API, store files in S3, orchestrate processing with Step Functions & Lambda	API Gateway, Lambda, S3, Step Functions
5 Document extraction	Convert files to images, integrate Textract OCR, parse tables & text	Textract, Lambda
6 LLM integration	Design prompts, call Bedrock to extract structured events, refine output	Bedrock, Lambda
7 Data persistence	Store structured timetable records, design queries & indexes, implement retrieval API	DynamoDB or RDS
8 Asynchronous processing	Decouple upload from processing using SQS queues, implement DLQ & retries	SQS, Step Functions
9 Frontend strategy	Choose UI framework (React), design responsive timetable component, plan hosting	S3 static hosting, CloudFront
10 Security & compliance	Implement authentication (Cognito), encryption (KMS), access control policies	Cognito, IAM, KMS
11 Monitoring & error	Configure CloudWatch logs & metrics, implement Step Functions retries, define fallback flows	CloudWatch, X-Ray, Step Functions

Phase	Key activities (short)	Relevant AWS services
handling		
12 Testing & QA	Develop unit tests, load tests, OCR accuracy tests using sample documents	–
13 Documentation & handover	Write README, API docs, architectural diagrams, create handover video	–

3.1 Requirement gathering

Stakeholder interviews – Meet teachers and product owners to understand how timetables are currently stored and used. Identify specific variations (handwritten vs. typed, languages, colours) and data privacy requirements.

Define acceptance criteria – Clarify what constitutes a successful extraction (e.g., events captured correctly, times parsed within ±5 minutes). Define performance metrics such as processing time, cost limits and acceptable error rates.

Collect sample data – Gather a diverse set of timetable documents (supplied example plus additional PDFs/images). Annotate ground truth for later evaluation.

3.2 Architectural design

Draw system diagrams – Create sequence diagrams and data-flow diagrams illustrating the S3 upload, Step Functions pipeline, OCR extraction, LLM interpretation and storage operations. Use diagramming tools (draw.io, Lucidchart or PlantUML). Annotate each service's responsibility.

Select AWS services – Use S3 for file storage due to its unlimited capacity and high durability (datacamp.com); Step Functions for orchestration because it coordinates distributed tasks and provides error handling (medium.com); Textract for OCR because it can extract text, key-value pairs and tables (cloudoptimo.com); Bedrock for LLM integration because it offers a variety of foundation models and unified API (caylent.com); DynamoDB for storing time-blocks due to its serverless scalability (n2ws.com); SQS for decoupling services and buffering messages (lumigo.io); API Gateway to expose endpoints (aws.plainenglish.io); Lambda for compute, which is serverless and automatically scales (serverless.com); CloudFront and S3 static hosting for front-end delivery (amnic.com).

Design data model – Sketch DynamoDB table design: Timetable table (PK: timetable_id) and TimeBlock table (PK: timetable_id, SK: start_time). Include teacher_id as a global secondary index for queries across timetables.

Identify non-functional requirements – Estimate throughput (number of uploads per day), target latency (e.g., <30 sec to process a timetable), security (encryption at rest & in transit, GDPR compliance), and cost constraints.

3.3 Environment setup

AWS account configuration – Create separate accounts for development and production. Set up AWS Organizations, implement cross-account roles for CI/CD.

IAM roles & policies – Define least-privilege IAM roles for Lambda, Textract, Bedrock and Step Functions. Example: a Textract role with textract:StartDocumentAnalysis and textract:GetDocumentAnalysis permissions (cloudoptimo.com). Define an S3 bucket policy to restrict access to specific principals.

Networking – For improved security, run Lambdas inside a VPC if necessary. Create VPC endpoints for S3, Textract and Bedrock to keep traffic within the AWS network.

CI/CD pipeline – Set up a Git repository (e.g., CodeCommit or GitHub). Use AWS CodePipeline/CodeBuild to automate deployments. Configure unit testing and static code analysis.

3.4 Backend prototype development

File-upload endpoint – Develop a Node.js Lambda function using an API Gateway trigger. Validate file types and size. Upload files to S3 using pre-signed URLs.

State machine implementation – Define a Step Functions JSON state machine. States: (a) CheckDocumentType; (b) ConvertToImage (if needed); (c) InvokeTextract; (d) ParseResult; (e) LLMInterpretation; (f) PersistResults; (g) NotifyUser. Configure retry policies and catch blocks for error handling.

Document conversion and OCR – Use Lambda to convert DOCX/ODT into PDFs or images. Use Textract synchronous AnalyzeDocument API for small documents or asynchronous StartDocumentAnalysis for large ones. The returned JSON contains blocks for lines, key-value pairs and tables (cloudoptimo.com).

Write parsing logic – Create functions to map Textract outputs into plain text. Identify table structures (rows/columns) and read time strings using regex or date-time parsing libraries. Send the extracted text to the LLM.

3.5 LLM integration strategy

Choose model – Evaluate models available in Bedrock; e.g., Amazon Titan or Anthropic Claude for extraction tasks. The advantage of Bedrock is model diversity and unified API (caylent.com).

Prompt design – Build a prompt that instructs the model to parse a timetable description and return structured JSON. Include system instructions, sample tables and few-shot examples. Use retrieval-augmented generation (RAG) if necessary by embedding the teacher's raw timetable into context.

Determinism and reproducibility – Use temperature=0 for deterministic output. Validate the JSON with schema definitions. Implement fallback logic: if the LLM returns invalid JSON or low confidence, route to manual review.

