The Open Source Path from VPN Sprawl to Zero Trust Access

May 6, 2026 / Gavin Jackson

pangolin zero-trust ztna vpn tailscale headscale twingate kasm teleport access-control network-security

I have been spending time with **Pangolin** recently, using the open source Community Edition as a hands-on evaluation path for a possible production deployment of around 50 users across multiple teams.

That framing matters.

This is not a "global enterprise with thousands of users" deployment story, but it is also not just a hobby network exercise. My current use is small-scale because that is the best way to understand the product properly. The question I am really testing is whether Pangolin has the right shape for a modest production environment where different teams need different levels of access.

The practical questions are:

- Can we expose internal services safely?
- Can we stop opening awkward firewall holes for individual applications?
- Can users get access to the resources they need without receiving broad VPN routes?
- Can different teams be given different access without turning firewall policy into archaeology?
- Can the setup stay understandable enough that we will still trust it six months from now?

So far, Pangolin sits in a very useful middle ground. It is not just another reverse proxy, and it is not just another VPN. It combines identity-aware web access with client-based private access, which makes it feel more like a practical zero trust access layer than a traditional "connect to the network and hope for the best" VPN.

The old VPN habit

Traditional VPNs solved a real problem: "I am outside the network, but I need to reach something inside it."

The classic answer was:

1. Put a VPN concentrator on the edge.
2. Authenticate the user.
3. Drop that user onto an internal IP range.
4. Use firewall rules to decide what the VPN subnet can reach.

That worked well enough when the network perimeter was clear and most important systems lived inside it. But it also created a security habit that has aged badly: once a user is on the VPN, they often become network-adjacent to far more than they actually need.

Even when you tighten the firewall, the model is still subnet-first:

- VPN users can reach this VLAN.
- Admin VPN users can reach that management subnet.
- Contractors get a smaller subnet.
- Exceptions accumulate because one person needs one port on one host.

Over time the firewall rule base becomes a historical document of every temporary access decision that became permanent.

Zero trust networking asks a better question: **what exact resource should this user, role, device, or service be allowed to reach right now?**

That is the shift.

Where Pangolin fits

Pangolin describes itself as an open source, identity-based remote access platform built on WireGuard. The useful part is the combination of two access patterns in one place:

- **Public resources** for browser-based access to web applications.
- **Private resources** for client-based access to specific hosts, ports, or network ranges.

The public resource model is what you would expect from an identity-aware reverse proxy. A user opens a URL, authenticates, and Pangolin brokers access to the backend service. That is a natural fit for things like:

- Kasm Workspaces
- observability dashboards
- internal documentation
- Git services
- internal admin portals
- business web apps that should not be naked on the internet

The private resource model is closer to a zero trust VPN. A user installs a Pangolin client, authenticates, and can then reach only the private resources their identity is allowed to access. This is where SSH, databases, RDP, VNC, internal APIs, or non-HTTP services come into play.

The key difference from a normal VPN is that the unit of access is not "the private network." The unit of access is the resource.

In Pangolin terms, those resources live behind **sites**. A site is the connection into the network where your services live. In the common deployment, the Newt connector runs inside that network and establishes outbound connectivity back to Pangolin. That means the private side can stay behind a firewall, NAT, or CGNAT without exposing inbound ports directly to the internet.

That is the part I like most for production evaluation. It changes the default posture from:

```
Internet -> open port -> service
```

to:

```
Internet -> Pangolin -> authenticated resource -> outbound tunnel -> service
```

The service is no longer the thing negotiating with the open internet. Pangolin becomes the access broker.

Zero trust networking, minus the slogans

Zero trust is an abused phrase, but the underlying idea is very practical.

NIST SP 800-207 frames zero trust as a move away from static, network-based perimeters toward protecting users, assets, and resources. The important bit is that network location should not grant implicit trust. Being "on the LAN" or "on the VPN" should not be enough.

For networking, that translates into a few very concrete design choices.

1. Authenticate before access

The user should prove who they are before the resource is reachable.

