CLOUDLENS · NETWORK VISIBILITY

CloudLens on AWS
Full-Stack Visibility, End to End

From CloudFormation stack creation to a proven,
packet-verified VPC Traffic Mirroring pipeline

One command. A few basic prompts.
Everything else automated, start to finish.

Prepared by **Brine-Ndam Ketum**
Systems Engineer, Keysight Technologies

Contents

1. Executive summary

The headline is the automation. A complete Keysight CloudLens network-visibility pipeline stands itself up on AWS from a single command, asking only a few basic prompts along the way. From an empty account to production workload traffic copied, brokered, and delivered to tools with no agent in the data path, every hop verified with real packets, the operator types one line and answers a handful of questions. Nothing is clicked by hand in the AWS console, and nothing is wired by hand in the CloudLens APIs.

One command. Basic prompts. Everything else automated.

```
curl -sSL https://raw.githubusercontent.com/Keysight-Tech/cloudlens-ansible-aws/main/
deploy/deploy-stack.sh | bash
```

That one line runs all seventeen phases below. The only things it asks a human for are the activation codes at licensing and a few yes/no choices (deploy KVO or not, create test workloads or bring your own, add the mirror session). It creates the stack, waits for the appliances to initialise, licenses and adopts them, installs the Linux and Windows sensors, builds the agentless AWS mirror fabric, wires the vPB traffic path, and stands up a capture host, then hands back the logins so the operator can watch. It is resumable: stop it anywhere and re-run, and it continues from where it left off. It runs the same in a laptop shell or in AWS CloudShell, from a clean account or a partial one.

Everything after this point is the detail behind that one command: what each phase does, what objects it builds, the bugs that were found and fixed to make it reliable, and the live evidence that the tapped traffic actually arrives.

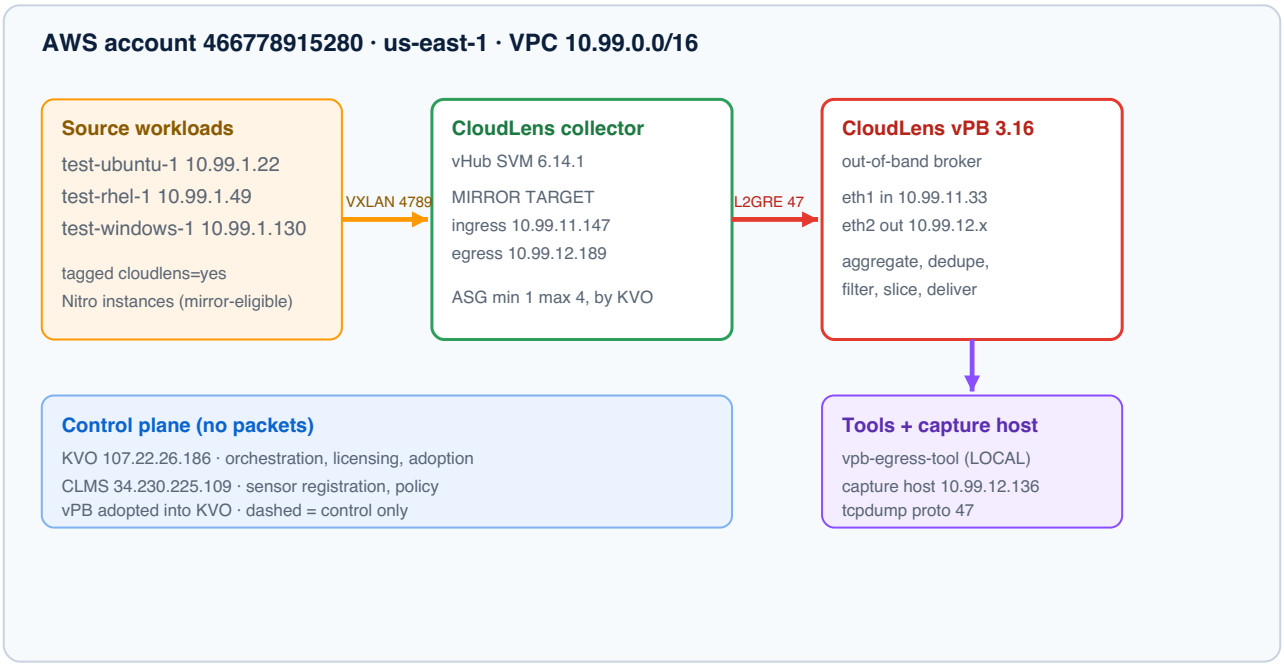
Proven end to end

Workload traffic is mirrored agentlessly from the source ENIs, aggregated by a CloudLens collector, brokered by a virtual Packet Broker (vPB), and delivered to a capture host where **42,010 GRE-encapsulated packets** were read with the original workload conversations intact inside each tunnel. Linux and Windows sensors both registered. The whole path was traced with AWS CloudWatch counters and live `tcpdump`.

