CLOUDFLARE

The agentic network: Secure private connectivity for agents

Nikita Cano
Senior Product Manager
Cloudflare

CLOUDFLARE

Agents cannot log in

Human access depends on four things an agent does not have

	Opens a browser	Signs in to the IdP	Approves an MFA push	Holds a session
A PERSON can do all four	✓	✓	✓	✓
AN AGENT can do none of them	✗	✗	✗	✗

An agent does a person's job without a person's credentials

Agents need network access, not application access

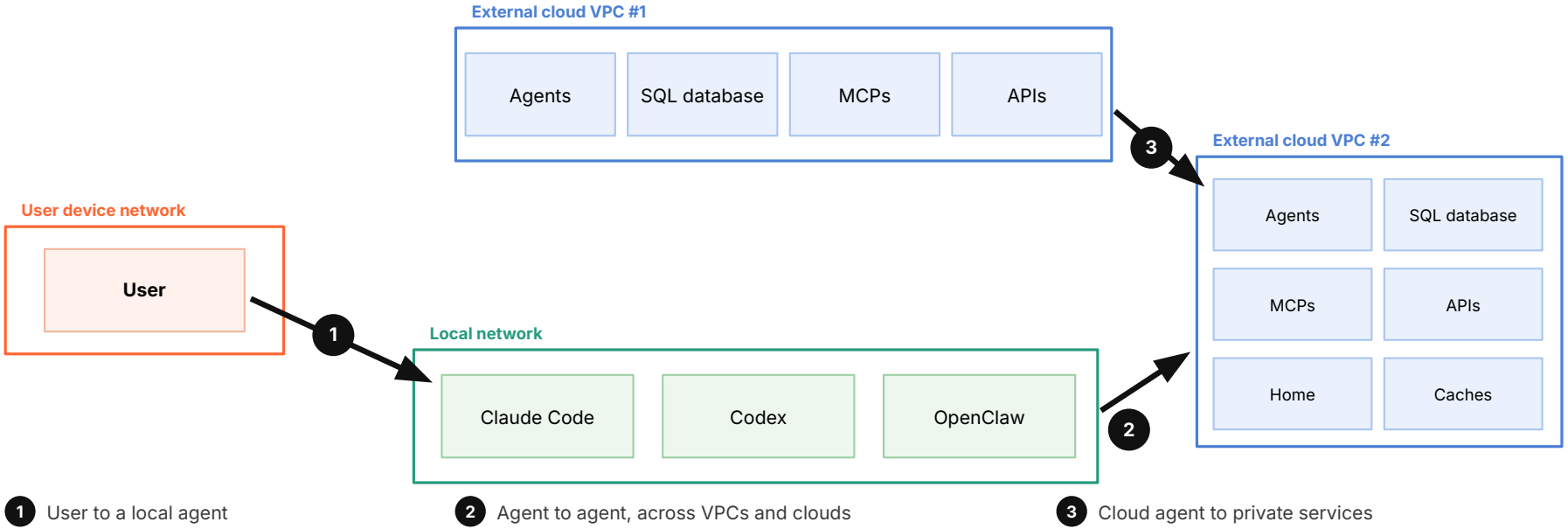
CLOUDFLARE

Three patterns that all need private IPs

All new in the last year – none of the usual tools covers them

NONE OF THESE FIT

- ✕ VPN
- ✕ SSH
- ✕ Public endpoints
- ✕ Tunnel



1 User to a local agent

2 Agent to agent, across VPCs and clouds

3 Cloud agent to private services

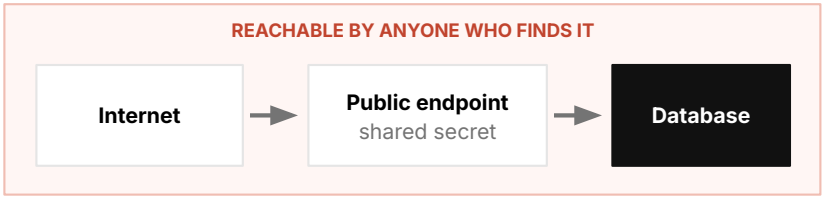
Every agent project turns into a network project