With a conventional VPN, authentication gets the user onto the network. After that, many applications still rely on their own credentials, host firewalls, or private IP obscurity. In a zero trust access pattern, the access layer asks:

- Who is the user?
- Which role or group are they in?
- Which device or client are they using?
- Which resource are they trying to reach?
- Is that resource allowed for this identity?

That check happens before the session is established.

2. Authorize the resource, not the subnet

This is where fine-grained access control becomes real.

Instead of allowing a VPN pool to reach `10.10.20.0/24`, you define access to the thing a person needs:

- `proxmox.example.net` for an infrastructure admin role
- `kasm.example.net` for isolated browsing
- `grafana.internal` for an operations role
- `db.internal:5432` for a service or admin role
- `ssh://linux-host` only for the people who administer it

That sounds like a small distinction. It is not. It changes how you reason about risk.

If a laptop is compromised, the attacker does not automatically inherit a flat route into everything behind the VPN. They inherit whatever that identity and device can reach.

3. Make the default deny visible

Good zero trust systems make denial boring.

If a user has no policy for a resource, the answer should simply be no. Pangolin's resource model fits this well: a site being online does not automatically expose the network. An administrator still has to define resources and grant access.

That default matters because it lets you add connectors without accidentally publishing entire environments.

4. Keep access close to identity

Firewall rules are still important, but firewalls do not know enough about human context on their own. A firewall can usually tell you source IP, destination IP, port, and protocol. It may not know that the source represents Gavin on a trusted laptop trying to access Kasm after MFA.

Zero trust access brokers bring that identity context closer to the network decision.

This is why ZTNA products have become popular. The access decision is no longer just "can this IP reach that IP?" It becomes "can this identity reach this resource under this policy?"

5. Reduce lateral movement

The best firewall rule is the one you never had to create.

If users only receive explicit access to specific resources, lateral movement becomes harder. An attacker who lands on one endpoint still has to fight through identity checks, client authorization, resource policy, and whatever application-level controls exist beyond that.

That is not magic. It is blast-radius reduction.

How this simplifies firewall rules

Pangolin does not remove the need for firewalls. I would not trust any product that made that promise.

What it can do is simplify the shape of the firewall problem.

In a conventional remote access design, you often end up with rules like:

```
VPN subnet -> management subnet -> TCP 22, 443, 8006
VPN subnet -> server subnet -> TCP 443, 5432, 3306
VPN subnet -> storage subnet -> TCP 445, 2049
contractor VPN subnet -> app subnet -> TCP 443
```

Then the exceptions arrive:

```
temporary vendor VPN -> one host -> one port
temporary admin VPN -> all lab hosts -> SSH
legacy user VPN -> file share -> SMB
```

With Pangolin, the edge firewall can often become much simpler:

- allow the Pangolin server to receive the public traffic it needs
- allow site connectors to make outbound connections to Pangolin
- keep the private network closed to inbound internet traffic
- allow the connector to reach only the backend targets it is meant to publish

On the inside, you still use host firewalls, segmentation, and egress controls. But you do not have to publish broad VPN routes just so one user can reach one service.

That is the practical win: **fewer network-wide exceptions, more resource-level intent.**

> ***The VPN Support Tax***
>
> *One of the least glamorous costs in remote access is not licensing. It is support.*
>
> *In one environment I worked with, the largest support cost from the MSP was user support around **WatchGuard VPN links**. That is not a criticism of WatchGuard specifically; it is a pattern I have seen across several engagements. Once client VPN becomes the default access method, the operational burden starts to sprawl.*

