Multicloud.io

STANDARDIZE · COMMODITIZE · CONNECT

Multicloud: The Compute Grid

The architecture of the compute grid: standardize the machines, commoditize the capacity, connect the clouds — AWS, Azure, and Google Cloud today — into one dependable, Kubernetes-native pool in cloud accounts you own. What reaches you is a standard socket: Kubernetes.

Architecture Whitepaper | July 2026 | multicloud.io

EXECUTIVE SUMMARY

The pool, in one page

Cloud compute is a commodity that doesn't trade like one. The same silicon carries different names, prices, and availability in every region of every cloud — and the spot market, where compute is cheapest, asks you to absorb interruptions in exchange. Multicloud operates that market for you, and hands you the result as something entirely ordinary: **a Kubernetes cluster** — or a single node inside the cluster you already have.

> Like a power plant behind a wall socket, the machinery is ours to run; what reaches you is standard.

Three principles shape the architecture:

- **Land beside, not migrate.** Your cluster, pipelines, and tooling stay exactly as they are; Multicloud attaches next to them.
- **Your accounts, your bill.** The machines behind your pool are provisioned into dedicated cloud accounts you own — compute at cost, on your own bill and your own rates. Multicloud charges for the platform, never a markup on machines.
- **Nodes are replaced, never repaired.** Every machine is disposable by design; all durable state lives outside the fleet, so losing any machine — or all of them — loses nothing.

Underneath: a continuously refreshed catalog of roughly **265,000 priced offers** with benchmarks keyed to real CPUs, one scheduler making one placement decision across all of it, an encrypted per-cluster network fabric spanning clouds, and a platform layer — serverless runtime, GitOps, observability, TLS, DNS — operated entirely on our side of the line.

You plug in through the door that fits: a read-only savings report, a virtual node in your existing cluster **(early access)**, a dedicated cluster whose keys you hold, or a PaaS-style deploy **(early access)**. The attach surface stays small enough to enumerate — and this paper enumerates it.

CONTENTS

What's in this paper

This paper describes the platform's architecture. Some surfaces are marked **(early access)**; for current availability and onboarding, talk to us. The companion engineering document is maintained alongside the codebase and versioned with it.

SECTION 01

01 The idea

Multicloud maintains a live catalog of priced, benchmarked machine types across AWS, Azure, and Google Cloud; buys the cheapest capacity that satisfies your workloads' real requirements (AWS and Google Cloud in production; Azure provisioning in final validation); absorbs the interruptions; and hands you standard, upstream-conformant Kubernetes. Nothing about your existing setup changes — the platform lands beside what you run, in dedicated cloud accounts you own.

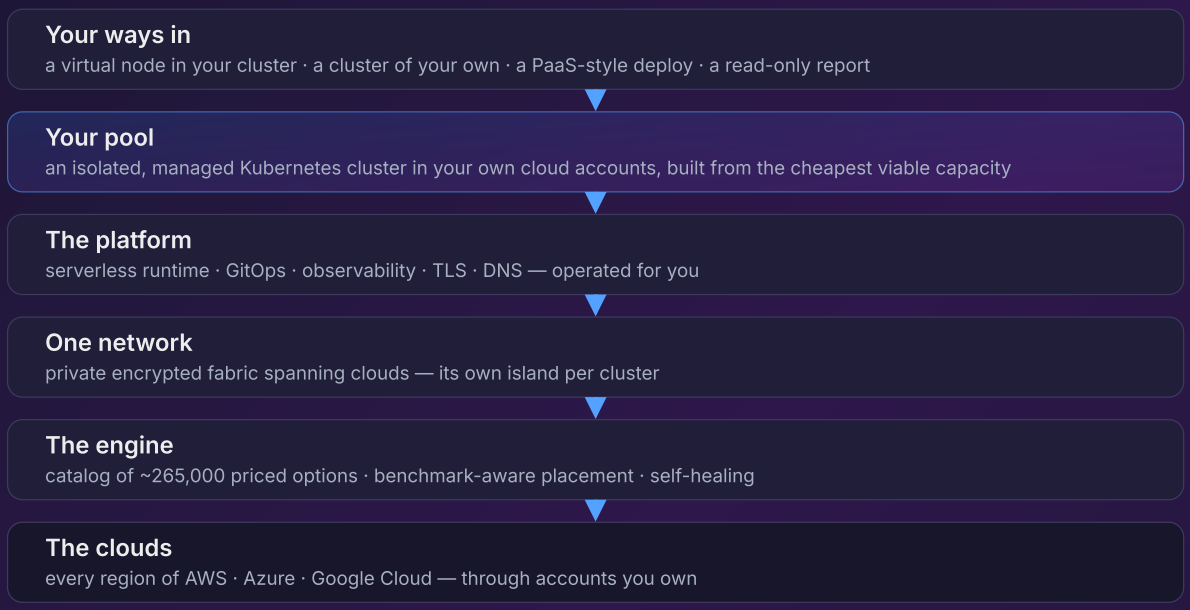

Figure 1 — the layers, top-down: every door lands on the same pool, the pool rides one engine, the engine buys from the clouds.

