D David Gomez

TECHNICAL & SECURITY ARCHITECTURE

AI DevOps Bot & 24/7 Monitoring

A complete technical breakdown of every resource deployed into your AWS account, exactly how the AI agent connects, and the layered security model that makes it read-only by construction.

🔒 Read-only by design | 🛡️ Defense in depth | 📡 No inbound attack surface | 🪪 Logged in your CloudTrail

DOCUMENT	AUDIENCE
Architecture & Security Brief	Engineering & Security teams

CLASSIFICATION

CONFIDENTIAL

Prepared by **David Gomez** — Fractional CTO & AWS / AI DevOps
services.itsdavidg.co · contactme@itsdavidg.co

Read-only.
Reversible.
Auditable.

OVERVIEW

Contents & Executive Summary

This brief is written for the people who will actually review the integration — your engineers and security reviewers. Nothing here is marketing-speak: every resource, identity, and network path is named.

01 · EXECUTIVE SUMMARY

What we deploy — in one page

The engagement installs three additive capabilities onto your AWS account: **24/7 CloudWatch monitoring with Slack alerting**, an **AI DevOps assistant in your Slack**, and an optional **live architecture diagram**. All three are **read-only**. None of them sit on your application's data path, require a restart, or can mutate a single resource.

0	1	9
inbound ports / public endpoints opened	scoped read-only identity for the agent	independent security boundaries

The core security claim is simple and verifiable: **the agent is read-only by construction, not by good behaviour.** Even if the model were told to do something destructive, three independent layers — a least-privilege identity, an instance-metadata egress block, and a non-privileged OS account — make a write to AWS technically impossible. We don't ask you to trust the prompt; we remove the capability.

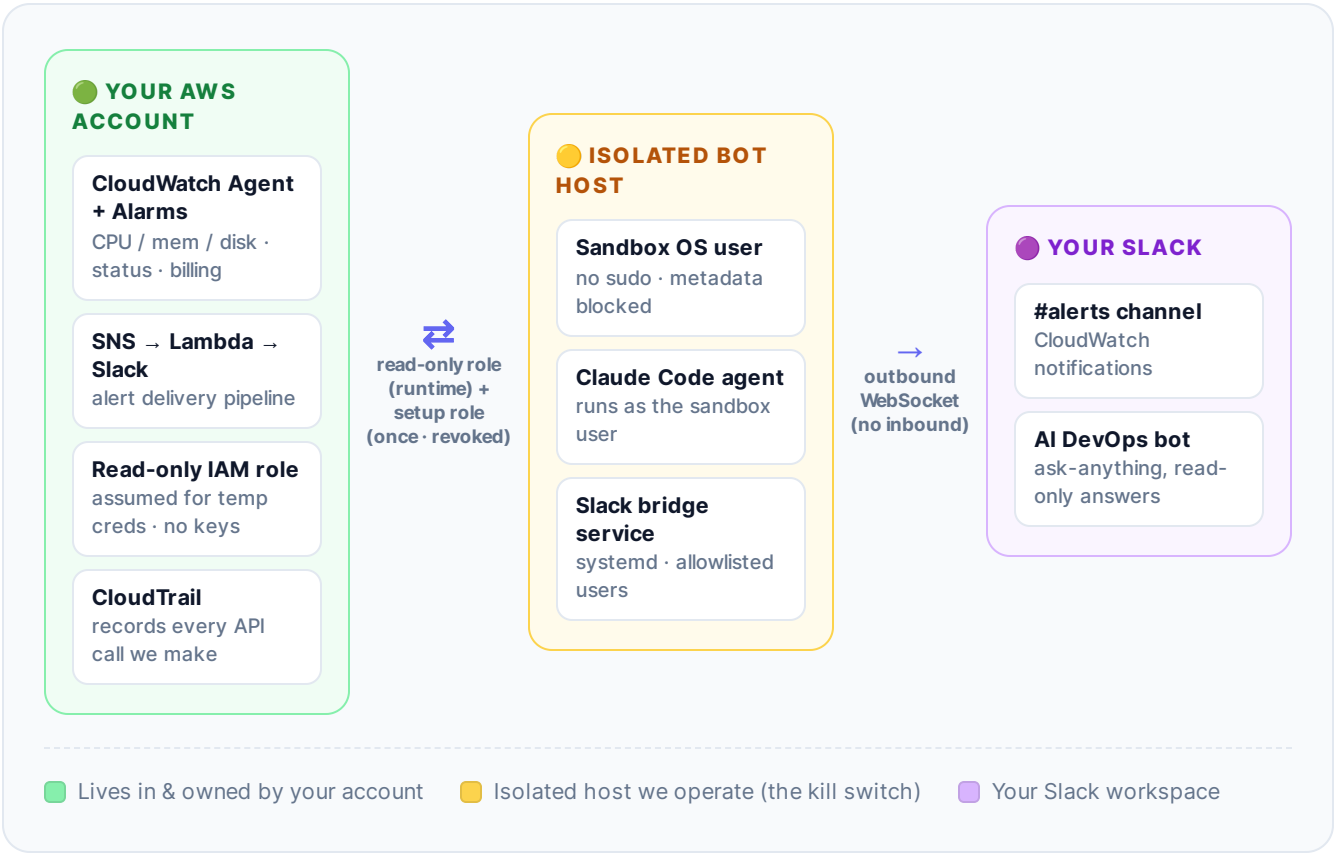
✓ The short version for your security reviewer

