QUBER · INFRASTRUCTURE DESIGN

Cloud Deployment

Containerizing `quber table --review` as a per-document Fargate job

A PDF dropped into S3 becomes reviewable table artifacts through a containerized job that runs once per document and then exits. This reference describes the system's two independent parts: the deploy path that ships the image, and the runtime path that processes each document. It also covers the AWS resources, the security model, and the cost profile. It assumes familiarity with AWS ECS, Lambda, and S3.

Design draft · AWS us-east-1 · CPU-only · Haiku

Xavier Mandeng · 2026-06-20 · QUE-266

Contents

Audience and scope

This is for the engineer who will stand up or operate the deployment. It assumes you know AWS ECS, Lambda, and S3, and that you have seen `quber table --review` run locally. It covers one stage of infrastructure work: the fixed design decisions, the two paths of the system, the AWS resources involved, and the security and cost properties. It does not cover the extraction algorithm itself.

1 Overview

A PDF dropped into S3 becomes reviewable table artifacts through a containerized job that runs once per document and then exits. Everything lives in one AWS account, in `us-east-1`. The extraction path is quber's default set-of-mark engine, which runs entirely on CPU. Poppler and Camelot pull the tables. The Anthropic API on the Haiku model then finds each table in the page image and corrects its values against the page text. Finally pymupdf writes the review artifacts. No step needs a GPU, and nothing runs on RunPod.

The system splits into two parts. They never share a running process or a set of credentials. The **deploy path** ships the container image. The **runtime path** processes one document per invocation.

DEPLOY PATH · CI/CD

A git tag builds the image, pushes it to ECR, and registers a new ECS task-definition revision. It authenticates through GitHub OIDC, so no AWS access keys are stored anywhere.

RUNTIME PATH · PER DOCUMENT

An S3 upload fires a Lambda that launches one Fargate task. The task reads the PDF, calls Haiku, and writes three artifacts back to S3.

1.1 Decisions settled for this stage

These choices are fixed for this stage. Later sections assume them.

Area	Decision
Region	`us-east-1` — `qubera-docs` already lives there, so there is no cross-region S3 transfer.
IaC and state	Terraform with the native S3 lockfile (`use_lockfile=true`, no DynamoDB). State stays in our own account, versioned and SSE-encrypted.
Buckets	Input `qubera-docs` (exists) to output `qubera-extracts` (created). The output path mirrors the inbound path.

Area	Decision
Trigger	S3 event to a small Lambda to `RunTask`. Chosen over a direct EventBridge rule so every trigger is logged for audit.
Secrets	SSM Parameter Store SecureString (KMS): `ANTHROPIC_API_KEY`, `LOGFIRE_TOKEN`. Never baked into the image.
Deploy	GitHub Actions on a git tag or release only. CI never runs Terraform.
Observability	CloudWatch for application logs; Logfire for per-document, per-call Haiku cost traces.

2 Deploy path: repo to image

Pushing a version tag is the only action that ships code. GitHub Actions assumes a short-lived role through the GitHub OIDC provider, so no AWS access keys are stored anywhere. It then builds the CPU-only image, pushes it to ECR tagged with the git SHA, and registers a new ECS task-definition revision pointing at that image.

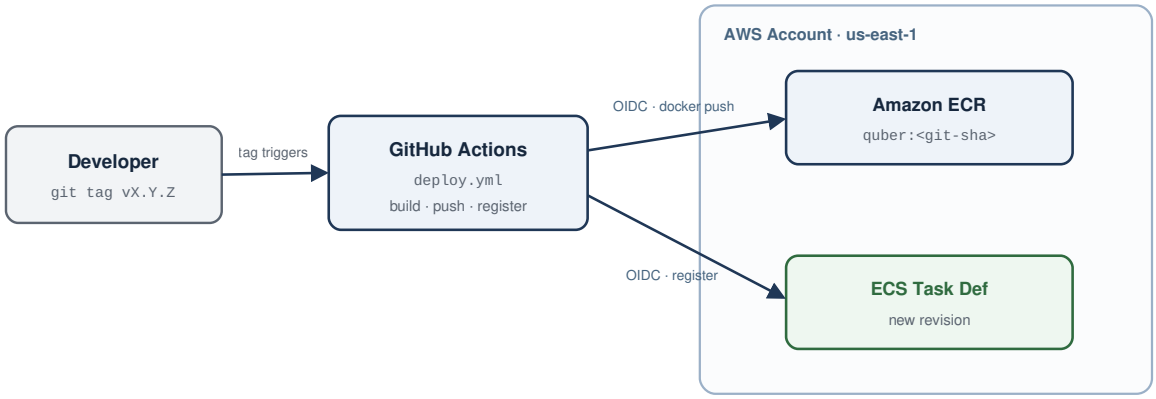

Figure 1 — A git tag builds the image, pushes it, and registers a task-definition revision. The operator applies Terraform separately.

WHY CI NEVER TOUCHES TERRAFORM

`deploy.yml` only builds, pushes, and registers. The operator applies infrastructure changes separately and infrequently. The short-lived CI runners therefore never hold Terraform state or broad infrastructure permissions.

3 Runtime path: per-document job

An upload starts a single Fargate task. The task reads the PDF, calls Haiku, writes the artifacts, and exits. Section 4 gives the numbered trace of that run.