Each layer is the subject of a section below. The order is deliberate: everything the paper describes exists to make the top layer — the way you consume it — boring, standard, and small.

SECTION 02

02 The engine: one catalog, one vocabulary, one placement decision

Underneath everything is a continuously refreshed catalog: roughly **265,000 priced offers** across ~3,900 machine types, every region and zone, spot and on-demand, with interruption-risk data — and, crucially, **benchmarks**. Benchmarks are keyed to the actual CPU model rather than the instance name, so a "4 vCPU" machine in one cloud can be compared honestly against a "4 vCPU" machine in another that delivers twice the compute.

When your workloads need capacity, the scheduler makes **one placement decision** over that whole catalog: it packs pending work into the smallest set of nodes that fits — by memory, by benchmarked compute, and by raw cores — then buys the cheapest concrete machines, in whatever cloud, region, and zone, on spot where your constraints tolerate interruption and on-demand where they don't, spreading across zones when resilience calls for it. Workloads can express what they need in normalized compute units, so "enough CPU" means the same thing on every cloud and every hardware generation.

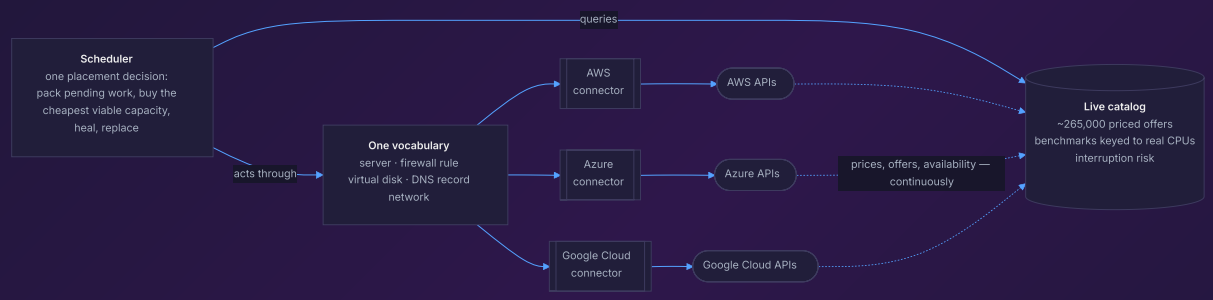

Figure 2 — the engine: the scheduler queries one catalog and acts through one vocabulary; thin connectors translate at the boundary while the clouds continuously feed prices, offers, and availability back into the catalog.

One vocabulary, three dialects

The catalog normalizes what we buy; the same discipline normalizes how we act. Each cloud's API and its quirks live in a thin connector at the boundary — and everything inboard of the connectors speaks one vocabulary: a **server** is a server, a **firewall rule** is a firewall rule, a **virtual disk** is a virtual disk, a **DNS record** is a DNS record, whichever cloud it happens to live in. Every behavior in this paper — placement, healing, firewall derivation, network reconciliation — is written once against that vocabulary, not once per cloud SDK.

> Clouds become interchangeable in practice, not just in principle: adding a cloud is one more connector, not one more platform.

Constraints: the cluster is the contract

The requirements and constraints are yours to set, per cluster. Want a cluster that only runs in the EU? One pinned to a single cloud — or free to roam every cloud we operate? A cluster that never exceeds $10 an hour? One that always keeps at least five nodes, with a warm spare standing by before it's needed? Nodes with a specific GPU, a memory floor, a benchmarked-compute floor — or on-demand only, because interruptions are off the table?

Each of these is a constraint the scheduler honors on every placement decision — not a support ticket:

Where nodes may exist
regions, zones, clouds — include and exclude rules with hierarchical precedence; "EU-only" is one rule.

What a node must be
memory floor, benchmark floors, GPU type / count / memory, disk size.

How much risk to take
a maximum interruption-risk knob is also the spot-vs-on-demand selector; zero means on-demand only.

The scaling envelope
minimum and maximum node counts, warm spare headroom, idle-reap timing, an aggregate hourly cost ceiling.

SECTION 03

03 Self-healing: interruptions are weather, not incidents

Spot capacity is reclaimed by the cloud with seconds-to-minutes of notice. The pool treats this as routine weather. Every node carries an agent that watches its cloud's reclaim signals. On notice, workloads are evicted in priority order, honoring your disruption budgets, inside each provider's real notice window. Replacements come from the same catalog decision as everything else — which means a reclaim in one zone is often an upgrade to cheaper capacity somewhere else.

