FlyBot on AWS: system architecture

How the parts of FlyBot on AWS fit together, what runs where, and which way the data goes. State of the system: 27 October 2026. The numbers and the reasons behind each choice are in the technical report; the interfaces are specified in CONTRACTS.md and the decisions in DECISIONS.md.

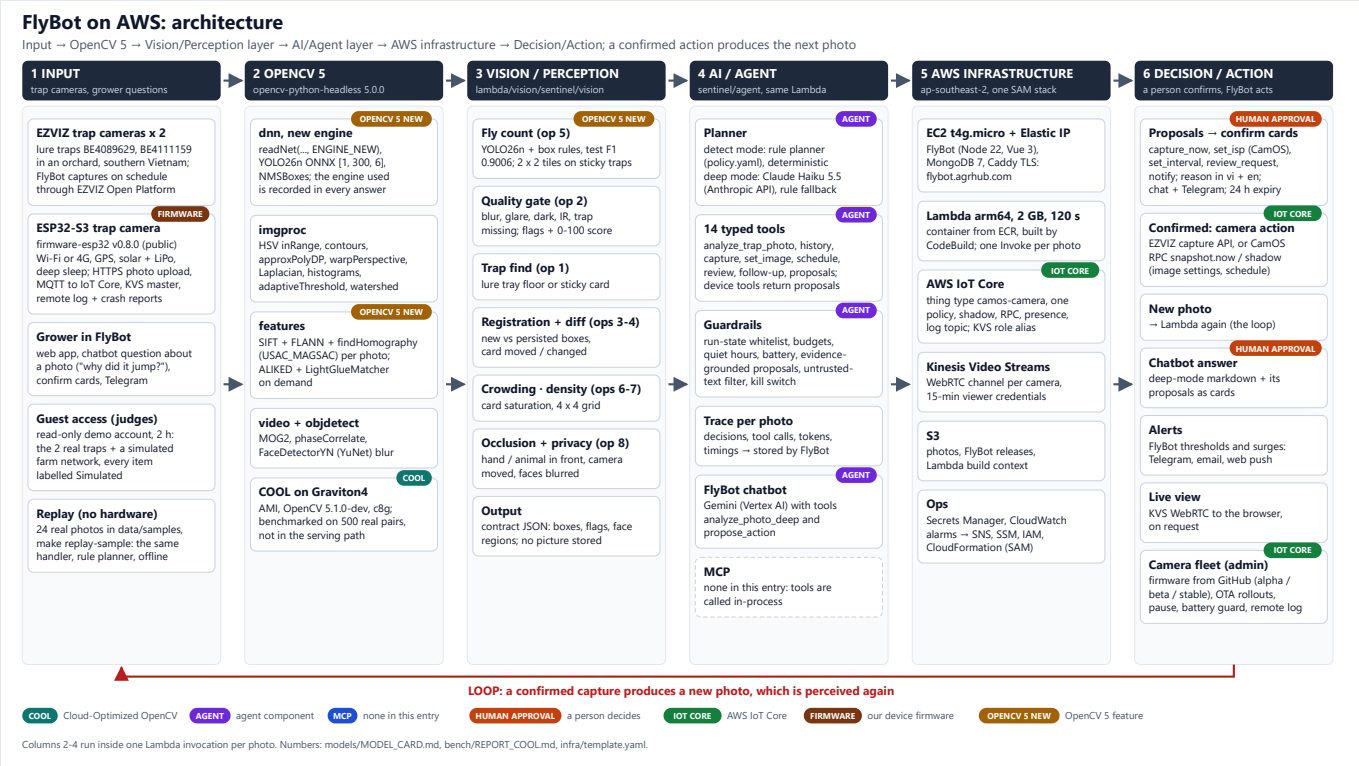


Figure 1. The six layers (`docs/diagrams/architecture.svg`). Markers: COOL, AGENT, MCP (none in this entry), HUMAN APPROVAL, IOT CORE, FIRMWARE, OPENCV 5 NEW.