CLOUDFLARE

Two bad options

Put it on the Internet, or hand the agent a VPN credential

OPTION A — EXPOSE IT PUBLICLY

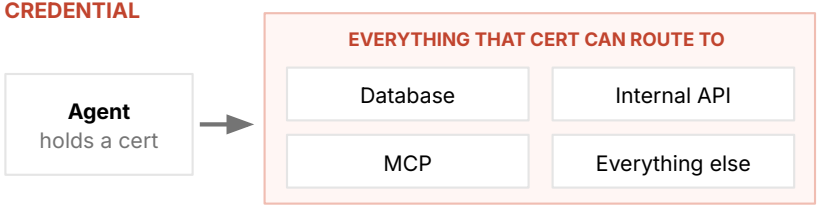


A token is all that stands in front of the data

Anyone holding it is you

The scope of impact is the whole Internet

OPTION B — HAND OVER A VPN CREDENTIAL



Scope is the network, not the tool

Identity is a device, not a workload

The log line reads "a device connected"

Both leave a credential in the open and no record of its use

Neither one passes a security review

CLOUDFLARE

Agents shipped faster than their controls

Four numbers from Okta's 2025 research on agent identity

51%
of companies have already deployed AI agents

23%
of IT professionals report agents tricked into revealing credentials

80%
report unintended actions taken by bots

44%
have a policy governing what agents may access

The connectivity layer underneath these agents was designed for people – replacing it is the work

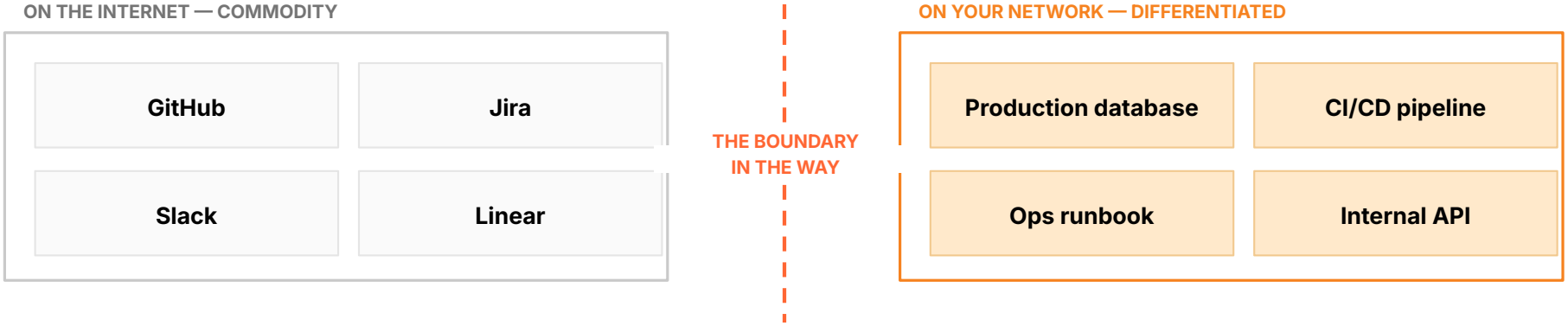
Source: Okta, "How Okta helps secure AI agent identity" (2025), citing PagerDuty and *The Times*

Agent connectivity is being built ahead of the controls around it

CLOUDFLARE

The MCP servers that matter are private

Commodity servers sit on the Internet, the valuable ones do not



A gateway can only govern what it can reach

Teams ship the internal server, point an agent at it, and find the gap in the incident review

A private server still needs a policy in front of it

CLOUDFLARE

Three products, three jobs

They work together – unified routing moves traffic between all three

Cloudflare Tunnel
Publish services / L7 apps

DIRECTION	Inbound to origin
LAYER	L7 (QUIC, HTTP/2)
CONNECTOR	cloudflared
BEST FOR	Publishing web apps

Cloudflare Mesh
Devices and servers, privately

DIRECTION	Bidirectional
LAYER	L3 (any IP) via MASQUE
CONNECTOR	warp-cli
BEST FOR	Private networking, agents