Access is a scoped, externally-ID'd, read-only role **you** create and own. The agent **assumes** it for short-lived credentials holding AWS-managed `ReadOnlyAccess` — no write permission exists on it, and **no long-lived AWS key is stored in your account**. Slack connectivity is an **outbound** WebSocket (Socket Mode), so no inbound endpoint is ever exposed. Every action is captured in **your** CloudTrail. You can revoke it all in under a minute.

02 · ARCHITECTURE

A two-plane model that keeps the kill switch in your hands

There are only two systems: **your AWS account** (where monitoring and the read-only identity live) and a small, isolated **bot host** we operate. They meet over a role and an outbound connection — nothing else.



2.1 Two roles — and you grant only what you need

The AI bot is **read-only**. The **only** time write access exists is to **build** monitoring — a separate, scoped role you grant once and we hand back. Two distinct identities:

ROLE	PERMISSIONS	USED FOR / LIFECYCLE
Read-only role `claude-readonly`	AWS-managed `ReadOnlyAccess`, ExternalId-gated. No write action exists on it.	The **bot's runtime** identity. **Permanent** — you delete it whenever.
Setup role `claude-deploy`	Scoped write — CloudWatch, SNS, Lambda, Logs, IAM (only what it creates), ExternalId-gated.	Provisions monitoring **once**; we **revoke it once verified**.

◆ À la carte — the bot stands alone

Want **only** the AI DevOps bot? You grant just the read-only role — the setup role is **never created**, we get no write access, and there's no monthly. Monitoring and the live diagram are **independent add-ons** — pick any combination.

03 · INVENTORY

Everything we deploy — and exactly what it's for

The complete list. Anything that lives in your account, you can see, audit and delete yourself at any time.

RESOURCE	PURPOSE	LIVES IN
CloudWatch Agent `amazon-cloudwatch-agent`	Pushes CPU, memory and disk-used-percent metrics at 60-second resolution. Memory & disk aren't visible to AWS by default — the agent fills that blind spot.	Your account
Tiered metric alarms `CWAgent / AWS/EC2`	Three-tier thresholds (info 60% → warning 80% → critical 95%) on CPU/mem/disk, plus instance `StatusCheckFailed` health alarms.	Your account
Billing alarm `AWS/Billing · EstimatedCharges`	Fires when the estimated monthly bill crosses a threshold you set — cost surprises caught while still small.	Your account
SNS topic `cloudwatch-alerts`	The fan-out hub every alarm publishes to — decouples alarms from delivery so channels can change without touching alarms.	Your account
Alerting Lambda `cloudwatch-to-slack (py3.12)`	Subscribes to the SNS topic, reformats raw alarm JSON into a readable message, posts to your channel via an Incoming Webhook.	Your account
Auth-failure pipeline `/host/auth · host-auth-to-slack`	Streams OS auth logs to a log group; a subscription filter matches failed logins / invalid users / sudo failures and alerts Slack — a gap GuardDuty and console-login alerts **cannot** see.	Your account
Read-only IAM role `claude-readonly-<you>`	The **only** identity the agent uses. Carries AWS-managed `ReadOnlyAccess` only; the agent **assumes** it for short-lived creds — no access key stored in your account.	Your account
Setup (deploy) role `sts:AssumeRole + ExternalId`	Short-lived, you-owned role used **once** to provision monitoring/diagram — **only created if you add them**, and you **revoke it once setup is live**. The bot never uses it.	Your account
Slack app "Claude Ops" `Socket Mode app`	The bot's Slack identity. Socket Mode connects **outbound** — no public request URL for an attacker to find or hit.	Your Slack
Slack bridge service `systemd · claude-slack`	Relays allowlisted Slack messages to the agent and streams answers back, with per-thread conversation continuity.	Bot host
Sandbox OS user `claudero (no sudo)`	The unprivileged Linux account the agent runs as. No sudo, group-scoped reads only, metadata egress firewalled — the containment layer.	Bot host
Architecture diagram `optional · private + auth`	An always-current map of your VPCs, subnets, instances, databases and CDN, behind your own password. Flags idle resources you're paying for.	Bot host