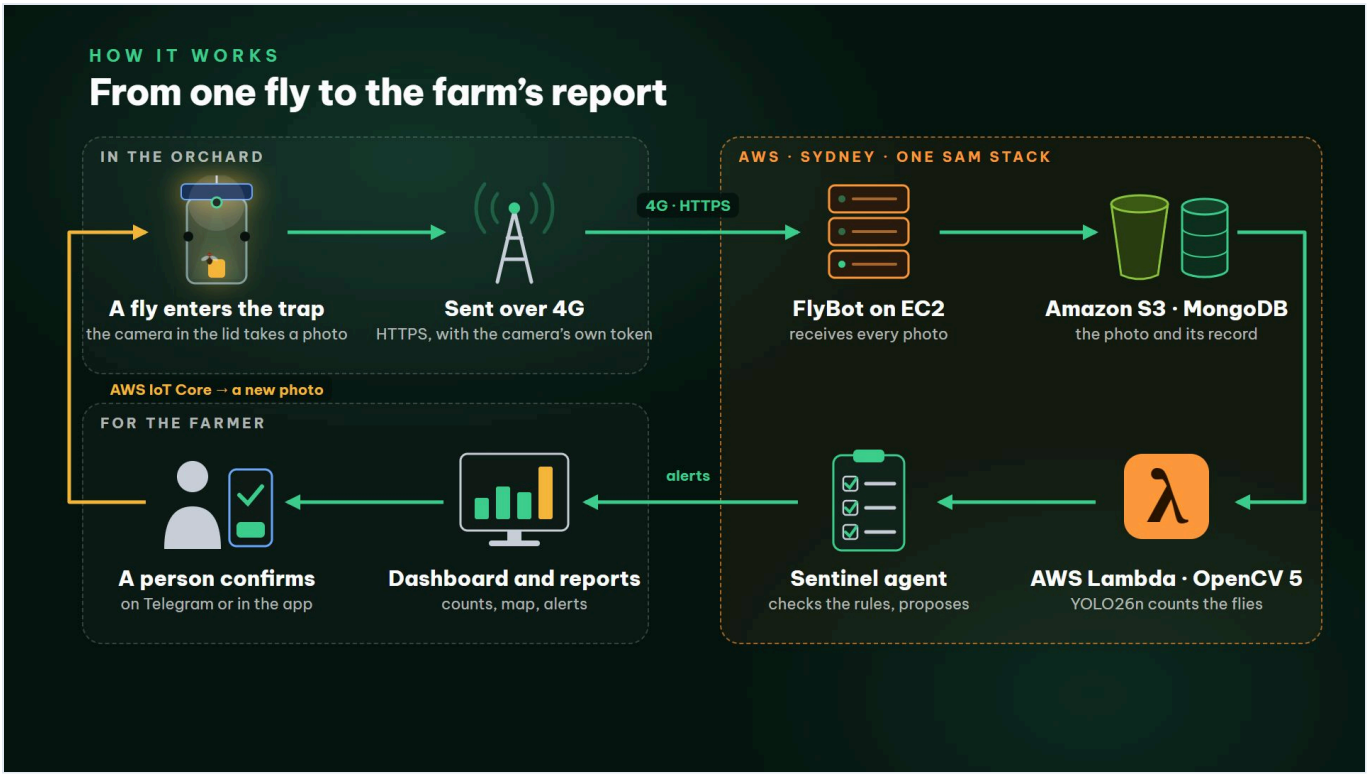
1. Components

Component	Runs on	Code	Job
EZVIZ trap cameras	the farm, EZVIZ cloud	(EZVIZ)	photos of lure traps on FlyBot's schedule, through the EZVIZ Open Platform
CamOS trap camera	ESP32-S3 board in the trap (Wi-Fi or 4G, GPS, solar)	camos-esp32-trap v0.9.2	photos over HTTPS, MQTT to IoT Core (shadow, RPC, presence, log), live view as KVS WebRTC master, OTA updates
FlyBot app	EC2 t4g.micro (Node 22 API, Vue 3 client, Caddy TLS; MongoDB 7 or an external MongoDB)	flybot `feat/aws`	users, farms, cameras, photos, schedules, alerts, chatbot, confirm cards, the Sentinel console, the camera fleet, guest access
Vision Lambda	Lambda arm64 container, 2,048 MB, 120 s	`lambda/vision/sentinel/vision/`	the OpenCV 5 ops on each photo and its previous photo: count, quality, trap, registration, change, crowding, density, occlusion, faces
Agent	the same Lambda invocation	`lambda/vision/sentinel/agent/`	rule planner (every photo) or Claude Haiku 5.5 planner (grower questions), 14 typed tools, guardrails, trace, proposals
Infrastructure	one SAM stack `flybot-<stage>` in ap-southeast-2	`infra/template.yaml`, `deploy.sh`	Lambda, ECR, CodeBuild, EC2, IoT Core, Kinesis Video Streams, S3, Secrets Manager, CloudWatch, SNS, IAM, SSM
External services	outside AWS		EZVIZ Open Platform, Anthropic API (deep mode), Vertex AI (FlyBot's chatbot), Telegram, SMTP, Let's Encrypt

The Lambda is stateless. FlyBot's MongoDB owns cameras, photos, counts, confirm cards and traces; S3 holds the photos.

2. Data flows

2.1 A photo, from capture to decision



The same path in the demo video (https://youtu.be/mfCE2EEFPqg).