Component	Role	Status
Keysight Vision Orchestrator (KVO)	Orchestrates the estate: licensing, adoption, cloud config, mirror fabric	live
CloudLens Manager (CLMS / vController 6.14.1)	Sensor control plane; serves the Windows installer	connected
Virtual Packet Broker (vPB 3.16)	Aggregate, deduplicate, filter, slice, deliver	adopted
CloudLens collector (vHub SVM)	Mirror target; re-encapsulates VXLAN to L2GRE	online
3 traffic mirror sessions	One per Nitro source ENI (ubuntu, rhel, windows)	active
Capture host (tool receiver)	Proves delivery: tcpdump on the tapped copy	capturing

2. Architecture at a glance

CloudLens on AWS uses two capture methods that feed one broker. **Host sensors** run inside the instance and see traffic before the ENI, including container and pre-encryption traffic. **VPC Traffic Mirroring** is agentless, for instances where no agent can be installed. Both encapsulate copies to the vPB. The vPB is never inline: if it fails, the application keeps serving.



Two capture methods, one broker. The vPB receives copies only and is never in the production path.

3. The deployment sequence, phase by phase

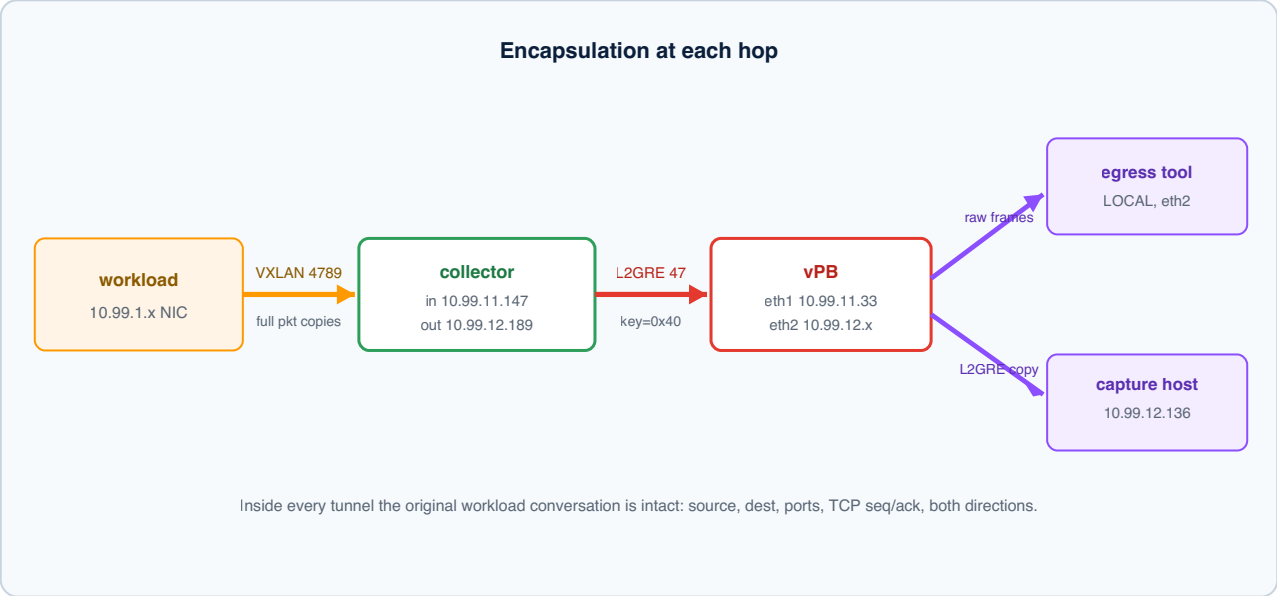
This is what the one command does for the operator, unattended, in order. A single orchestrator, `deploy/deploy-stack.sh`, runs all seventeen phases. It is resumable: each phase detects whether its work already exists and skips it, so a run that stops can be re-run and picks up where it left off. Only the phases marked prompt ever stop for input, and even those accept defaults or command-line flags for a fully unattended run.

#	Phase	What it does	Why it matters
1-5	Preflight + stack	Environment checks, marketplace AMI check, CloudFormation deploy of the VPC, subnets, KVO, vController, vPB	Greenfield infrastructure in one template
6	Deploy the stack	Creates VPC 10.99.0.0/16 with three subnets: mgmt 10.99.1, data/ingress 10.99.11, tool/egress 10.99.12	The dual-plane layout the vPB needs
7	Wait for vController	Polls the real login route until the appliance answers	The appliance needs ~15 min to initialise
8	vPB bootstrap	Waits for KCOS, drives the vPB CLI (EULA, context)	The vPB must accept its EULA before adoption
9	Project key + login	Rotates the default password, provisions the working UI login	Returns credentials so the operator can watch
10-11	Sensor mode + licensing prompt	Accepts the KVO EULA, activates licences via REST. Asks for the activation codes and how many to apply.	KVO is the licence authority for the whole fabric
12	CLMS adoption + Cloud Config	Adopts the manager into KVO; the Cloud Config provisions the project and its key	One manager owns both agent and agentless visibility
13	Sensors	Installs and registers Linux (docker/podman) and Windows sensors	In-guest, pre-encryption visibility
14	Adopt the vPB	Brings the vPB into KVO as a managed device	The broker becomes orchestrable
15	AWS mirror	Presence, Cloud Config (Aws), collection, collector ASG, tool, policy, mirror sessions	Agentless VPC Traffic Mirroring
16	vPB traffic path	C2DL binds the collection to vPB eth1; egress bound to the tool; monitoring policy	Must run after 15: needs the AWS cloud config
17	Capture host	Launches a tcpdump host on the egress subnet, wires it as a REMOTE tool	Makes the tapped traffic visible

