For Q1 Model

Proposed System Architecture – Live VMC Prediction API

Client → HTTPS → API Gateway → FastAPI service (Docker) → Model in memory
Cloud Object Store (model files)

Core components and tools

API gateway: AWS API Gateway. Off-loads TLS, rate-limiting, key validation. Zero servers to manage.

Container platform: AWS ECS Fargate (or Google Cloud Run). Run the FastAPI image without caring about EC2 nodes/K8s clusters. Scales to zero if traffic drops.

Serving app: FastAPI + Uvicorn in Python 3.11. Lightweight, async, automatic OpenAPI docs, good Pydantic validation.

Model artefact store: S3 bucket with versioning. Keeps model.joblib outside the image so we can swap models without rebuilding.

CI/CD: GitHub Actions → ECR → ECS blue/green deploy. One command (act) reproduces the pipeline locally and tags every image with Git SHA.

Secrets: AWS Secrets Manager. Store DB creds or signing keys, delivered as env vars at run-time.

Observability: CloudWatch logs + AWS X-Ray traces. No extra agents. Metrics and structured logs in one place.

Drift and retrain: Scheduled Lambda that checks daily residuals. if RMSE > threshold, fires CodePipeline to retrain in SageMaker. Keeps the model honest without manual babysitting.

Request life-cycle

1. **Sensor or mobile app** sends a POST /predict with {"normalized_value": 6123} and its API key.
2. **API Gateway** terminates TLS, verifies the key and forwards the JSON to the Fargate service.
3. **FastAPI container**

    - parses the request via Pydantic,
    - looks up (or has already cached) isotonic_model.joblib from S3,
    - returns {"predicted_vmc": 19.7}.
4. The round-trip is <50 ms if the container is warm.

Deploy and update workflow

1. **Code push** GitHub Actions runs tests and builds vmc_calib:<sha>.
2. Image is pushed to **ECR**.
3. **ECS service** is updated via blue/green: new tasks start, pass health checks, traffic shifts; old tasks drain.
4. **Model updates** (for example quarterly retrain) are just aws s3 cp new_model.joblib s3://vmc-models/2025-07-v2/ and an env var switch in the task definition.

For Q2 Model

Container layer

Serving image: Docker. Repeatable build of the whole stack, local and cloud.

Process manager: Gunicorn with Uvicorn workers. Starts multiple async workers for FastAPI, scales well on CPU and GPU nodes.

Each model resides in its own lightweight container image, produced by the same Dockerfile template but loaded with a different weight file.

Inference microservices

REST API: FastAPI. Endpoints: /predict/model_a and /predict/model_b. Accepts an image in multipart or a URL. Returns JSON with bounding boxes and confidences. automatic OpenAPI docs, async request handling and native Pydantic validation.

Model runtime: Ultralytics YOLOv8 + PyTorch. Loaded once per worker, kept in memory. GPU is used if available, otherwise CPU.

Pre and post-processing utilities: NumPy, OpenCV. Resize, tiling logic for full-frame trap photos, NMS.

Gateway and routing

HTTPS reverse proxy: NGINX. TLS termination, path-based routing to the two model services, gzip compression for JSON responses.

Authentication: AWS Cognito. Validates incoming tokens in NGINX with an auth plugin.

Orchestration and scaling

Cloud run-time: AWS ECS Fargate. Handles container scheduling and auto-scaling without full Kubernetes overhead.

GPU option: AWS G4dn or G5 tasks. Attach when higher throughput is needed.

Storage and artefact management

Model weights: S3 (versioned bucket). Keeps model binaries, pulled into the image during build or at start-up.

User uploads: S3 pre-signed URLs. API accepts image URL instead of raw file, avoids large payloads.

Logs and metrics: CloudWatch. Centralised aggregation, retained for audits.

Observability

Prometheus scrapes FastAPI and GPU metrics (yolov8 speed, memory) via metrics endpoint. **Grafana** dashboards show latency p50, p95, GPU utilisation. **Sentry** captures unhandled exceptions and returns HTTP 500 with a trace ID so users can report issues.

CI / CD pipeline

GitHub Actions

- Lint and unit tests (pytest)
- Build and push Docker images to **GitHub Container Registry**
- Deploy via **terraform apply** or **AWS Copilot**

Request flow summary

1. Client sends HTTPS POST /predict/model_b with a signed S3 image link.
2. NGINX terminates TLS, checks JWT, and routes to the Model B service.
3. FastAPI worker downloads the image, runs tiling inference if needed, assembles boxes, returns JSON.
4. Metrics exporter logs latency and number of detections.
5. Cloud autoscaler adds more replicas if CPU /GPU usage exceeds the threshold.