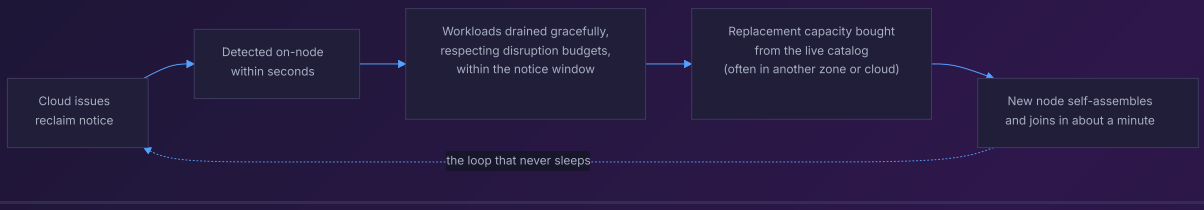

Figure 3 — the loop that never sleeps: notice → on-node detection → graceful drain → replacement bought from the live catalog → self-assembly and join, in about a minute.

Machines that die without notice, go idle, or drift are handled by the same family of reconcilers: liveness reconcilers catch unannounced deaths, idle reapers return capacity nobody is using, and replacement is always the same act as first provisioning — there is no special "recovery" code path to go wrong.

> Because no machine holds durable state — cluster state itself lives in a managed external database — even the loss of every node simultaneously is designed to be a re-provisioning event, not a data-loss event.

SECTION 04

04 One network, isolated per cluster

Each cluster gets its own private network **in each cloud it spans** — never shared, never peered with anyone's, not even your own other clusters. A customer can run any number of clusters; every one is its own island. Within the island, three rings do the work:

Ring 1 — the cloud's firewall: derived, not managed. The scheduler continuously reconciles each network's firewall rules from what the cluster actually runs: a port is open because a published service exists behind it, and closes when the service goes. SSH is never open to the internet. No rule is opened "temporarily" by a person, because no person is in the loop — and when a cluster is deleted, its networks are torn down with it.

Ring 2 — an encrypted fabric: membership is a key. Across clouds, nodes join a WireGuard mesh over IPv6. Being on a cluster's network *is* holding a key its fabric recognizes — not a source-IP rule — and traffic without a recognized key is dropped at the first packet. Everything crossing the fabric is encrypted node-to-node: pod traffic and host traffic alike. Tunnel geometry (MTU, TCP segment sizing) is tuned for the real cross-cloud wire, where path-size discovery can't be trusted.

Ring 3 — a modern pod network: per cluster, kernel-fast. On top of the fabric, each cluster runs its own instance of Cilium, an eBPF-based container network: one flat, encrypted, dual-stack pod network spanning providers, load-balanced in the kernel rather than by proxies, with traffic kept zone-local where possible to avoid cross-zone cost and latency.

At the edge, the platform operates DNS, TLS certificates, and ingress for you: hostnames are published automatically for the services you expose, certificates are issued and renewed automatically, and every node in the pool can terminate traffic — so the edge scales with the fleet and survives any node's loss.