Cloudflare WAN
Connect whole sites

DIRECTION	Bidirectional
LAYER	L3 (any IP)
CONNECTOR	IPsec / GRE / CNI
BEST FOR	Branches, hardware on-ramp

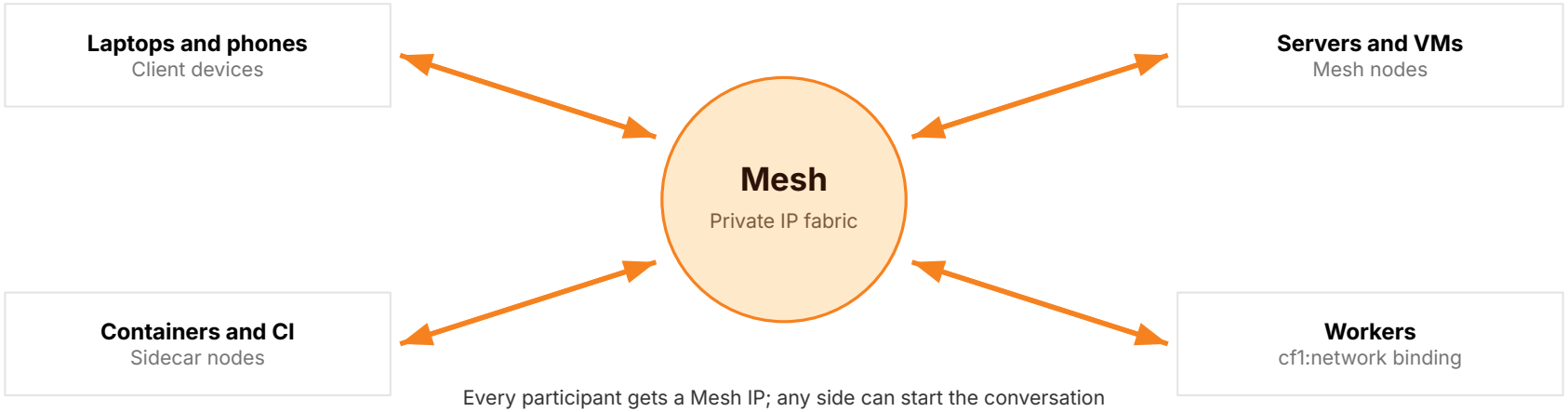
Bidirectional? → Mesh | **Source IP preserved? → Mesh** | **Agent connectivity? → Mesh** | **LAN or 1 Gbps+? → WAN**

If it is an application, use Tunnel

If it is a network, use Mesh

The agent becomes a participant

Devices, servers, containers, and Workers all get a private Mesh IP



TCP, UDP, and ICMP in both directions, over MASQUE with post-quantum key agreement

Cloudflare stays in the path, so policy, posture, and logs apply to every flow

CLOUDFLARE

Same software, two modes

One client: What changes is who the participant is

Cloudflare One Client

Laptops, phones, tablets

- Identity = User

- GUI with a system tray
- The user signs in through your IdP
- Coffee shop, home, office
- Agents on a laptop ride this connection

Cloudflare Mesh Node

Servers, VMs, containers

- Identity = Service

- Headless, runs as a daemon
- Per-node token auth, no browser
- Proxies traffic to a whole subnet
- High-availability failover