Resumability, concretely The orchestrator refuses to store secrets in its state file, detects an already-CONNECTED CLMS and an already-adopted vPB, refreshes a stale helper clone, and lets an operator bring their own workloads by passing a subnet and tag instead of creating throwaway VMs. The goal is a run that is smooth from a clean account or resumed from any interruption, in a laptop shell or AWS CloudShell.

4. The data plane: what rides each hop

The single most important thing to understand is what is carried at each step, because the tap is a chain of encapsulations. A copy of the workload packet is wrapped, moved, unwrapped, processed, and re-wrapped several times before a tool ever sees it.

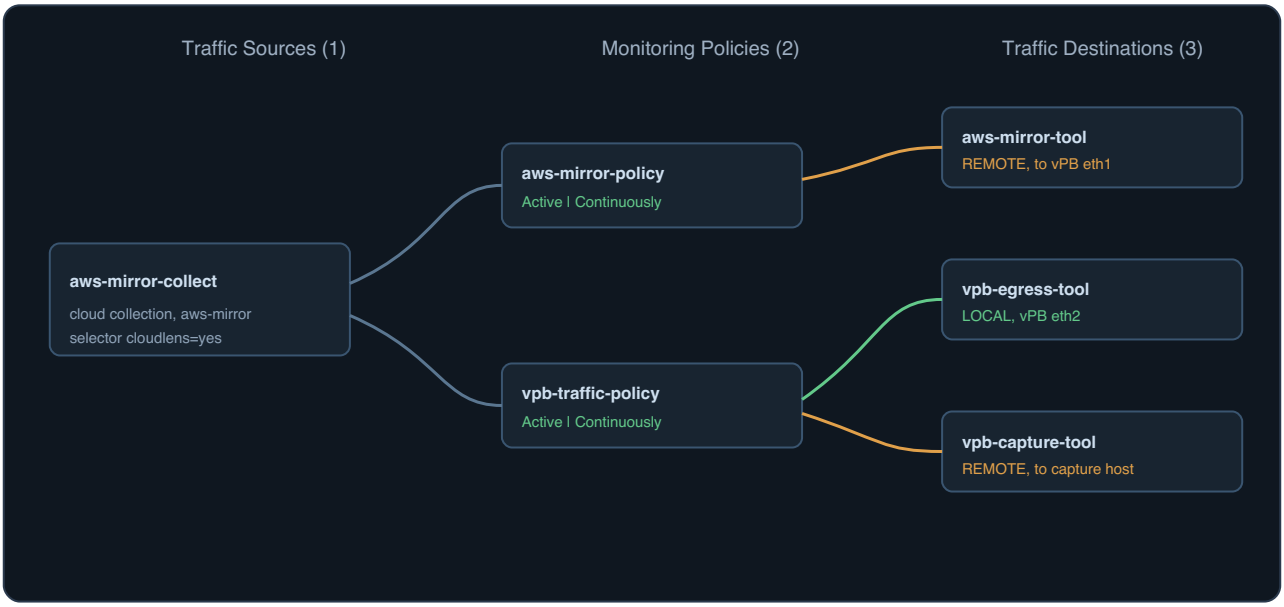


The mirror target is the collector, not the vPB. VXLAN carries the copy to the collector; L2GRE carries it onward.

Interface	In	Out	Encapsulation
Source ENI (workload)	-	AWS copies every packet, both directions	plain, then wrapped by the mirror session
Collector ingress 10.99.11.147	mirrored traffic (this ENI is the target)	-	VXLAN UDP/4789
Collector egress 10.99.12.189	-	re-encapsulated toward the vPB and capture host	L2GRE, IP protocol 47
Collector mgmt 10.99.1.153	KVO control	KVO control	none (the GRE tunnel is tracked from this IP)
vPB eth1 ingress 10.99.11.33	L2GRE from collector, via the C2DL; decapsulated here	-	L2GRE 47
vPB eth2 egress 10.99.12.x	-	processed copies to tools	LOCAL raw + REMOTE L2GRE to capture host
Capture host 10.99.12.136	L2GRE copy; tcpdump shows inner packets	-	L2GRE 47, src/dst check OFF

5. The KVO Visibility Fabric

KVO models the tap as objects that connect a **source** through a **monitoring policy** to a **destination**. The automation builds all of them. Below is exactly what exists in the running system, matching the Monitoring Policies and Visibility Fabric views in the KVO console.



The live KVO fabric: one source, two policies, three destinations. This mirrors the console's Monitoring Policies view.