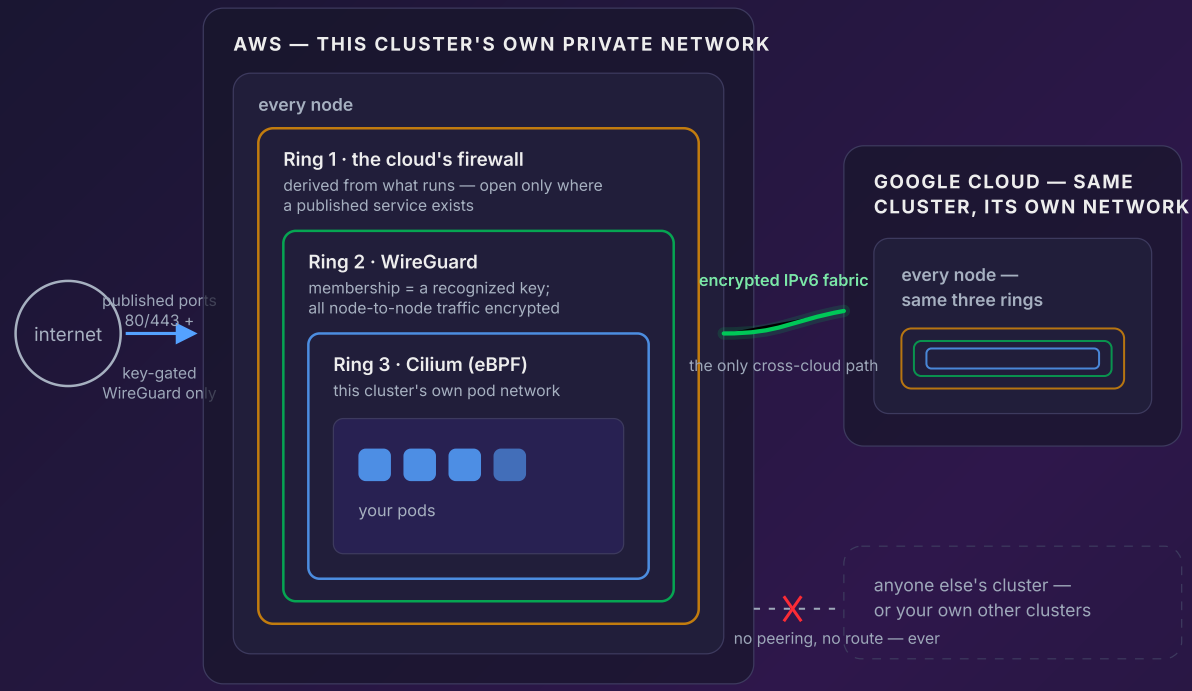

Figure 4 — defense in depth, per node: the derived cloud firewall, the key-authenticated encrypted fabric, and the per-cluster pod network. The fabric is the only cross-cloud path; other clusters — including your own — have no route in.

Traffic is also kept **zone-local by preference**: services that opt in are served by the nearest healthy backend first, falling back across zones only when needed — dodging cross-zone transfer cost and latency without giving up resilience.

SECTION 05

05 How a new node joins — no inbound management surface

A pool that replaces machines all day is only as trustworthy as its join path, so here it is, spelled out. The short version: **a new node lets itself out — nothing lets itself in.**

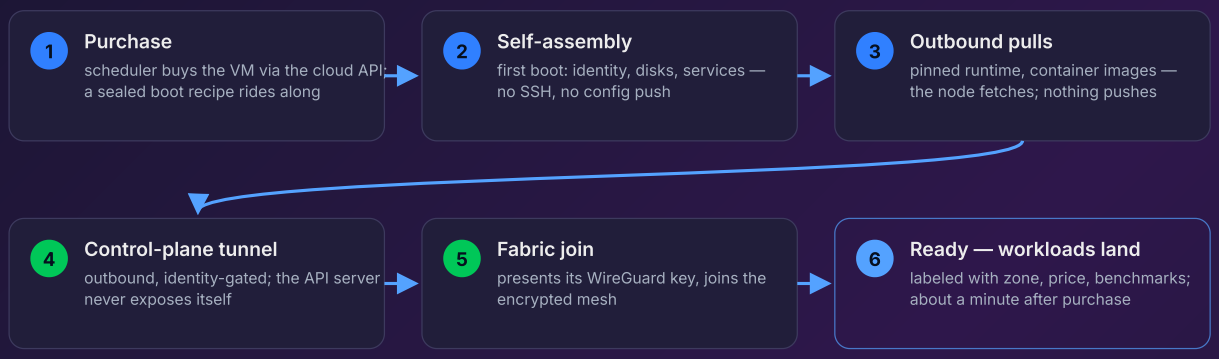

Figure 5 — the join path. Every arrow that crosses a trust boundary points outward: the node dials out for everything it needs.

Three properties make this shape matter:

- **Configuration travels with the purchase, then never again.** The boot recipe — the node's identity and the exact set of services it runs — is attached to the cloud's own provisioning call. Nothing ever pushes configuration to a running node; a node that should be configured differently is replaced, not edited. There is no drift, because there is no mutation.
- **The control plane never exposes itself.** The rendezvous is a tunnel the *node* initiates, outbound and identity-gated, with two independent credentials in the path: an edge service token and the cluster's own join token. Your cluster's API server is never open to the internet — it doesn't even need a route back to the node.
- **The listening surface is enumerable.** What will accept an unsolicited packet on a pool node: the fabric's WireGuard port, which is key-authenticated — without a recognized key the conversation ends at the first packet — and the public service ports (80/443) on nodes terminating traffic you chose to publish. No internet-facing SSH, no management API, no agent port. That is the complete list.

06 A real Kubernetes, batteries included

What you get is not a Kubernetes-like abstraction — it is upstream-conformant Kubernetes. On top of it, the platform runs and operates the pieces most teams end up assembling themselves:

- a **serverless runtime** (Knative) for HTTP services with scale-to-zero,
- **GitOps delivery** (Argo CD): you push, the platform pulls and deploys,
- **observability** — logs, metrics, and traces collected and queryable, with cost visibility that understands per-machine spot prices rather than flat rates,
- **TLS, DNS, SSO, and secrets management** as managed plumbing,
- a per-node **image cache**, so a fleet spanning clouds doesn't hammer registries.

> All of it is operated by control loops on our side of the line. None of it lands on your on-call.