Cost & performance optimisation – Use asynchronous Bedrock calls where possible. Cache model responses for identical inputs. Monitor model latency and cost. Keep prompts short to reduce token usage.

3.6 Data persistence and API layer

Database schema – Implement DynamoDB tables with primary keys and sort keys to support queries by day, teacher or timetable. Use DynamoDB Streams to trigger notifications or downstream processes when new events are inserted.

Backend endpoints – Create RESTful endpoints: /upload (POST), /timetables/{id} (GET), /teachers/{teacher_id}/timetables (GET). Use API Gateway's built-in CORS and throttle settings.

Pagination and filtering – Implement query parameters for day, date range and subject filtering.

3.7 Asynchronous processing & error handling

SQS queue – After uploading a document, push a message onto an SQS queue. Worker Lambdas poll the queue to start the Step Functions state machine. This decouples ingestion from processing and smooths traffic spikes (lumigo.io).

Dead-letter queue – Configure a DLQ for failed messages. Use Step Functions' built-in retries and error catching to handle time-outs and service failures (medium.com).

Monitoring – Send processing metrics and errors to CloudWatch. Set up alarms for high error rates or latency. Use CloudWatch dashboards to display pipeline health.

3.8 Frontend strategy

Framework selection – Use React or Next.js for component-based UI and easy state management. Implement a responsive timetable component that can display a matrix of days vs. time slots. Use libraries like FullCalendar or a custom CSS grid.

Static hosting – Host the compiled static assets on S3 and distribute via CloudFront. CloudFront reduces latency by serving content from edge locations and provides built-in security and scalability (amnic.com).

User authentication – Integrate Cognito into the front-end for sign-in/sign-up flows. Use tokens to call protected backend APIs.

UX considerations – Provide visual cues for overlapping events, allow editing event names and notes before saving, and support internationalisation.

3.9 Security and compliance

Access control – Use Cognito user pools for authentication and assign IAM roles via identity pools. Secure the API with JWT authorisers.

Encryption – Ensure S3 buckets have server-side encryption; optionally enable S3 Object Lock and versioning. DynamoDB provides encryption at rest using KMS keys (n2ws.com).

Audit and logging – Enable CloudTrail to log API calls. Use VPC flow logs if Lambda functions run inside a VPC.

Data retention & privacy – Define retention policies. Implement lifecycle rules on S3 to transition or delete old uploaded files, using S3 lifecycle policies (datacamp.com).

3.10 Monitoring and observability

CloudWatch metrics – Collect metrics for Lambda execution time, error rates, Step Functions state transitions and DynamoDB read/write throughput.

X-Ray tracing – Instrument Lambda functions to capture traces. Visualise end-to-end request paths.

Alerts – Configure CloudWatch alarms and integrate with SNS or Slack for alerting when thresholds exceed (e.g., OCR failure rate > 5 %).

Dashboard – Build a dashboard summarising processing throughput, latency and success rate.

3.11 Testing & QA

Unit tests – Write tests for each Lambda handler and parsing function.

Integration tests – Use sample timetable documents to test the full pipeline. Validate that the extracted events match annotated ground truth.

Performance tests – Benchmark the pipeline under simulated load, measure latency, throughput and cost. Test Textract and Bedrock quotas.

Security tests – Conduct penetration testing and static analysis. Validate IAM policies and misconfiguration.

3.12 CI/CD and infrastructure as code

Infrastructure as Code – Use AWS CDK, Terraform or CloudFormation templates to define S3 buckets, Step Functions, IAM roles, DynamoDB tables and API Gateway resources.

Automated deployments – Configure CodePipeline or GitHub Actions to build and deploy backend code to Lambda and update the state machine. Use separate stages for dev, staging and prod.

Versioning & rollback – Enable versioning for Lambda functions. Use canary or blue/green deployments to minimise downtime.

3.13 Documentation & handover

Documentation – Maintain a detailed README describing how to set up the environment, install dependencies, and run the backend. Document API endpoints (methods, paths, request/response examples) and the database schema. Provide guidelines for prompt engineering and LLM configuration.

Architecture artefacts – Include diagrams illustrating the AWS architecture, state machine and data model. Explain the rationale for each service selection with references to the features (Textract's OCR capabilities (cloudoptimo.com), Bedrock's model diversity (caylent.com), Step Functions' orchestration features (medium.com), etc.).

Handover video – Prepare a video (using Loom or similar) demonstrating the pipeline: uploading a sample timetable, observing the processing and visualising the resulting JSON and UI display. Discuss design choices, lessons learned and how AI tools were used for planning and coding.

4 Conclusion

This plan describes the activities required to design, build and deliver a teacher timetable extraction platform using Amazon Web Services. The solution leverages managed services like S3 for durable storage (datacamp.com), Step Functions for orchestrating workflows (medium.com), Textract for extracting text and table data from documents (cloudoptimo.com), Bedrock for generative-AI interpretation of unstructured text into structured events (caylent.com), DynamoDB for scalable storage (n2ws.com), SQS for decoupling and buffering (lumigo.io), API Gateway for exposing HTTP endpoints (aws.plainenglish.io), Lambda for serverless compute (serverless.com) and CloudFront for fast, secure content delivery (amnic.com). By following the phases described above—requirement analysis, design, environment setup, backend prototype, extraction, LLM integration, persistence, security, testing and documentation—the team can deliver a robust, scalable and adaptable solution.