> *Someone has to:*
>
> - *build VPN profiles for different user groups*
> - *distribute client installers and configuration bundles*
> - *help users through certificate, password, MFA, and profile issues*
> - *diagnose split-tunnel, DNS, route, and local network conflicts*
> - *support client behaviour across Windows, macOS, and mobile devices*
> - *maintain access exceptions as teams and applications change*
> - *remove or rotate access when staff, contractors, or vendors leave*
>
> *That work does not always look dramatic on an architecture diagram, but it shows up in tickets, escalations, delays, and user frustration. It also scales poorly. Each new team, contractor group, or application tends to create another access profile, another firewall exception, or another slightly different set of instructions.*
>
> *The more painful part is that you still do not necessarily get fine-grained access control at the end of it.*
>
> *A traditional client VPN often answers the wrong question. It answers "can this user join this network?" when the better question is "can this user reach this specific resource?" You can tighten that with firewall rules, but then you are paying for complexity in a different place. The VPN profile gets the user near the network, and the firewall rule base becomes the real access policy.*
>
> *That is not sustainable for many small and mid-sized environments.*
>
> *Pangolin and its zero trust peers change the shape of the support problem. You still have administration to do, and private resource clients still need care, but the model becomes more resource-centric:*
>
> - *web applications can be exposed through an authenticated browser path*
> - *private resources can be granted by role instead of broad subnet reachability*
> - *team access can be changed in the access layer rather than by redistributing VPN profiles*
> - *firewall rules can become simpler and more stable*
> - *users can be denied access to everything they were never meant to see*
>
> *That is the operational argument for ZTNA that often lands harder than the security slogan. It is not just "zero trust is more secure." It is "the old VPN operating model is expensive to keep alive, and it still gives you a blunt access control tool."*

Community Edition is a good proving ground

For evaluation, Pangolin Community Edition is the interesting part of the story.

As of 6 May 2026, Pangolin's own documentation says Community Edition is AGPL-3 compliant and includes the core Pangolin features. In practice, that gives you enough to test the access model properly:

- self-hosted control
- public web resources
- private resources through the Pangolin client
- sites and outbound tunnels
- role-based access to resources
- an identity-aware front door for internal services

That is enough to replace a lot of casual VPN usage and, more importantly, enough to validate whether the access pattern fits your environment.

For my purposes, the appeal is that I can think in terms of "which service should be reachable?" instead of "which subnet should I route?" That is a cleaner mental model before you even get to the commercial features.

At around 50 users across multiple teams, the important questions quickly move beyond "does the tunnel work?"

They become:

- How will identity provider integration work?
- Can roles map cleanly to teams and responsibilities?
- How are new devices approved or denied?
- Who can change resources and how is that audited?
- What happens when the access layer is down?
- Can operators see failed resources, unhealthy sites, and policy changes quickly?

That is where the Enterprise Edition feature set becomes much more relevant.

What Enterprise Edition adds

The Enterprise Edition story is worth calling out because Pangolin's Community Edition is not the whole feature set.

As of 6 May 2026, the self-hosted Enterprise Edition is a separate image, distributed under the Fossorial Commercial License, and requires a license key. Pangolin's licensing page says individuals and small businesses below $100,000 USD gross annual revenue can use Enterprise features at no cost, but still need a valid key. Larger businesses need a paid license.

The important point: **Enterprise is not a different core product so much as the operationally mature version of the product.**

The enhanced features that stand out are:

Area	Enterprise capability	Why it matters
Identity	Organization-scoped identity providers, built-in Google and Azure Entra ID options, and multiple roles per user	Better fit for real organizations with multiple teams, tenants, and role mappings
Device trust	Device approvals	Lets admins approve new client devices before they can access resources
SSH	Built-in certificate-based SSH access	Removes long-lived SSH key handling and gives short-lived, identity-bound access
Resource publishing	Wildcard public resources	Useful for Kubernetes ingress, nested reverse proxies, or app platforms with many hostnames
Audit	Action logs for administrative changes	Helps answer who changed what and when
Alerting	Alert rules for sites, resources, and health checks	Turns access infrastructure into something operators can monitor properly
MFA	Organization-level 2FA enforcement for internal users	Helps enforce stronger authentication where users are Pangolin-managed
User experience	Branding and maintenance pages	Useful when users depend on the platform and need clear failure states
Resilience	Clustering for high availability	Required when Pangolin becomes production access infrastructure rather than a single-node convenience

That is a sensible split.