07 Isolation and sovereignty

Customers get isolation in the tier that fits, on one shared machinery:

- **Workspace tier** — your workloads run in their own namespace on managed shared infrastructure, walled off at the API layer: your credentials map to your workspace and nothing else. It is designed for instant provisioning and near-zero idle cost.
- **Dedicated cluster tier** — your own Kubernetes cluster: your own control plane, API server, certificate authority, RBAC domain, network fabric, and datastore. Control planes are hosted as pods on our infrastructure (no control-plane machines for you to pay for or babysit), while worker capacity is bought from the catalog on demand — provisioned into the dedicated cloud accounts you own, every machine visible in your own console and on your own bill. Promotion from workspace to dedicated cluster is a configuration step, not a migration.

A dedicated cluster whose keys you hold — standard kubeconfig, full admin — is what we call a **sovereign cluster**: yours to point any tooling at, operated for you underneath, reachable only through its own authenticated endpoints.

SECTION 08

08 Four ways in

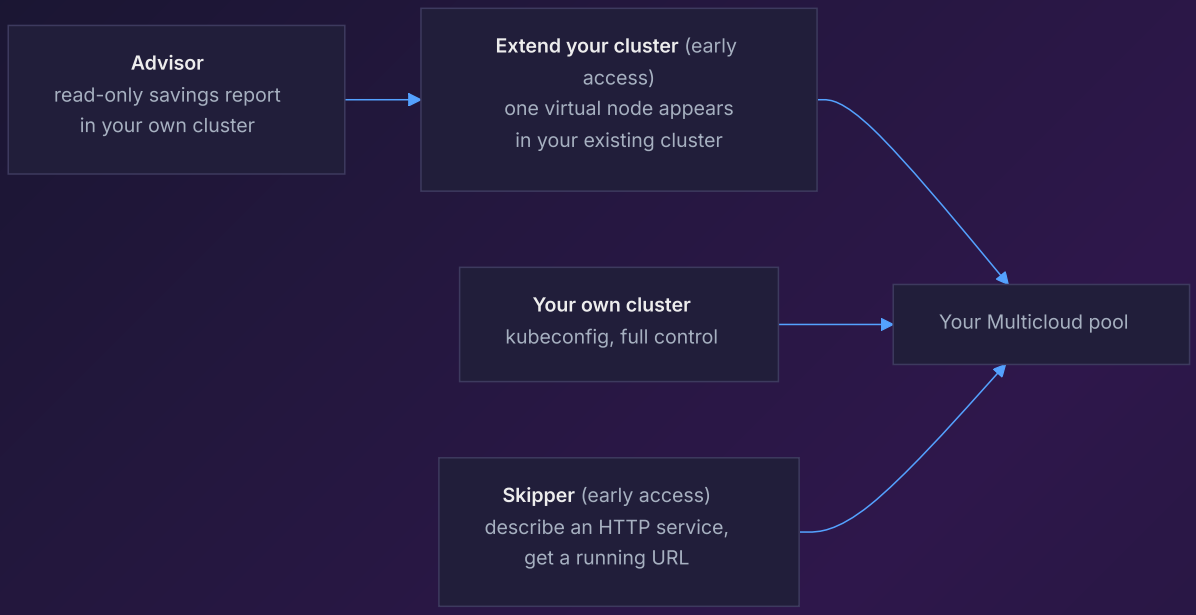

Figure 6 — the doors: evidence first, then attach at the depth that fits. Every door lands on the same pool.

Start with evidence — Advisor. A read-only report you install in your own cluster. It inspects nothing but resource declarations, sends nothing outside except one anonymous catalog query, and computes what your estate would cost if it were placed through the catalog — with interactive levers for right-sizing, cloud choice, regions, and spot. It quantifies; it changes nothing.

Extend your cluster (early access). Your existing cluster (EKS, GKE, AKS, or self-managed) gains one new node — a virtual node backed by your Multicloud pool. Pods you mark for it are scheduled there like anywhere else; pending demand behind it drives our autoscaler. Your control plane, your tools, and every other workload remain untouched. When you want more than burst capacity, the same pool is promotable to a sovereign cluster.

Your own cluster. A dedicated cluster, handed over: kubeconfig, full admin, standard Kubernetes — operated for you underneath.

Deploy without a cluster — Skipper (early access). For teams that just want software running: you describe an HTTP service in a short manifest (image, port, environment, secrets, scaling bounds) and the platform compiles it onto the serverless runtime — a URL, TLS, logs, and scale-to-zero included, authenticated by organization-scoped API keys.

A machine interface runs alongside all four: a CLI, client libraries, and an MCP server that lets AI assistants search and compare the live catalog directly.

SECTION 09

09 How this differs from the alternatives

Almost everything in this market makes the cloud you are *already* on cheaper to use. Multicloud changes which cloud you buy from. That is a different axis — and the two multiply rather than compete.