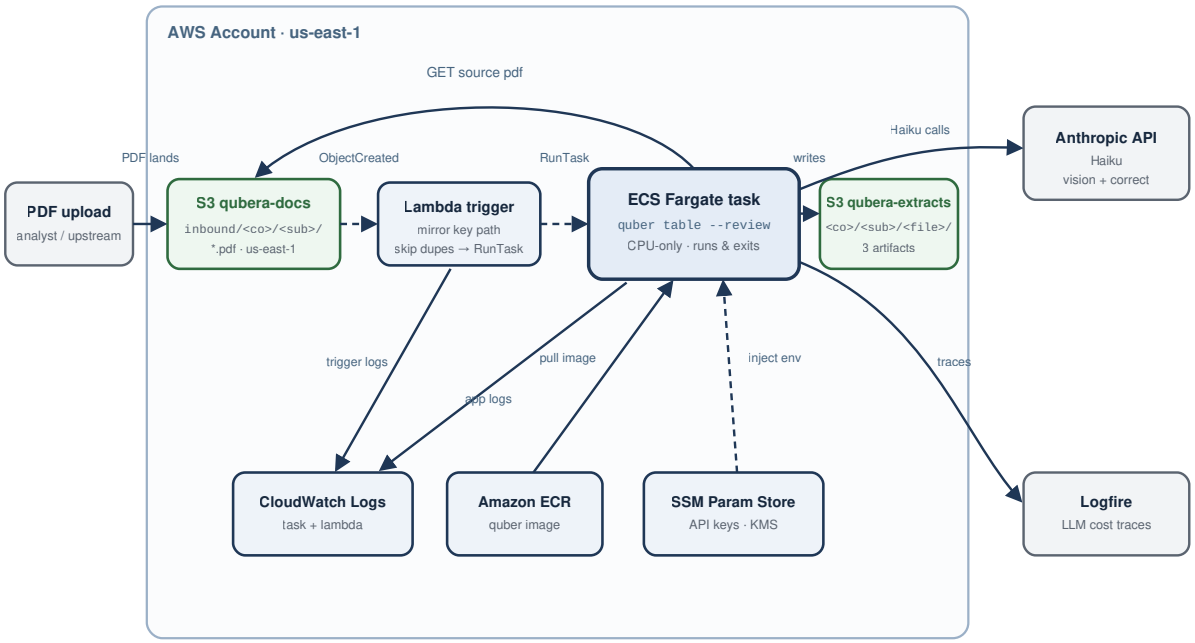

Figure 2 — One Fargate task per PDF. Dashed edges carry control or credentials; solid edges carry data.

4 Runtime flow

One upload runs through these six steps, then the task exits.

1. A PDF lands at `qubera-docs/inbound/<co>/<sub>/file.pdf`.
2. An S3 `ObjectCreated` event, filtered to the `.pdf` suffix, invokes the Lambda trigger.
3. The Lambda decodes the key, mirrors the path to the output bucket, skips duplicate and non-PDF events, and calls `RunTask`.
4. The Fargate task pulls the `quber` image from ECR and reads the two API keys from Parameter Store, which the execution role injects as environment variables.
5. The task GETs the source PDF from `qubera-docs`, then runs `quber table --review`: Camelot extracts the tables, and Haiku locates and corrects them.
6. It writes the three artifacts to `qubera-extracts/<co>/<sub>/file/`, streams logs to CloudWatch and cost traces to Logfire, then exits.

5 AWS component reference

Every resource the system uses, and what each one is for.

Resource	Purpose
Amazon ECR	Holds the `quber` container image, tagged by git SHA.
S3 · qubera-docs	Exists. Source PDFs under `inbound/<co>/<sub>/`. We add the `ObjectCreated` notification, not the bucket.
S3 · qubera-extracts	Created. Artifacts under a path mirroring the inbound one.
Lambda trigger	Decodes the S3 key, mirrors the path, skips duplicates and non-PDFs, calls `RunTask`. Logs every trigger.
ECS Fargate cluster + task def	Runs `quber table --review`, with CPU and memory sized for the CPU-only job. One task per document.
SSM Parameter Store	SecureString (KMS): `ANTHROPIC_API_KEY`, `LOGFIRE_TOKEN`.
CloudWatch Logs	Task and Lambda application logs.
GitHub OIDC provider + role	Lets `deploy.yml` push to ECR and register task defs with no static keys.
Task execution role	Pull from ECR, read Parameter Store, write CloudWatch logs.
Task role	Read `qubera-docs/inbound/*`, write `qubera-extracts/*`.
Lambda role	Call `RunTask`, `PassRole` the task roles, write its own logs.
Terraform state bucket	Versioned, SSE-encrypted, native S3 lockfile. One-time bootstrap.

6 Security and observability

6.1 Least privilege, no static keys

Each identity gets only what it needs. The task role reads exactly `qubera-docs/inbound/*` and writes exactly `qubera-extracts/*`. The execution role pulls images and reads the two parameters. The Lambda role can launch tasks and nothing else. CI authenticates through GitHub OIDC, so there are no long-lived AWS access keys to leak.

6.2 Secrets and state stay in our trust boundary

The API keys live in Parameter Store as KMS-encrypted SecureStrings, injected at runtime and never baked into the image. Terraform state stays in our own AWS account rather than a third-party service. This matters given the liability that comes with handling financial documents.

> **AUDITABILITY**
>
> The Lambda logs every trigger to CloudWatch. For financial documents, a file that is silently dropped with no record is unacceptable. The trigger is therefore a Lambda whose every invocation is logged, not a direct EventBridge rule that would leave no trace.

6.3 Cost profile

The job runs on CPU to completion and then exits, so compute is billed per run rather than per idle hour. At any realistic volume the dominant cost is Haiku API tokens, not infrastructure. Logfire attributes that spend per document and per agent, so it can be watched directly rather than reconstructed from a monthly invoice.

Status. This is a design draft for review, and pairs with the QUE-266 plan-review.