Community Edition gives you the core access model. Enterprise Edition adds the controls that matter when the platform becomes shared infrastructure: stronger identity integration, device governance, auditability, operational alerting, certificate-based SSH, and high availability.

For a small production environment, I would treat those as evaluation criteria rather than nice-to-have extras. Community Edition can prove the access model; Enterprise Edition is where Pangolin starts to look like shared operational infrastructure.

Quick comparison: Headscale, Tailscale, Twingate

Pangolin overlaps with several tools, but it is not a clone of any one of them.

Tool	Best description	Where it shines	Where Pangolin feels different
Headscale	Self-hosted implementation of the Tailscale control server	Self-hosted mesh networking using Tailscale clients	Headscale is about controlling a tailnet; Pangolin is more resource-publishing and identity-aware proxy plus ZTNA
Tailscale	Managed WireGuard-based mesh connectivity platform	Peer-to-peer device connectivity, subnet routing, low-friction setup, excellent clients	Tailscale is brilliant for connecting devices; Pangolin feels more natural when the main job is publishing specific internal apps behind an auth layer
Twingate	Commercial ZTNA platform	Polished enterprise VPN replacement with resource-level access, connectors, device policies, and SaaS management	Twingate is the mature commercial path; Pangolin is attractive when self-hosting and open source control matter

Headscale

Headscale is the right comparison if your main desire is a self-hosted Tailscale-style control plane.

It is excellent when you want control of the coordination server while still using the familiar Tailscale client ecosystem.

That client compatibility is a real advantage. Headscale aims to support recent official Tailscale client releases across the major platforms, so a self-hosted control plane does not necessarily mean giving up the polished commercial Tailscale client experience.

But Headscale is not trying to be an identity-aware reverse proxy for web applications. It is a mesh networking control plane. That makes it a better fit when the problem is "connect my devices securely" rather than "publish these services through authenticated resource policies."

Tailscale

Tailscale is still the benchmark for making WireGuard-based connectivity feel easy.

It gives you a mesh network, strong client support, subnet routers, exit nodes, ACLs, and now grants for more expressive policy. If I want to connect laptops, servers, phones, cloud workloads, and branch resources with minimal fuss, Tailscale is hard to beat.

The trade-off is control plane trust. Tailscale is managed. For most people that is a feature, not a bug. For people who want self-hosted infrastructure as a design requirement, it is a consideration.

Pangolin also feels more app-front-door oriented. If I want `photos.example.net`, `kasm.example.net`, or `admin.example.net` behind login without users thinking about a mesh network, Pangolin maps more directly to that problem.

Twingate

Twingate is probably the closest conceptual competitor for the ZTNA side.

It is resource-oriented, denies access by default, uses connectors in private networks, and gives users access to authorized resources without exposing those resources publicly. It has a strong enterprise story around identity providers, device posture, audit, policy, and operational maturity.

If I were buying for a company that wanted a supported, polished, commercial VPN replacement, Twingate would absolutely be on the shortlist.

Pangolin's appeal is different: open source Community Edition, self-hosting, and a combined public-resource plus private-resource model that works nicely when you want direct control of the access layer.

The Clean Room Pattern: Pangolin, Kasm and Teleport

This is where Pangolin gets especially interesting.

I have written before about building secure cross-domain solutions with Kasm and Teleport, and I also called out Kasm as a practical isolation control in the Essential Eight user application hardening article.

Pangolin fits neatly into that same pattern.

Think about the layers:

- **Pangolin** is the front door for selected services and private resources.
- **Kasm** is the clean room where risky browsing or isolated work happens.
- **Teleport** is the privileged access layer for SSH, Kubernetes, databases, and administrative sessions.

That gives you a powerful architecture:

```
User
 -> Pangolin authenticated public resource
 -> Kasm workspace
 -> isolated browser or desktop
 -> internal app, research target, or admin portal

User
 -> Teleport
 -> short-lived privileged access
 -> SSH, Kubernetes, database, or application session
```

There are a few good ways to combine them.