⚠ **Host-level visibility:** account-level monitoring (metrics, billing, GuardDuty, the diagram) works entirely through the read-only role. OS-level signals — CPU/mem/disk and auth logs — need the lightweight CloudWatch agent on the instances you choose to monitor, since no AWS role sees inside a running OS.

04 · ACCESS

How the AI agent actually reaches your AWS

This is the question every security reviewer asks first. The answer is a single, scoped, read-only identity — and three independent things stopping it from ever becoming more than that.

4.1 The credential chain, end to end

- ✓ **You** create a read-only **role** you own, in your own console, from a template we provide. Its trust is gated by a unique **ExternalId**. You own it; you can delete it.
- ✓ It carries **only** AWS-managed `ReadOnlyAccess`. There is no inline policy, no write action, no `iam:*`, no `s3:PutObject` — the role simply has no mutating permission to exercise.
- ✓ The agent **assumes** the role for **short-lived credentials that expire automatically**. There is **no access key** to store, leak, or rotate — and none is ever placed in your account.
- ✓ The agent's AWS calls resolve to that assumed role and **only** that role. We verify this on every deployment with `aws sts get-caller-identity`.

Verified at deploy time — write attempts are denied. Part of the standard checklist; a write returns AccessDenied and metadata is unreachable:

```
# the agent's identity is the ASSUMED read-only role — never an admin role
$ aws sts get-caller-identity --query Arn --output text
arn:aws:sts::<your-acct>:assumed-role/claude-readonly—<you>/claude-ops-bot

# a write is structurally impossible, not just discouraged
$ aws ec2 create-tags --resources i-0abc... --tags Key=x,Value=y
An error occurred (UnauthorizedOperation) … AccessDenied

# the host's own admin credentials are unreachable from the sandbox
$ curl -m3 http://169.254.169.254/latest/meta-data/
000  (connection blocked)
```

4.2 Why it can't escalate

A common attack on cloud agents is to reach the host's **instance metadata service** (IMDS) at `169.254.169.254` and borrow the more-powerful instance role. On the bot host that path is firewalled at the kernel for the sandbox user, so even a fully-compromised agent process cannot trade up from read-only — no sudo, and group-scoped file reads only.

> **✓ The principle**
>
> The agent operates with autonomy **inside** a box whose walls are enforced by IAM and the OS — not by asking the model to please behave. Capability, not intent, is what's constrained — which is what makes autonomous operation safe.

4.3 Identity hygiene

The Anthropic credential powering the agent is a dedicated token belonging only to this bot, independent of every other login. The agent's AWS access is **short-lived assumed-role credentials** — nothing long-lived to rotate; revoke instantly by removing the role's trust. No credential is shared between clients or sessions.

05 · SLACK

The Slack integration — and why there's no open endpoint

Most Slack bots expose a public HTTPS request URL that Slack POSTs events to — an inbound endpoint an attacker can probe. This one doesn't. It uses **Socket Mode**.

5.1 Socket Mode = outbound only

The bridge opens a persistent, authenticated **outbound** WebSocket to Slack and receives events over it. There is no request URL, no inbound port, and no public surface to harden or get wrong. From a network-security standpoint the bot is invisible from the outside.

CONNECTION
Outbound WebSocket (Socket Mode)

INBOUND ENDPOINTS
None — zero public request URLs

TRANSPORT
TLS, Slack-authenticated app token

WHO CAN USE IT
Fail-closed allowlist of Slack user IDs

5.2 Least-privilege Slack scopes

The app requests only what it needs to read mentions and post replies in the channels you invite it to:

SCOPE / SETTING	WHY IT'S NEEDED
`chat:write`	Post the bot's answers and alerts back into the channel.
`app_mentions:read`	See when someone @-mentions the bot to start a thread.
`channels:history` / `groups:history`	Read follow-up replies in a thread the bot already owns, so you don't have to @-mention every message.
`im:history` / `im:write` / `im:read`	Allow direct-message conversations with the bot.
`connections:write` (app token)	Establish the outbound Socket Mode connection.