Every step explained, for a first reader

The deploy prints a list of seven things it is about to build: the AWS presence, the Cloud Config, the collection, the collector Service VM, the tool, the monitoring policy, and the traffic mirror sessions. Those are KVO's own words, and none of them mean anything on first reading. This section explains each one in plain terms, in the order the deploy creates them, because the order is the reason the whole thing works.

The goal in one sentence: **copy the network traffic of ordinary EC2 machines and deliver that copy to a monitoring tool, without installing anything on those machines.** That is what "agentless" means here. AWS itself can duplicate an instance's traffic, a feature called VPC Traffic Mirroring. Everything below exists to tell AWS which machines to copy and where to send the copy, and to give the copy somewhere to land.

1. The AWS presence: giving KVO the keys

KVO is the orchestrator, but it lives outside your AWS account and has no rights in it. The AWS presence is where you hand KVO an AWS access key so it can act for you: create mirror sessions, launch instances, and so on. Think of it as the login KVO uses to talk to AWS.

> The key must carry the CloudLens Zone Tapping permissions. A key with no policy still creates the presence successfully, and then nothing else works: no collector appears, no sessions are cut, and the only clue is a KVO alert about checking credentials. A silent, valid looking presence with a powerless key is the single most confusing failure in this build.

2. The Cloud Config: describing the AWS environment

The Cloud Config tells KVO the shape of the network it is working in: which region and availability zone, which three subnets and three security groups the collector's interfaces will use, which SSH key pair, and

where the CloudLens Manager is. KVO needs this because it is about to launch a machine into your VPC and must know exactly where to put it.

The three security groups are separate on purpose, one per collector interface: management, ingress, egress. The schema requires all three; there is no single-group shortcut.

3. The Cloud Collection: choosing what to tap

The collection answers "which machines do we want to watch?" You express it as a tag rule, for example `cloudlens = yes`. Any instance in this VPC carrying that tag becomes a source. In the KVO console this is the Workload Selector, and the console lists the matched network interfaces so you can confirm it found what you expected.

This is the *source* half of the picture. Nothing here says where the traffic goes.

4. The collector Service VM: where the copies are sent

AWS traffic mirroring needs a destination inside the VPC, called a mirror target. That target is the collector: a CloudLens appliance KVO launches for you into the subnets from step 2, behind an auto scaling group. Every mirrored packet from every tapped machine arrives here first.

> The collector reads its configuration once, when it boots, and does not go back for it. If it starts before the configuration is finished, it runs with an incomplete picture forever and quietly creates nothing. Rebooting does not help, because the machine keeps its state. Only a brand new instance re-reads the configuration, which is why the automation deliberately terminates the collector after the fabric is complete and lets the scaling group replace it. It is also why the collector can do nothing at all until tagged machines exist in its own VPC: with no sources, a perfectly healthy collector sits idle and produces zero sessions.

5. The Tool: where the traffic finally goes

A Tool is a destination for the collected traffic, and it is the *other* half of the picture. Here the traffic is sent onward to the virtual Packet Broker over an L2GRE tunnel, which wraps each mirrored packet inside another packet so it can cross the network intact. A second Tool can point at a plain capture host, which is how the traffic is proved with tcpdump.

Tools come in two kinds, and mixing them up stops the build. A **REMOTE** tool is an address somewhere on the network. A **LOCAL** tool is a physical port on a device KVO manages. Only a LOCAL tool can be bound to a vPB port.

6. The Monitoring Policy: connecting source to destination

Nothing so far links the two halves. The collection knows what to tap; the tool knows where traffic should land; the policy is the sentence that joins them: *send this source to that destination*. Committing the policy is the moment KVO has a complete instruction and can act.

One policy can have several destinations, which is how the same tapped traffic reaches both the vPB and a capture host at once.

7. The traffic mirror sessions: the actual tap

A session is the real AWS object that copies one network interface's traffic to the target. One session per tapped interface. These are not created by the deploy script: the collector creates them, once it can see a complete policy and the tagged interfaces. That is why they appear minutes after the deploy says it has

finished, and why counting them is the only honest proof that the pipeline works. Everything before this point is configuration; this is traffic.

And the C2DL, if a vPB is involved

The Cloud to Device Link, shown in the console as "Connect To Device", attaches the cloud side of the fabric to a physical port on the vPB. Without it the path has a gap: the policy commits against an incomplete route and no sessions are cut. It has to be attached to the AWS Cloud Config before the collector boots, or the collector must be replaced afterwards.

Reading the result

Two commands tell you whether it worked, and they are worth more than any status message:

```
aws ec2 describe-traffic-mirror-sessions --region us-east-1
aws ec2 describe-traffic-mirror-targets  --region us-east-1
```

One session per tagged machine means the tap is live. Zero sessions with everything else present is the normal failure, and it almost always means one of three things: the collector booted before the configuration was complete, there are no tagged machines in this VPC, or the IAM key in the presence has no permissions.