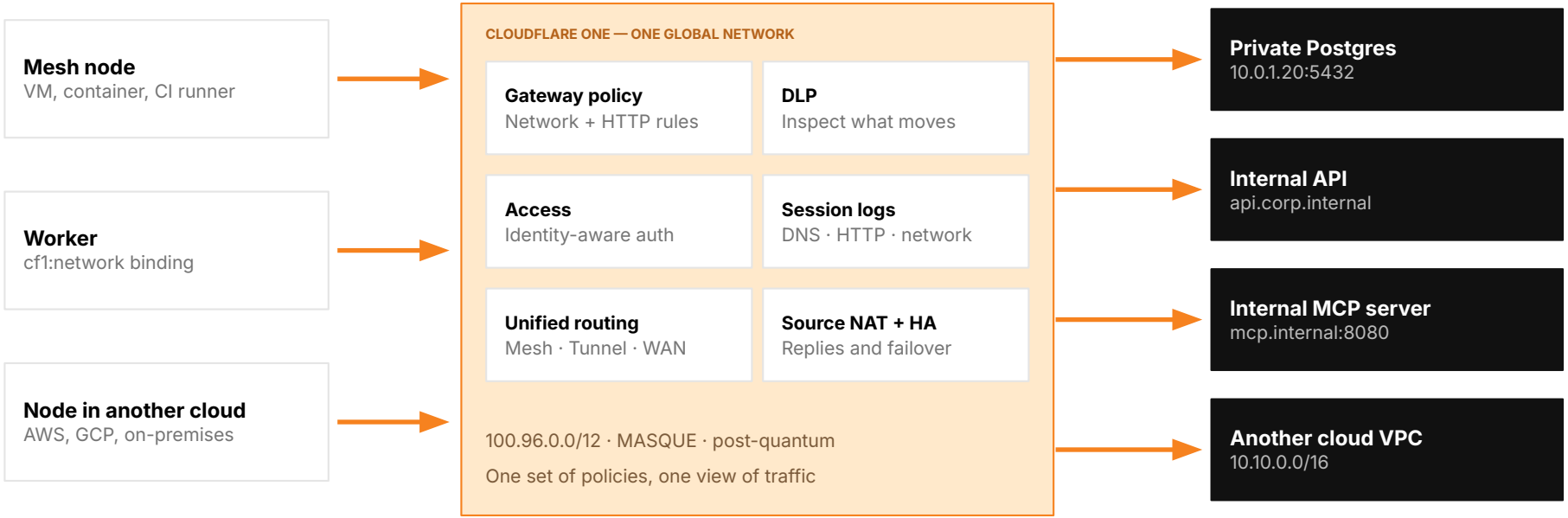
An agent on a laptop uses the user's identity; an agent on a server gets a service identity

That is what lets you write one policy for people and another for workloads

An agent takes the identity of the participant it runs beside

CLOUDFLARE

Three on-ramps, one policy plane



Agents and employees reach private resources through the same controls

CLOUDFLARE

A coding agent reaches private Postgres

No bastion, no port forward, no VPN client in the agent's runtime

ONCE, ON THE BOX BESIDE THE DATABASE

```
$ sudo warp-cli --accept-tos connector new <TOKEN>
$ sudo warp-cli --accept-tos connect
Status update: Connected
# the node now has a Mesh IP
```

EVERY TIME, FROM THE AGENT'S SHELL

```
$ nc -vz 100.96.4.17 5432
Connection to 100.96.4.17 5432 succeeded!
$ psql -h 100.96.4.17 -U agent -d orders
orders⇒
# no public IP, no bastion, no port forward
```

WHAT THE AGENT GETS

A private address

Nothing to install in the agent's runtime and no network credential for it to hold or leak

WHAT YOU KEEP

Routes and policy

The node advertises only the routes you give it, and Gateway decides what is allowed

Put a node beside the private resource, then dial its Mesh IP

CLOUDFLARE

A Worker reaches an internal API

One binding: The address you pass at runtime picks the destination

WRANGLER.JSONC

```
{
"vpc_networks": [
{
"binding": "PRIVATE_NETWORK",
"network_id": "cf1:network",
"remote": true
}
]
}
// or pin one: "tunnel_id"
```

SRC/INDEX.TS

```
export default {
async fetch(request, env) {
const res = await env.PRIVATE_NETWORK.fetch(
"http://api.corp.internal/orders/42"
);
// raw TCP: Redis, MQTT, anything
const s = await env.PRIVATE_NETWORK
.connect("10.0.1.50:6379");
return res;
},
```