No `admin`, no `files`, no workspace-wide directory access. The bot only sees messages in channels you deliberately invite it to.

5.3 Access control on the bot itself

- ✓ **Fail-closed allowlist** — only the Slack user IDs you nominate can invoke the agent. Everyone else is silently ignored, by default.
- ✓ **Per-thread isolation** — each Slack thread maps to its own conversation context; threads don't bleed into each other.
- ✓ **It answers, it never acts** — because the underlying AWS identity is read-only, the worst a misdirected message can do is **read** something.

◆ Net effect

An attacker on the public internet has no endpoint to attack. An unauthorized person in Slack gets no response. An authorized person gets a knowledgeable read-only assistant.

06 · SECURITY MODEL

Defense in depth — nine independent boundaries

No single control is trusted to be perfect. For the agent to do harm, **every** one of these would have to fail at once. They are independent by design.

1 Least-privilege assumed role
The role the agent assumes carries only AWS-managed `ReadOnlyAccess`. Writes aren't blocked by a guardrail that could be misconfigured — the permission to write simply doesn't exist on it.

2 Cross-account role with ExternalId
The one-time install role uses a unique `ExternalId` to defeat the confused-deputy class of attack, and is yours to revoke the moment setup is verified. No standing broad privilege.

3 Unprivileged OS sandbox
The agent runs as a dedicated Linux user with `no sudo` and group-scoped reads only. It cannot escalate on the host or read other tenants' secrets.

4 Instance-metadata egress block
A kernel firewall rule blocks the sandbox user from `169.254.169.254`, so the agent can never borrow the host's more-powerful instance role. Read-only stays read-only.

5 No inbound attack surface
Slack connectivity is an `outbound WebSocket`. No public request URL, no open port, nothing on the internet to probe or exploit.

6 Fail-closed authorization
Only an explicit `allowlist` of Slack users can talk to the agent. The default for everyone else is "no response," not "maybe."

06 · SECURITY MODEL (cont.)

The remaining boundaries — and what stays in your hands

7 **Secrets hygiene by default**
No long-lived AWS read key exists in your account at all. The remaining secrets — Slack tokens, the Anthropic token, the alert webhook — live only in `chmod 600` files owned by the sandbox user, or in a Lambda's encrypted environment. None are shared between clients; none are ever placed in chat or tickets.

8 **Short-lived, independent credentials**
The agent's AWS access is `temporary assumed-role` credentials that expire on their own — nothing to rotate. Its Anthropic token is dedicated to this bot and independent of every other login, so no credential change ever breaks another session.

9 **Full auditability**
Every API call the agent makes is recorded in `your CloudTrail` under a clearly-named identity. Access is read-only and reversible, and my open-source AWS audit tool lets you independently verify your account's posture. Nothing is hidden.

The kill switch is yours, and it's instant

You are never locked in and never dependent on us to disconnect. Any one of these ends the agent's access immediately:

- ✓ **Delete the read-only role (or its trust)** — the agent loses all AWS access the instant it can no longer assume it.
- ✓ **Remove the Slack app** — the outbound connection dies and the bot goes silent.
- ✓ **Ask us to stop the bot host** — the agent stops running entirely.

Importantly, your monitoring keeps working regardless — the alarms, SNS topic and alerting Lambda live in **your** account and keep protecting you whether or not the agent is connected.

✓ Alignment with AWS Well-Architected & common frameworks
Least-privilege IAM, no long-lived broad credentials, no public attack surface, full CloudTrail auditability, encrypted secrets at rest, and a clean separation of duties between monitoring and the agent. These map directly to the Security pillar of the AWS Well-Architected Framework and to the access-control and logging controls your SOC 2 / ISO reviewers will look for.

07 · GAPS CLOSED

The security gaps this stack is built to surface

Beyond being safe to install, the point of the engagement is to **find** the open measures already in your environment. These are the blind spots it lights up.