Put Kasm behind Pangolin

Kasm is browser-delivered, so it is a natural Pangolin public resource.

Instead of exposing Kasm directly, publish it through Pangolin and require authentication at the edge. Then use Kasm's own controls for the workspace:

- no downloads for high-risk browsing

- no uploads where data leakage matters
- no clipboard for sensitive domains
- disposable sessions for unknown sites
- separate workspace images for different trust levels

That gives you a clean split: Pangolin controls who can reach Kasm, Kasm controls what the session can do.

Keep Teleport as the admin lane

Teleport should not disappear just because Pangolin exists.

Pangolin is excellent for brokering network access to resources. Teleport is still stronger when the problem is privileged administrative access with short-lived credentials, session recording, approvals, and deep audit trails.

That is why I would keep Teleport as the admin lane for Linux-heavy environments, as discussed in the restricted administrative privileges article.

For example:

- Pangolin publishes the Kasm manager or internal admin web UI.
- Kasm provides disposable workspaces for risky browsing and separated workflows.
- Teleport brokers SSH, Kubernetes, and database access with stronger privileged access controls.

You do not have to force one tool to do every job. The architecture is cleaner when each layer has a purpose.

Use Pangolin to reduce exposed management surfaces

The best immediate win is boring and valuable: stop putting management UIs directly on the internet.

Put these behind Pangolin:

- Kasm
- Grafana
- Proxmox
- Home Assistant
- Portainer
- internal documentation
- operations dashboards

Then put SSH, Kubernetes, and database administration behind Teleport where session-level audit and privileged access controls matter more.

That is a practical zero trust pattern for a small production environment: publish user-facing management surfaces through Pangolin, keep privileged administration in Teleport, and use Kasm where browser or workspace isolation is the point.

A few cautions

Pangolin is moving quickly, so I would treat dated feature lists with care. The Community versus Enterprise split in this article is based on the published documentation I checked on 6 May 2026.

I would also be careful with raw TCP/UDP public resources. Pangolin's docs note that protocol-aware HTTPS resources can use identity and context policies, while raw public TCP/UDP resources are not protocol aware in the same way. For most non-web services, private resources through the client are the cleaner zero trust pattern.

And finally: do not let the access broker become the new unprotected crown jewel.

For a self-hosted Pangolin deployment:

- patch it
- back it up
- protect the admin account
- enforce MFA where available
- keep the host firewall tight
- monitor access logs
- document what each published resource is for

Zero trust is not a product you install. It is a discipline you make less painful with the right product.

The bottom line

Pangolin is compelling because it gives smaller production environments a practical path away from broad VPN habits.

Community Edition is already enough to make private services cleaner to expose and easier to reason about. For a 50-user multi-team deployment, Enterprise Edition is where the operational story becomes more serious: stronger identity integration, device approvals, SSH certificates, audit logs, alerting, wildcard resources, branding, maintenance pages, and clustering.

Compared with Headscale, Pangolin is less about owning a Tailscale-style mesh control plane and more about publishing resources safely. Compared with Tailscale, it is less device-mesh-first and more application-front-door-first. Compared with Twingate, it is less polished commercial ZTNA and more self-hosted open source access platform.

That is a nice niche.

For a small production rollout, Pangolin gives me the thing I actually want to evaluate: fewer exposed services, fewer broad VPN routes, and a clearer answer to the question "who can reach what?"

References

- Pangolin introduction
- Pangolin resources

- Pangolin sites
- Pangolin system architecture
- Pangolin vs. reverse proxy vs. VPN
- Pangolin Enterprise Edition
- Pangolin identity providers
- Pangolin device approvals
- Pangolin SSH access
- Pangolin action logs
- Pangolin alert rules
- Pangolin clustering
- NIST SP 800-207: Zero Trust Architecture
- Headscale documentation
- Headscale client support
- Tailscale: What is Tailscale?
- Tailscale grants
- Twingate resources
- Twingate architecture