The three security groups, the port roles, and the termination IP

Three details in this build reject the configuration outright if they are wrong, and none of them explain themselves: why the collector needs three separate security groups, what "ingress" and "egress" mean on a vPB port, and what a termination IP is for. The rules below are from the *Keysight Vision Orchestrator 3.0.1 User Guide*, AWS Cloud Configs and Cloud to Device Links.

Why three security groups, not one

The collector (the vHub) is built with three network interfaces, each doing a different job, so each gets its own security group:

Group	Interface job	What must be open
Management	How the collector talks to the CloudLens vController and KVO. This is also the group that decides which VPC the deployment belongs to.	`443` for vController to KVO and vController to vHub, and `22` for SSH.
Ingress	Where mirrored traffic arrives from the tapped machines.	The GRE tunnel traffic from the mirror sources.
Egress	Where the collector sends traffic onward, to the vPB or another tool.	The outbound tunnel to the tool address.

> The guide is explicit and this is the part most often got wrong: the Management, Ingress and Egress security groups **have to be in separate subnets, with separate IPs, for the vHub to work properly**. Three groups pointing into one subnet is not a valid deployment. The schema also marks all three as required, so there is no single-group shortcut: sending them as optional is rejected before anything is created.

Keeping them apart is not bureaucracy. Mirrored traffic can be a large multiple of normal traffic, and if it shared the management interface it would compete with the very control channel KVO uses to manage the collector. Separating them means a flood of tapped traffic cannot cost you control of the appliance.

Ingress and egress on the vPB: which port does what

The same two words appear again on the vPB itself, meaning something related but distinct. The vPB has three interfaces and the roles are fixed by convention in this build:

Port	Role	Carries
`eth0`	Management	SSH and the KVO management channel. Never carries tapped traffic.
`eth1`	**Ingress**, bound to the C2DL	Mirrored traffic arriving from the collector, still wrapped in GRE.
`eth2`	**Egress**, bound to the LOCAL tool	Traffic leaving the vPB towards the monitoring tool, after any processing.

Read in one direction it is simple: traffic comes *in* on eth1 and goes *out* on eth2. The binding is what makes it real. eth1 is bound to the Cloud to Device Link, which is the cloud side arriving. eth2 is bound to a Tool, which is the destination leaving. A REMOTE tool cannot be bound to a device port at all, which is why the egress tool has to be LOCAL.

> These data ports must already be up as DPDK data ports on the vPB. That bring-up is not automated, so on a brand new vPB the device configuration has no ports and there is nothing for the bind step to attach to. If the port-bind steps themselves fail, this is why.

The termination IP: the address the tunnel actually ends at

Mirrored traffic does not arrive as bare packets. It arrives wrapped in a GRE tunnel, addressed to something. The termination IP is that address: the IP on the vPB's ingress interface where the tunnel ends and the outer wrapper is removed, so the original packet can be read. In this build it is the vPB's ingress address, and it is the same value the collector's tool points at.

It is not optional. The guide states that a remote collection point can be bound to ports according to the number of configured termination IPs, and that **if no termination IP is configured the collection point cannot be bound to any port at all**, with an error saying a termination IP must be configured first. Without it the path has no endpoint and the bind is refused.

Two constraints apply specifically to the Cloud to Device Link, and both are fixed rather than chosen:

- **GRE only.** It is the only protocol a Cloud to Device Link supports.
- **A 4 byte key.** The only key size supported for a Cloud to Device Link. Where remote collection points and Cloud to Device Links are bound to the same device, both the protocol and the key size have to match.

This is why the L2GRE key appears in the packet capture, and why seeing `GREv0, key=0x40` on the capture host is meaningful: it is the tunnel arriving at its termination address with the key the fabric was configured with, carrying the original packet inside.

The objects, and why each one mattered

Object	What it is	Note

AWS Presence `aws-mirror`	KVO project holding the Zone-Tapping IAM keys	-
Cloud Config `aws-mirror` (Aws)	Holds the awsConfiguration: three distinct security groups, the three subnets, the vController IP, the SSH key pair. Provisions and scales the collector ASG.	The three SG fields are non-null in the schema. Sending them as optional was bug 1.
Cloud Collection `aws-mirror-collect`	The traffic source: the mirrored workload traffic. Workload selector field=cloudlens, regex=yes.	The selector field must be the tag key, not the literal word "tag". That was bug 4.
Tool `aws-mirror-tool` (REMOTE)	L2GRE to the vPB ingress 10.99.11.33. reachableFrom CLOUD_CONFIG, vlanStripping false.	Carries the mirrored copy from collector to vPB.
Tool `vpb-egress-tool` (LOCAL)	Points at the vPB's own eth2.	A REMOTE tool cannot bind a device port, so this must stay LOCAL.
Tool `vpb-capture-tool` (REMOTE)	L2GRE to the capture host 10.99.12.136.	Added so the tapped traffic is visible to tcpdump. That was bug 6.
C2DL `vpb-c2dl`	Cloud to Device Link: binds the collection to vPB eth1 ingress.	Must be associated to the cloud config via deviceLinks. That step was refused until the CloudLens vPB licence was activated: bug 3.
Policy `aws-mirror-policy`	Sends aws-mirror-collect to aws-mirror-tool.	-
Policy `vpb-traffic-policy`	Sends aws-mirror-collect to both vpb-egress-tool and vpb-capture-tool.	One policy, two destinations.