COMMON OPEN GAP	HOW THIS STACK CLOSES IT
Memory & disk exhaustion invisible AWS shows CPU but not RAM/disk by default	The CloudWatch agent surfaces `mem_used_percent` and `disk_used_percent` with tiered alarms — you see the wall before you hit it.
OS-level brute-force unseen SSH / sudo failures leave no CloudTrail trace	Auth logs are streamed and pattern-matched; failed logins, invalid users and sudo failures alert Slack in near-real-time — a gap GuardDuty structurally cannot cover.
Cost surprises the bill is a monthly shock, not a signal	A billing alarm warns you while the overage is small, and the diagram flags idle/forgotten resources you're paying for.
Over-permissioned access admin keys used for routine read tasks	Models the right pattern: a dedicated read-only identity for observation, with write power held only transiently and externally-ID protected.
Dangling DNS / takeover risk records pointing at de-provisioned resources	The read-only audit toolkit and architecture diagram inventory DNS, endpoints and orphaned resources so subdomain-takeover risks are visible.
Silent backup failure backups assumed, never verified	Daily backup-completion checks confirm recovery points actually exist, instead of discovering the gap during an incident.
No single source of truth architecture lives in someone's head	An always-current, interactive diagram of VPCs, subnets, instances, databases and CDN — refreshed on demand, behind your own login.

◆ It's a tool you can run yourself

The same read-only audit that backs this is open source and on PyPI. Your team can run `pip install aws-audit-checklist` and `aws-audit --report` on your own account, right now, before we ever connect — a 30-point security checklist with exact fix commands, and nothing changed.

08 · DATA & EXIT

Data handling & decommissioning

8.1 What data is and isn't touched

- ✓ **Read-only metadata only.** The agent reads AWS configuration and metric data — instance types, security-group rules, alarm states. It is not on your application's data path.
- ✓ **No customer / application data.** Nothing reads your databases' contents, your S3 object payloads, or your application traffic. `ReadOnlyAccess` describes configuration; the engagement never requests object-level data access you don't explicitly approve.
- ✓ **Alerts carry only operational facts** — "CPU at 95% on i-0abc" — never application data.

8.2 Decommissioning

Offboarding is as clean as onboarding. Delete the read-only identity and the Slack app and the agent is gone. The monitoring resources are standard AWS primitives in your account — keep them running, or remove them with a single teardown; either way they're yours.

8.3 Offboarding leaves nothing behind you don't want

No proprietary agent stays resident in your account — the monitoring is standard AWS primitives (alarms, SNS, a Lambda) you can keep or delete. The bot's read-only role and the Slack app are the only things tied to us, and both are yours to remove in seconds.

Have your security team review it first — please

We'd rather you did. My read-only AWS audit tool is public and MIT-licensed — run it in a sandbox account to see the rigor first-hand. The deployment automation itself is proprietary, but it operates strictly read-only and every action lands in your CloudTrail, so your team can verify exactly what it did — and we'll walk them through it before we begin.

09 · ONBOARDING

Onboarding — the exact steps & what changes hands

From "yes" to live in 2–3 days — the precise sequence and exactly what's exchanged. The one thing your team is **never** asked for: credentials.

1 **Kickoff & scope** · Day 0, 30 min
A short call. You choose: the **AI bot alone**, or add monitoring and/or the live diagram. Nothing is bundled.

2 **I send you** · a template + two values
After our call, a **ready-to-run** `CloudFormation` template with your two values **already filled in** — nothing to assemble, you just deploy it. Adding monitoring? A second template too.

3 **You deploy it** · your console, ~2 clicks
The template creates the **read-only role** for the bot (and the **scoped setup role** if you add monitoring). You own both and can delete either anytime.

4 **You send back** · identifiers only
Just the **role ARN(s)** it outputs, your **region**, and — for monitoring — which **instances** to watch. **No keys, no passwords — not even Slack; I run that on my side.**

5 **I deploy & you go live** · Day 1–2
On your dedicated runner I wire it up, then invite your team to a **private Slack channel I run** — that's where the bot answers and alerts flow. You also get the diagram's private URL + login. For monitoring, once verified **you delete the setup-role stack** — no standing write access remains.

The exchange at a glance

↑ YOU SEND ME (IDENTIFIERS ONLY)	↓ I SEND YOU	✗ YOU NEVER SEND
The read-only role ARN (+ setup-role ARN if monitoring)	The CloudFormation template(s)	AWS access keys
Your AWS region	My runner ARN + a one-time ExternalId (pre-filled)	Admin / root credentials
(monitoring) which instances to watch	A private Slack channel invite (I host Slack)	Console passwords
…nothing else — no Slack, no secrets	(at go-live) the diagram's private URL + login	Any secret — even Slack is mine

Ready to see your AWS account clearly?

Book a free 30-minute kickoff — we'll scope exactly what goes in, read-only from the first minute, no charge before work is done.

Book your kickoff call →

services.itsdavidg.co · contactme@itsdavidg.co · github.com/davidgomezbravo/aws-audit