1. **Capture.** FlyBot asks an EZVIZ camera for a photo on schedule, or a CamOS camera uploads one over HTTPS with its device token. FlyBot stores the photo in S3 (or an S3-compatible store) and a photo record in MongoDB.
2. **Analysis.** FlyBot calls `Lambda.Invoke` once, synchronously: the photo and the previous photo of the same camera by S3 key (or presigned URL), the camera, its trap kind and platform, its battery, 14 days of counts (`docs/CONTRACTS.md` section 1).
3. **Perception.** The Lambda decodes both photos and runs the OpenCV 5 ops (report section 4): YOLO26n on the new DNN engine (2 x 2 tiles on sticky cards), the quality gate, trap finding, SIFT registration, the change since the last photo, crowding, density, occlusion and a moved camera, YuNet face regions.
4. **Decision.** The agent turns OpenCV's numbers into decisions under guardrails (run states, budgets, quiet hours, battery, evidence-grounded proposals, untrusted-text filter, kill switch). Device tools never reach a camera from the Lambda: an accepted action leaves as a **proposal** (`capture_now`, `set_isp`, `set_interval`, `review_request`, `notify`) with its reason in Vietnamese and English.
5. **Answer.** One JSON response: counts, boxes, quality, diff, density, proposals, trace, privacy regions and timings. The Lambda stores no picture. FlyBot saves the count and the trace with the photo, draws the boxes on its own photo and blurs the returned face regions.

2.2 A proposal, from card to camera (human approval)

1. FlyBot turns each proposal into a **confirm card** in the owner's chat panel, the Sentinel console and Telegram, in the reader's language. Unanswered cards expire after 24 hours without any action.
2. Only a person's **Confirm** runs it: an EZVIZ capture, an IoT Core RPC (`snapshot.now`), or a device-shadow change (capture interval, image settings).
3. A confirmed capture produces a new photo, which goes through flow 2.1 again: the loop closes.

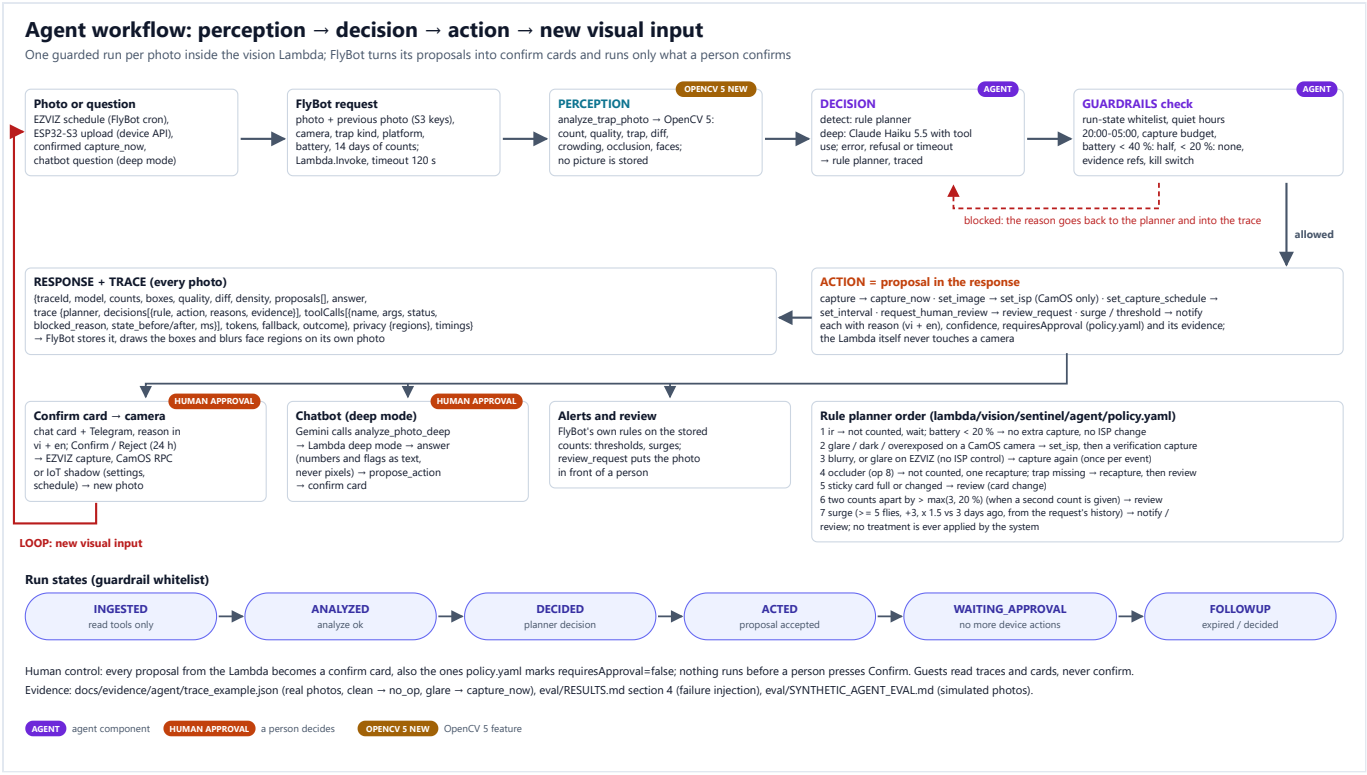


Figure 2. The agent's perception, decision and action steps (`docs/diagrams/agent_workflow.svg`).