6. The six bugs, each hiding the next

The pipeline had been believed blocked by the product. It was not. Six distinct defects, all in the automation, stacked so that fixing one only revealed the next. This is the debugging record, because the lessons are more valuable than the fixes.

#	Symptom	Root cause	Fix (commit)
1	createCloudConfig(Aws) rejected before anything ran	mgmt/ingress/egress security-group IDs sent as optional; the schema marks them non-null	send all three (4809c20)
2	Traffic-path step 6 failed on a missing awsConfiguration	The path was wired against the custom (sensor) cloud config, which has none. It must target the AWS cloud config the mirror phase creates.	run mirror before path (1054322)
3	deviceLinks step refused; change request stuck		activate CloudLens

	InProgress; our poll waited 600s in silence	"No CloudLens vPB licence installed." A refused change request never moves to Failed, and the reason is only on the KVO Alerts page, not the GraphQL status field.	codes; warn on stuck CR (e59244b)
4	Cloud collection matched 0 of 8 interfaces	Workload selector sent field="tag" (the literal word) instead of the tag key	field = tag key (b301357)
5	Collector forwarded 135 MB, vPB received nothing; sessions existed	The vPB security group allowed VXLAN UDP/4789 but never IP protocol 47 (GRE). Every mirrored packet was dropped silently. The only symptom was a KVO alert: "Tunnel of type GRE: Remote destination not reachable." The tunnel source is the collector management IP, so the rule had to allow the mgmt/VPC CIDR, not just the data subnet.	allow proto 47 (e5a7cbc)
6	A capture host on the egress subnet saw nothing, even though tapping worked	The vPB egress tool is LOCAL: the vPB emits raw frames onto the egress subnet, which AWS will not deliver to another instance. A capture host needs its own REMOTE tool tunnelling a copy, and its ENI needs source/dest check off.	REMOTE capture tool + capture host (dc892d9, 952c583)

The lesson worth keeping Read which sub-step actually stopped before accepting any inherited explanation, and read the KVO Alerts page: it named the licence refusal and the GRE drop precisely while the GraphQL status fields said nothing. A silently dropped tunnel looks exactly like a working sender, so always measure both ends of every hop.

7. Live evidence: the tap actually works

Configuration is not proof. Traffic was generated on the workloads and measured at every hop with AWS CloudWatch, then read packet by packet on the capture host.

CloudWatch counters, idle versus under load

Measurement point	Idle	Under load
Collector NetworkIn	28,874 B / 66 pkt	105,535,186 B / 83,934 pkt
Collector NetworkOut	30,915 B	135,788,560 B
vPB NetworkIn (before the proto-47 fix)	358 pkt	358 pkt (nothing arrived)
vPB NetworkIn (after the proto-47 fix)	358 pkt	144,560 pkt / 154 MB
Capture host (tcpdump)	0 GRE	42,010 GRE packets

The mirror sessions AWS created

```
tms-01e434cd32b09c306   eni-0eaa0948e2db6f904 (test-rhel-1)     -> tmt-06c388c226e1dfbbd
tms-00f8048896d0d7efa   eni-0b16a8fb1fce64343 (test-windows-1)  -> tmt-06c388c226e1dfbbd
tms-003b9e9c7576d4157   eni-0ca949f29c73b5e41 (test-ubuntu-1)   -> tmt-06c388c226e1dfbbd
   target tmt-06c388c226e1dfbbd = collector ingress ENI eni-06bfded70a1d8a420 (10.99.11.147)
```

The proof, one packet, read on the capture host

```
10.99.12.189 > 10.99.12.136: GREv0, key=0x40, length 368       <- OUTER: the tunnel
    10.99.1.22.41636 > 34.230.225.109.443: Flags [P.], seq ..., length 294   <- INNER: the real
traffic
```

What this shows

The outer header is the collector egress (10.99.12.189) tunnelling to the capture host (10.99.12.136) over L2GRE key 0x40. The inner header is test-ubuntu-1 (10.99.1.22) talking HTTPS to the vController, with real TCP sequence numbers, correct checksums, and both directions present. Nothing is synthesized; this is the workload's genuine conversation, copied at the source and delivered to the tool.

8. Windows: agentless and by-sensor, both proven

Windows visibility was proven two independent ways.

Agentless

test-windows-1 (10.99.1.130) has its own traffic mirror session, so its traffic reaches the capture host with no software installed on the VM at all. VPC Traffic Mirroring needs only a Nitro source.

By sensor

The Windows sensor also registered for the first time. The root cause of every prior failure was found: nothing was ever downloading the installer. The CLMS serves it at a predictable static path, so the script now auto-discovers the URL. The live install on Server 2022 returned exit code 0, the CloudLens service ran, agent.yml was written, and the sensor entered a healthy control loop with the CLMS (pulling config, posting state). Commit 0dbc054.