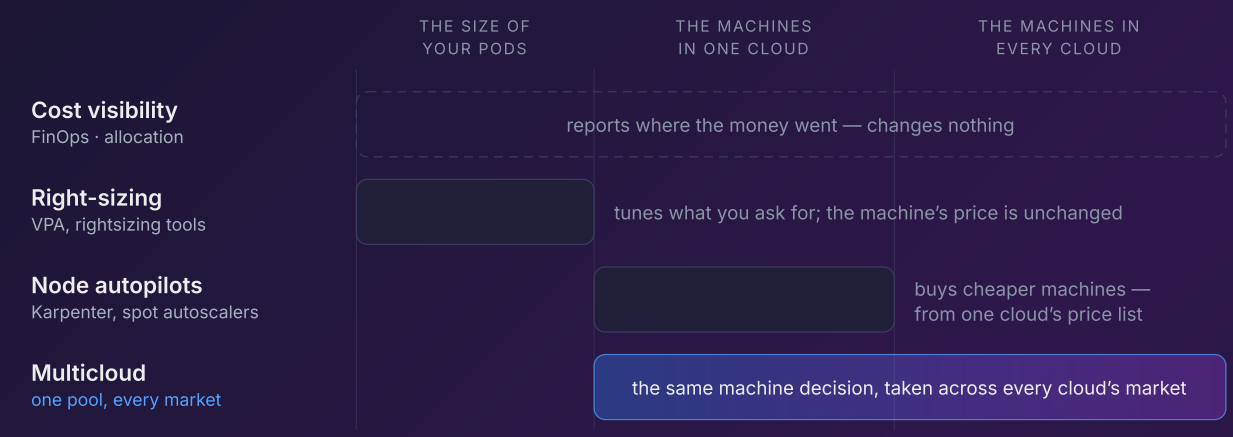

Figure 7 — how far each approach reaches. The field competes over the two left-hand bands; the right-hand band is the one a single-cloud tool cannot enter.

They compose. Asking for the right amount and packing it tightly are real savings, and you should keep doing both — we pack your pods into the smallest set of machines that fits, and the Advisor prices right-sizing for you. But once the requests are honest, one provider's price list is the floor. Ours is the whole market.

A single-cloud autopilot cannot cross the street. Inside its own cloud it is mature and effective: better SKUs, spot, tighter bin-packing. What it cannot do is notice that the same benchmarked compute is a third cheaper in another provider's region this week — its catalog is that provider's own. Making the comparison honestly takes a normalized, benchmark-keyed, continuously priced catalog spanning the clouds. That catalog is the product.

Trust runs the other way, too. To act, an autopilot needs write credentials into the accounts your production already lives in. Multicloud runs in dedicated accounts created for it (section 10), so the blast radius is bounded by construction rather than by policy.

> Where the field is ahead of us: the single-cloud autopilots have years of production traction, compliance certifications, and polished dashboards, and parts of this platform are still early access. If you hold committed-use discounts, your floor on that cloud is below list — which is exactly why the front door is a read-only report computed against your numbers, not a headline percentage.

SECTION 10

10 Your accounts: onboarding without handing over keys

The machines Multicloud runs for you live in **cloud accounts you own**, not ours. Onboarding is built around that separation:

1 · Create fresh, dedicated accounts. Under your existing organization you create new accounts that exist solely for Multicloud — an AWS account, a Google Cloud project, an Azure subscription. Nothing you already run is touched, and nothing Multicloud does can reach outside them. An AI-assisted setup skill and a set of scripts we provide (early access) automate and streamline the whole process, end to end.

2 · Grant scoped credentials. You hand the platform API credentials to those dedicated accounts — and only those. Your existing accounts, IAM, and workloads stay invisible to us.

3 · Keep the off-switch. Everything the platform creates — machines, networks, DNS — is visible in your own console, auditable in your cloud's own logs, and billed at your own negotiated rates. Rotate or revoke the credentials and the platform is cut off completely.

> Your cloud gave you credits, or a special discount? Amazing — use them. Because the machines are bought in your accounts, startup credits, enterprise discount programs, and committed-use plans all still apply — stacking on top of whatever the catalog already saved you.

This is what makes "compute at cost" literal rather than a discount promise: the machines are purchased from your cloud, in your accounts, on your bill. Multicloud operates them, and charges for the platform.

SECTION 11

11 Security posture, summarized

- **Isolation is structural.** Per-cluster networks with no peering — a customer's clusters are isolated even from each other; each dedicated cluster has its own CA and datastore; workspace credentials cannot address another tenant's workspace even in principle.
- **Control traffic is identity-gated.** Cluster control planes are reached through authenticated, per-cluster tunnel endpoints — API servers are never simply open to the internet, and nothing requires inbound access into your environment.
- **Everything cross-cloud is encrypted** node-to-node (WireGuard), for pod traffic and host traffic alike.
- **Least exposure by construction.** SSH is never world-open; public ports exist only where a service is actually published; secrets flow from a managed external store into workloads without ever landing on machine disks.
- **Your cloud, your bill.** Compute runs in dedicated accounts you own (section 10) — at cost, under credentials scoped to those accounts and revocable by you, with Multicloud charging for the platform, never marking up the machines.