2.3 A grower's question

FlyBot's chatbot (Gemini on Vertex AI) calls two tools, `analyze_photo_deep` and `propose_action`. The first calls the same Lambda in `deep` mode: the Claude Haiku 5.5 planner reads OpenCV's numbers as text (never the photo) and answers in markdown; any LLM error falls back to the rule planner and a templated answer. Proposals from a question become confirm cards like any other.

2.4 A CamOS camera: onboarding, live view, fleet

- **Zero-touch registration.** The camera generates its key on first boot; FlyBot signs its CSR through IoT Core, creates the thing `camos-camera` and its KVS channel; the owner claims it by QR code or serial and verify code (`docs/CONTRACTS.md` section 2a). Each certificate may use only its own topics `camos/<serial>/...`.
- **Control.** Shadow (desired and reported settings), RPC (`snapshot.now`, `live.start`, `gps.fix`, `ota.apply`, `log.start`), presence, events.
- **Live view.** Kinesis Video Streams WebRTC: the camera is the master (KVS credentials through an IoT role alias), each browser a viewer with 15-minute credentials for one channel; software H.264 at 320x240, or the camera's own JPEG frames over the data channel (no hardware H.264 on the ESP32-S3). A camera on battery joins at its next wake: a Wi-Fi check-in every 2-10 minutes by day, or its next photo slot on 4G; FlyBot shows it as Sleeping with its wake time and the live page waits for it.
- **Fleet.** Serial registry, firmware releases from GitHub checked against `SHA256SUMS` on alpha, beta and stable channels, rollouts with per-camera progress, a battery guard, pause, a remote console log on `camos/<serial>/log` (24 hours kept) and crash reports.

3. Deployment

Piece	How it ships
Stack	`./deploy.sh` (one command, idempotent): preflight, discovery, models, secrets, the SAM stack, the FlyBot release, the Lambda image, the host bootstrap, health, smoke test and a cold/warm latency measurement
Lambda image	built on ARM by CodeBuild, tagged with a hash of its code and models, stored in ECR; nothing is built on the server
FlyBot release	built on the operator's computer, published as a GitHub pre-release `flybot-<commit>` with its sha256; the server follows a pointer in S3 and switches atomically (the previous release kept for `--rollback`)

Piece	How it ships
Host	Amazon Linux 2023 arm64; `infra/host/bootstrap.sh` run through SSM installs Node 22, Caddy and (without `MONGO_URI`) MongoDB 7 and starts FlyBot under systemd
Firmware	GitHub releases of camos-esp32-trap (web flasher for the first install, OTA afterwards)

The live demo uses FlyBot's existing external MongoDB (`MONGO_URI`) and its S3-compatible photo store outside AWS (`S3_ENDPOINT`): the Lambda reads its photos through presigned URLs (report section 5).

4. Security and privacy

- **Secrets** live in Secrets Manager only; FlyBot reads them at start-up with the instance role; `infra/ops/secrets-scan.sh` checks every commit. IMDSv2 is required; SSH is optional (SSM is used).
- **Least privilege.** The instance role invokes only its Lambda, reads only its buckets and manages only `camos-camera` things and their certificates. One certificate per camera, limited to its own topics. Browsers never see server credentials.
- **People in photos.** No photo is ever sent to an LLM; a photo with a suspected person is not counted; face regions are returned so FlyBot blurs them; the Lambda stores no picture.
- **Guests.** One shared read-only account for two hours; every write is refused except sign-out, chatbot messages, live view of the demo cameras opened to visitors and up to five reports of its own (removed when the session ends); guests read traces and proposals but cannot decide.
- **People decide.** Nothing reaches a camera without a person's Confirm; there is no actuator for treatments.

5. Limits of this architecture

One region and one server, no high availability; MongoDB on the server is backed up only with its disk unless an external `MONGO_URI` is used. The Lambda's guardrails hold within one invocation; across photos the limit is FlyBot's confirm step. The LLM runs on the Anthropic API, not Bedrock (no Bedrock model quota in the account). MCP is not used: the 14 tools are called in-process. Field operation of the ESP32-S3 camera (4G, battery, solar) is not measured (report section 9).