9. Access and credentials

System	Address	Login
KVO (Vision Orchestrator)	https://<kvo-ip>	admin / admin (Keycloak)
CloudLens Manager (CLMS)	https://<vcontroller-ip>/cloudlens/login	admin / (changed on first use)
vPB management	ssh admin@<vpb-ip>	SSH mgmt, vPB CLI
Capture host	ssh -i <key>.pem ubuntu@<capture-host-ip>	then sudo tcpdump -i any -nn 'proto 47' -v

All appliance logins are changed on first use, and KVO gates on an EULA at first boot. IPs shown in this document are the lab's values for illustration; treat every credential as secret.

10. Reproduce it: one command

The entire pipeline above runs from a single command, resumable, in a laptop shell or AWS CloudShell. It creates the stack, waits for the appliances, licenses and adopts, installs the sensors, adopts the vPB, builds the mirror fabric, wires the traffic path, and stands up the capture host, prompting for the activation codes and offering the mirror session along the way.

```
curl -sSL https://raw.githubusercontent.com/Keysight-Tech/cloudlens-ansible-aws/main/deploy/
deploy-stack.sh | bash
```

Windows is now hands-off Run with `--test-vms all` and the script auto-discovers the Windows installer URL from the CLMS, so the Windows sensor installs with no manual input. Bring-your-own workloads by passing a subnet and tag.

Appendix A. Complete resource inventory (live)

Every resource in the running system, so the topology can be reconstructed exactly.

Instances

Name	Instance ID	Type	Public	Private
cloudlens-stack-kvo	i-0714545a1fbfb334a	c5.2xlarge	107.22.26.186	10.99.1.202
cloudlens-stack-vcontroller (CLMS)	i-09d72aa9228ed11f2	t3.xlarge	34.230.225.109	10.99.1.64
cloudlens-stack-vpb	i-0810a81b2c8314762	t3.xlarge	54.85.23.44	10.99.1.27
collector (vHub SVM)	i-09a162651864789c6	t3.xlarge	100.53.55.19	10.99.1.153
capture host (tool receiver)	i-00980741f9a358c3b	t3.medium	13.220.38.142	10.99.12.136
test-ubuntu-1	i-0edafb3148290e1bd	t3.small	-	10.99.1.22
test-rhel-1	i-0fc878e690b421360	t3.small	-	10.99.1.49
test-windows-1	i-06e7e2a9e36cc3dee	t3.medium	-	10.99.1.130

Network

Object	ID	CIDR / detail
VPC	vpc-04a810b44b6e7e601	10.99.0.0/16
mgmt subnet	subnet-01122642251c46eeb	10.99.1.0/24
data / ingress subnet	subnet-0342818bbc3283327	10.99.11.0/24
tool / egress subnet	subnet-0df0120ae6b83562d	10.99.12.0/24
vPB SG	sg-07c33cc3887a67440	

		22, 9022, 443, VXLAN 4789, 10800-10801, GRE proto 47
collector mgmt/ ingress/egress SGs	sg-07c1f23d65b70788e / sg-09cfd109dcf74b23d / sg-04abbcb197518da26	three distinct, as the schema requires
capture host SG	sg-0ab4afdfa88c71e04	GRE 47, VXLAN 4789/10800-10801 from VPC, SSH from admin

Mirroring objects

Object	ID	Detail
Mirror target	tmt-06c388c226e1dfbbd	points at collector ingress ENI eni-06bfded70a1d8a420 (10.99.11.147)
Mirror filters	tmf-08755de95f278d238, tmf-0ca4bdb9755ba9e5d	ingress + egress rules
Mirror sessions	tms-01e4..., tms-00f8..., tms-003b...	one per source ENI (rhel, windows, ubuntu)
vPB ENIs	eth0 10.99.1.27 · eth1 10.99.11.33 · eth2 10.99.12.94	mgmt · ingress · egress; SourceDestCheck off on data ENIs

Appendix B. What each phase actually runs

The orchestrator invokes these helper scripts. Each is idempotent by name or state.

Script	Responsibility
deploy/deploy-stack.sh	The 17-phase orchestrator: preflight, stack, waits, licensing, adoption, sensors, vPB, mirror, path, capture host, summary. Resumable via probe/detect/resume_decide.
deploy/cloudformation/ stack.yaml	VPC, three subnets, IGW, routes, SGs (now including GRE proto 47), KVO, vController, vPB.
scripts/kvo_license.py	Accepts the KVO EULA, activates codes via REST /api/v2/licensing, reports coverage. Prompts for code + quantity; no codes are hard-coded.
scripts/ kvo_aws_mirror.py	Builds the AWS presence, Cloud Config (Aws) with three non-null SGs, collection with the correct selector, collector ASG (min 1 max 4), REMOTE tool, monitoring policy, and optionally the capture tool.
scripts/vpb_wire_path.py	syncDeviceConfigPorts, C2DL, bind eth1 ingress, LOCAL tool, bind eth2 egress, deviceLinks, monitoring policy. Adds the REMOTE capture tool to the egress policy when --capture-ip is given.
scripts/deploy-test-workload-vms.sh	Optional Ubuntu/RHEL/Windows workloads, tagged for discovery, with the SSM instance profile.
playbooks/windows.yaml	Windows sensor over SSM: asserts an installer source, downloads from the CLMS or a local file, silent-installs, waits for the service.