Appendix A goes one level deeper: who authenticates how, isolation tier by tier, and how secrets move. It is written to survive your security team's questions, including the ones about what is *not* finished yet.

SECTION 12

12 What crosses the line

The entire attach surface between your world and ours is small enough to enumerate: the **scoped credentials** you grant to your dedicated Multicloud accounts; an **outbound-initiated, identity-gated tunnel** for control traffic; a **git remote** you push to; the **key-authenticated encrypted fabric** between pool nodes; and, if you use it, the virtual node's outbound connection from your cluster. Your IAM, your network policies, and your existing ingress remain exactly as they were.

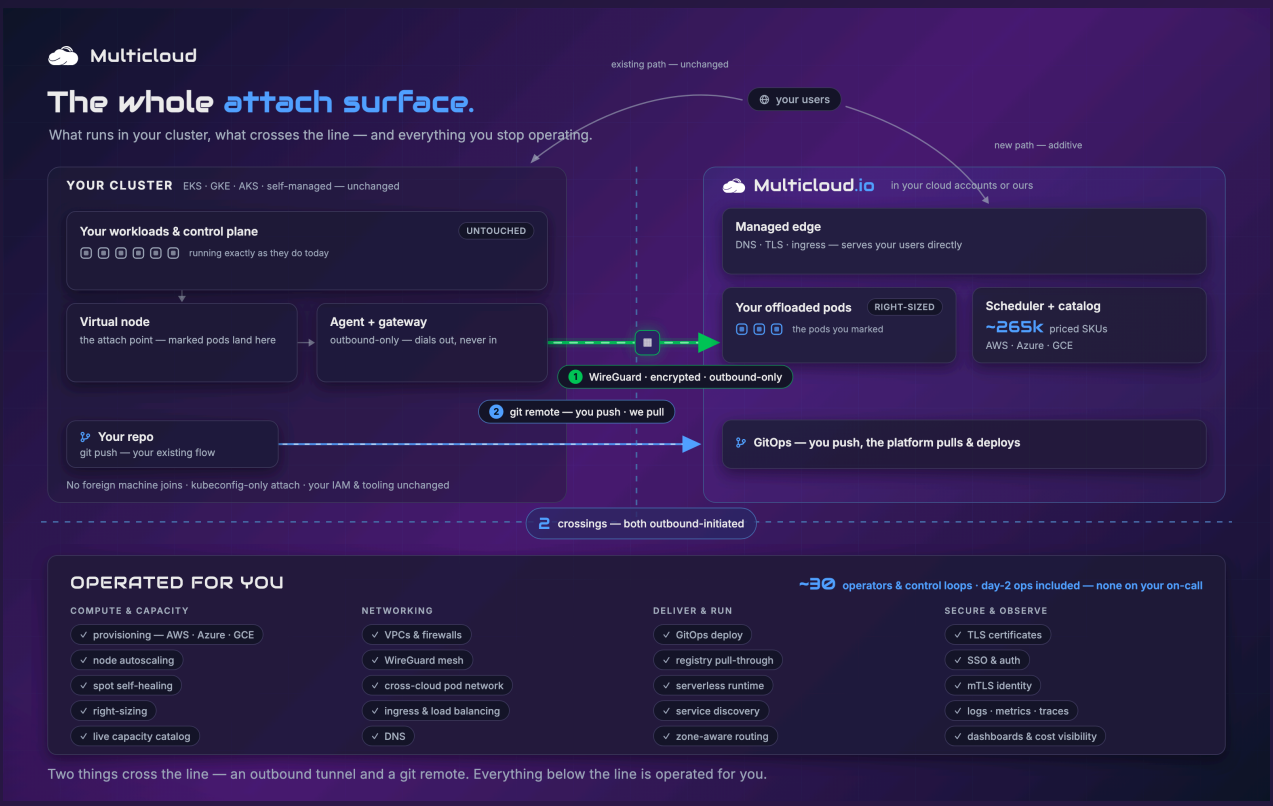

Figure 8 — the whole attach surface: your cluster on the left, the operated pool on the right, exactly two crossings on the seam — both outbound-initiated — and the full rail of platform jobs that stop being your on-call.