SCOPE

cf1:network covers everything

Mesh nodes, routes, Tunnels, and WAN on-ramps

WATCH FOR

remote: true is required

Local dev cannot reach a private network

One binding reaches private services however they connect

CLOUDFLARE

Two clouds, one mesh, both directions

No VPC peering, no transit gateway, no overlapping-CIDR negotiation

AWS NODE — 10.0.0.0/16 BEHIND IT

```
$ sudo warp-cli connector new <TOKEN-A>
$ sudo warp-cli connect
# advertise the subnet behind it
# Networking > Mesh > Routes
10.0.0.0/16  →  aws-node
```

GCP NODE — 10.10.0.0/16 BEHIND IT

```
$ sudo warp-cli connector new <TOKEN-B>
$ sudo warp-cli connect
$ curl http://10.0.3.14:8080/health
{"status":"ok"}
# reached AWS from GCP, and back
```

OUTBOUND ONLY

No inbound holes

Both nodes dial out. Neither cloud opens a port, and there is no tunnel to run.

PRODUCTION

Replicas, not routes

Add replicas sharing one token. Failover is automatic, and the route never changes.

REACH

Subnets behind nodes

A route makes everything behind the node reachable, not just the node.

An agent in one cloud can reach data in another

CLOUDFLARE

An agent calls a private MCP server

The server stays unroutable from the Internet and still gets governed

THE TOOL CALL, OVER THE FABRIC

```
// the server never leaves the private network
const res = await env.PRIVATE_NETWORK.fetch(
"http://mcp.internal:8080/mcp",
{ method: "POST", body: JSON.stringify(call) }
);
```

WHY THIS ONE

Highest value, highest risk

The internal MCP server wraps your production database and your deploy pipeline

Every pattern is the same three steps

1 — Put a node beside it
Container, VM, or CI runner

→

2 — Give it a route or a name
Subnet, or db.internal

→

3 — Dial it from the agent
Mesh IP, hostname, or fetch()

Keep the server private and still see every call it receives

CLOUDFLARE

Run the node where the agent runs

One container image, amd64, and arm64, beside the workload

cloudflare/mesh

one image, amd64, and arm64

DOCKER COMPOSE

Connect a stack

Add one service to compose.yaml, and the whole stack reaches private infrastructure

KUBERNETES

Node or sidecar

Run it as a StatefulSet or as a sidecar in the agent's own pod

CI / CD

Join and leave

The job joins the mesh, tests against private infra, and then exits

Source NAT is built in, so private services reply without route changes

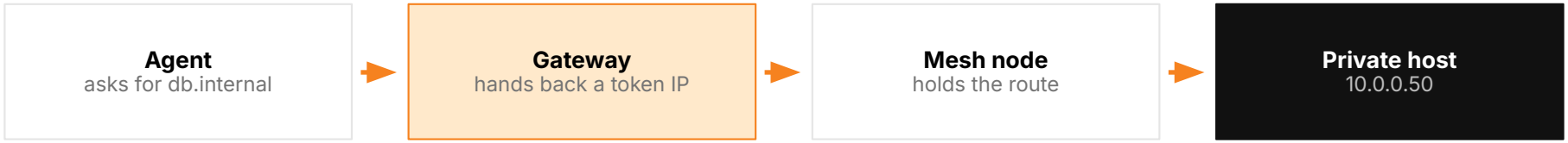
Persist the registration volume for a stable identity; replicas share one token

No host packages, no route-table project: The node lives with the workload

CLOUDFLARE

Give the agent a name, not an IP

One hostname maps to one connector, so the answer is never ambiguous



PRIVATE NAME

Reach an internal service

Works when the private IP is unknown or changes, which is normal in a cloud VPC and fatal for a hard-coded prompt

PUBLIC NAME

Pick the exit point

Send a public hostname out through a chosen node when the destination expects a known egress

Names keep working when the private IP changes

CLOUDFLARE

Source NAT decides whether it works

A connection that opens and then hangs is almost always this

WITHOUT SOURCE NAT

Replies time out

The host gets a packet from 100.96.x.x, has no route back to that range, and the connection hangs

WITH SOURCE NAT

Replies just work

Mesh rewrites the source to the node's LAN IP; the host answers through its own default gateway