Appendix C. Windows sensor, the complete procedure

The Windows sensor is a native executable, unlike the Linux container. The root cause of every prior failure was that nothing fetched the installer. The full working procedure, proven live on Server 2022:

1. **Discover the installer URL.** The CLMS serves it at a predictable static path whose directory returns a JSON listing, so the filename is read programmatically:

```
https://<clms>/cloudlens/static/agent-update/windows/latest/
  -> [{ "name": "cloudlens-win-sensor-6.12.0.316.exe", "size": 62636500 }]
```

2. **Read the project key from a working sensor** so Windows joins the same project as Linux:

```
docker inspect cloudlens-agent --format '{{json .Args}}'
  -> --project_key 828f57cee6ba450d9346a30f97296d07 --server 34.230.225.109 --ssl_verify no
```

3. **Download and silent-install on the host** (in-VPC via the CLMS private IP 10.99.1.64):

```
<exe> /install /quiet Server="10.99.1.64" Project_Key="828f...07" SSL_Verify="no"
Auto_Update="no"
  -> Installer exit code: 0 ; CloudLens service: Running
```

4. **Confirm registration** (not just the service): agent.yml is written and the sensor enters a control loop with the CLMS: `Config received ... Posting sensor state!`

SSM automation gotchas Newlines in an SSM command string arrive as literal `\n` and break the PowerShell parser: base64-encode the script and Invoke-Expression it. Invoke-WebRequest against the self-signed CLMS fails "underlying connection closed on send": use an add-type ICertificatePolicy TrustAll, force TLS 1.2, and WebClient.DownloadFile.

Appendix D. KVO and CLMS reference learned in this build

Fact	Detail
KVO admin	admin / admin, Keycloak at /auth/realms/keysight/protocol/openid-connect/token, client_id vision-orchestrator. NOT the vController password.
CLMS login fields	The identity login expects Email and Password, not username. The admin password is rotated on first use.
Change-request lifecycle	createChangeRequest, mutate, commitChangeRequest. A refused CR stays InProgress and raises an alert; it never moves to Failed, and the reason is only on the Alerts page.
MonitoringPolicy tools	Not a top-level queryable field; read under settings { tools { name } source { name } }. Edit with updateMonitoringPolicyById(uid, changeID, settings).
REMOTE vs LOCAL tool	A REMOTE/cloud tool cannot bind a device port (reachableFrom CLOUD_CONFIG). The vPB egress tool must stay LOCAL; a capture host rides a second REMOTE tool on the same policy.
Workload selector	{field, tag, regex} where field is the tag key. field="tag" matches nothing.

awsConfiguration	mgmt/ingress/egress SG IDs are non-null. Supply three distinct SGs.

Appendix E. Change history

Every fix that made this pipeline work, in reverse chronological order, on branch main of github.com/Keysight-Tech/cloudlens-ansible-aws.

```
0dbc054  windows sensor: auto-discover the installer URL from the CLMS, proven live
5b5a901  windows sensor: download the installer from the CLMS, fail early when missing
952c583  deploy: provision a tcpdump capture host and wire it into the vPB egress
dc892d9  capture host: give the vPB egress a REMOTE tool so tapped traffic is visible
1054322  deploy: run the AWS mirror phase before the vPB traffic path
e5a7cbc  vpb: allow L2GRE (IP protocol 47) inbound; collector ASG max 1 -> 4
b301357  kvo-mirror: the workload selector field is the tag key, not the word tag
e59244b  vpb-path: say when KVO is refusing a change request
4809c20  kvo-mirror: the security groups are required, not optional
3d76ad6  windows bucket and licence coverage, both inside the run
735a496  vpb-path: stop blaming DPDK for a step that never ran out of ports
23b62c3  vpb-cli: multi-line CLI input never reached the CLI
```

Appendix F. Operational notes

Licence stranding

KVO is the licence authority and has no licence-release API. Destroying a KVO strands its licensed quantity permanently. When rebuilding, keep the existing KVO or obtain fresh activation codes first; do not tear down a licensed KVO expecting to re-use spent codes.

Reuse over rebuild

For re-validation, keep the control plane (KVO, CLMS, vPB, already licensed and adopted) and rebuild only the data plane (workloads, collector, mirror fabric, capture host). The orchestrator detects a CONNECTED CLMS and an adopted vPB and skips those phases.

Cost and teardown

Mirrored traffic is billable VPC traffic, and cross-AZ delivery adds transfer charges: keep the vPB in the same AZ as its sources. On teardown, the collector ASG and orphaned EBS volumes must be removed explicitly; a security group can otherwise deadlock stack deletion.

Recorded from live work on AWS account 466778915280, us-east-1. All numbers in Section 7 and the identifiers in Appendix A are real values captured during the build. Source and full history: github.com/Keysight-Tech/cloudlens-ansible-aws