APPENDIX A

Security appendix: identity, isolation, secrets

Who authenticates how

CALLER	MECHANISM	BOUNDARY IT ESTABLISHES
Machine clients / CLI	Mutual TLS at the edge; client certificates verified against a per-environment CA	A dev credential cannot reach prod — environments are cryptographically separate CAs, not naming conventions
Deploy & workspace APIs	Organization-scoped API keys — 256-bit random, stored only as hashes, shown once; no disable switch exists by design	The tenancy wall: your authenticated organization is the *only* input that selects your workspace
Your engineers (dashboards)	Single sign-on (OIDC), with tokens cryptographically re-verified by each app behind the proxy — the proxy is not the last line	Customer plane, deliberately isolated from the operator plane's SSO state
Cluster workers & your kubectl	Per-tenant identity-gated tunnel tokens, short-lived join tokens, and client certificates from your cluster's own CA	Tenant control plane: cross-tenant access fails at three independent layers (TLS, edge identity, client auth)

Isolation, tier by tier

Workspace tier: the enforced boundary is the API layer — your organization maps to your namespace by a pure function, so cross-tenant addressing is impossible at the request layer. Secrets add three independent per-tenant walls: path-prefix scoping, a namespaced secret store pinned to your prefix, and a per-tenant cloud IAM role. Network-level east-west policy between workspace namespaces is on the hardening roadmap; the request-layer and secrets walls are the enforcement today — stated plainly because your security team will ask.

Dedicated tier: a full per-tenant control plane, certificate authority, RBAC domain, network fabric, and datastore. There is no shared API layer to trust: isolation is structural.

How secrets move

Customer secrets flow from a managed external secrets store into namespace-local Kubernetes secrets and then into workloads — never onto node disks, because nodes have none worth trusting: no persistent volumes, ephemeral machines. Envelope encryption of platform-side secret material is on the roadmap; secret distribution to workloads is enforced as described today.

APPENDIX B

B Frequently asked questions

Is this a fork of Kubernetes?

No. Clusters are upstream-conformant Kubernetes. Anything that works against standard Kubernetes — operators, Helm charts, CI tooling, kubectl — works here.

What happens if Multicloud disappears?

Your machines run in accounts you own, and your clusters are standard Kubernetes with their state in a managed database. The platform's value is the operating loops; losing the operator does not strand your compute or hold your workloads hostage.

Which clouds are supported today?

AWS and Google Cloud in production; Azure provisioning is built and in final validation. The catalog already reaches further — it prices additional providers for comparison — and the connector architecture makes each new cloud one more connector, not a re-platform. The roster grows.

What can Multicloud see of our data?

The platform operates infrastructure: machines, networks, DNS, deploy pipelines. Your application data lives in your workloads and your stores. Advisor, the evaluation tool, reads resource declarations only and sends nothing outside except one anonymous catalog query.

Who owns the cloud accounts?

You do. They are created under your organization, dedicated to Multicloud, and governed by credentials you can scope, rotate, or revoke at any time. Revocation is a complete off-switch.

Can we leave?

Yes, and cheaply: nothing about your workloads is proprietary to the platform. Standard Kubernetes manifests run anywhere; the accounts, and everything in them, are already yours.

How do interruptions affect our SLOs?

Spot is a choice per cluster, not a requirement. Where you allow it, disruption budgets are honored during drains and replacements arrive in about a minute; where you don't, the scheduler buys on-demand. Most estates mix both.

How is Multicloud priced?

Compute is yours at cost, on your bill. Multicloud charges for the platform — never a percentage markup on machines, so our incentive stays aligned with buying you the cheapest viable capacity.

APPENDIX C

Glossary

Pool
Your Multicloud capacity: an isolated, managed Kubernetes cluster built from the cheapest viable capacity across clouds, in your own accounts.

Catalog
The continuously refreshed database of ~265,000 priced offers — every machine type, region, zone, spot and on-demand, with interruption risk and CPU-true benchmarks.

Normalized compute units
The platform's compute currency: benchmarked capability per core rather than raw core count, so "enough CPU" means the same thing on every cloud and hardware generation.

The fabric
The per-cluster encrypted WireGuard mesh over IPv6 that connects a cluster's nodes across clouds — membership by key, the only cross-cloud path.

Workspace tier
Namespace-scale tenancy on managed shared infrastructure, walled off at the API layer; instant to provision, near-zero idle cost.

Sovereign cluster
A dedicated cluster whose keys you hold: your own control plane, CA, RBAC domain, fabric, and datastore — standard kubeconfig, operated for you underneath.

Virtual node
The Extend-your-cluster attach point: one node in your existing cluster, backed by your entire pool; pods you mark for it run there.

Boot recipe
The sealed first-boot payload attached to a machine purchase — identity plus the exact services the node runs. Nodes are never configured after boot; they are replaced.

NEXT STEP

Start with evidence. Install Advisor — read-only, in your own cluster — and see what your estate would cost placed through the catalog.

Then pick your door: a virtual node beside what you run, a sovereign cluster of your own, or a URL from a manifest. multicloud.io