WHAT THE PRIVATE HOST SEES

```
without:  src=100.96.4.17  →  10.0.1.20    no route back, dropped
with:     src=10.0.0.9     →  10.0.1.20    replies via default gateway
# per-node setting: off only if you add the return route
```

Nothing else on the private network needs a new route

Most first connections fail on the reply path

CLOUDFLARE

The policy plane is already in the path

Six controls that apply to agent traffic with no new setup

GATEWAY POLICY

Your rules apply

Network and HTTP policies decide what the agent may reach

DLP

Inspect what moves

DLP profiles apply to agent traffic, so a model cannot drain a customer table

IDENTITY

Participants are named

Node traffic is logged as warp_connector@<team>, separate from your people

SESSION LOGS

Every flow is a row

DNS, HTTP, and network logs, pushed to your SIEM like everything else

NARROW SCOPE

Routes, not networks

Advertise only the subnets the agent needs; a node with no routes exposes itself

POST-QUANTUM

On by default

MASQUE uses TLS 1.3 with hybrid ML-KEM, in both directions

Turning a control on is configuration, not a migration

CLOUDFLARE

Start with connectivity

Add the rest of the platform when you want it, not to get started

SASE PLATFORM RAMP Start with Mesh and grow into the rest, one toggle at a time

DAY 1	Private connectivity	Mesh IPs	Device enrollment	
WHEN READY	Gateway policies	DNS filtering	Device posture	Access rules
ADVANCED	DLP	Browser isolation	CASB	SSH/RDP sessions

WHY THIS ORDER

Connectivity is the blocker

It is the only thing an agent project actually stops on; every control after it is a setting

WHAT SECURITY GAINS

A different conversation

They stop arguing about whether to allow the connection and start deciding what it may do

Day one is private connectivity – the platform can wait

AUDIT

Every agent flow is a log row

Which participant, which destination, which policy, how many bytes

GATEWAY ACTIVITY LOG — ILLUSTRATIVE

identity warp_connector@acme...
participant aws-node (Mesh node)
destination 10.0.1.20:5432
action allow policy: agents → db
protocol tcp
bytes 14.2 KB up / 1.1 MB down

ATTRIBUTION

Agent or person

Node traffic carries the service identity; client traffic carries the enrolled user

Ask your team which agent reached which system last Tuesday

A VPN answers “a device connected” – this answers the question

Push it to your SIEM, and agent activity becomes auditable

AVOID THIS

Five mistakes that cost weeks

Each one arrives in the room as “Cloudflare doesn’t work”

✗ **Using Tunnel as a site gateway** A private app behind cloudflared still egresses on its own route	✓ Use Mesh or WAN when the network initiates
✗ **Forgetting the reply path** Private hosts drop traffic they cannot route back to	✓ Keep Source NAT on, or add the route
✗ **Confusing routes with replicas** Longest prefix wins, and no health check runs first	✓ Add replicas to a node, not routes
✗ **Split tunnel excluding Mesh** Nodes need 100.96.0.0/12 through Cloudflare, on MASQUE	✓ Check the device profile first
✗ **Binding a Worker too broadly** cf1:network reaches everything Mesh and WAN reach	✓ Narrow routes or use a service binding

Windows Firewall also blocks inbound 100.96.0.0/12 by default

TRY IT

Build it in 10 minutes

5 steps, then point an agent at it

1. **Turn on settings**
   Enrollment, profile, unique IPs, proxy
2. **Create a node**
   Networking > Mesh, then two commands
3. **Enrol your laptop**
   Cloudflare One Client, then nc -vz the port
4. **Add a route**
   Only for the subnet behind the node
5. **Bind your Worker**
   cf1:network, then fetch() that address

DOCS

Start here

developers.cloudflare.com/mesh

developers.cloudflare.com/workers-vpc

COST

The Free plan is enough

A zero trust organization on the Free plan builds all of this – Workers VPC is free in beta

Connect two participants, then point an agent at one of them

CLOUDFLARE

Agents don’t need a VPN

They need a network

Put a node beside the private resource. Let the agent dial it by name.
Keep the policy, the reply path, and the logs you already have.