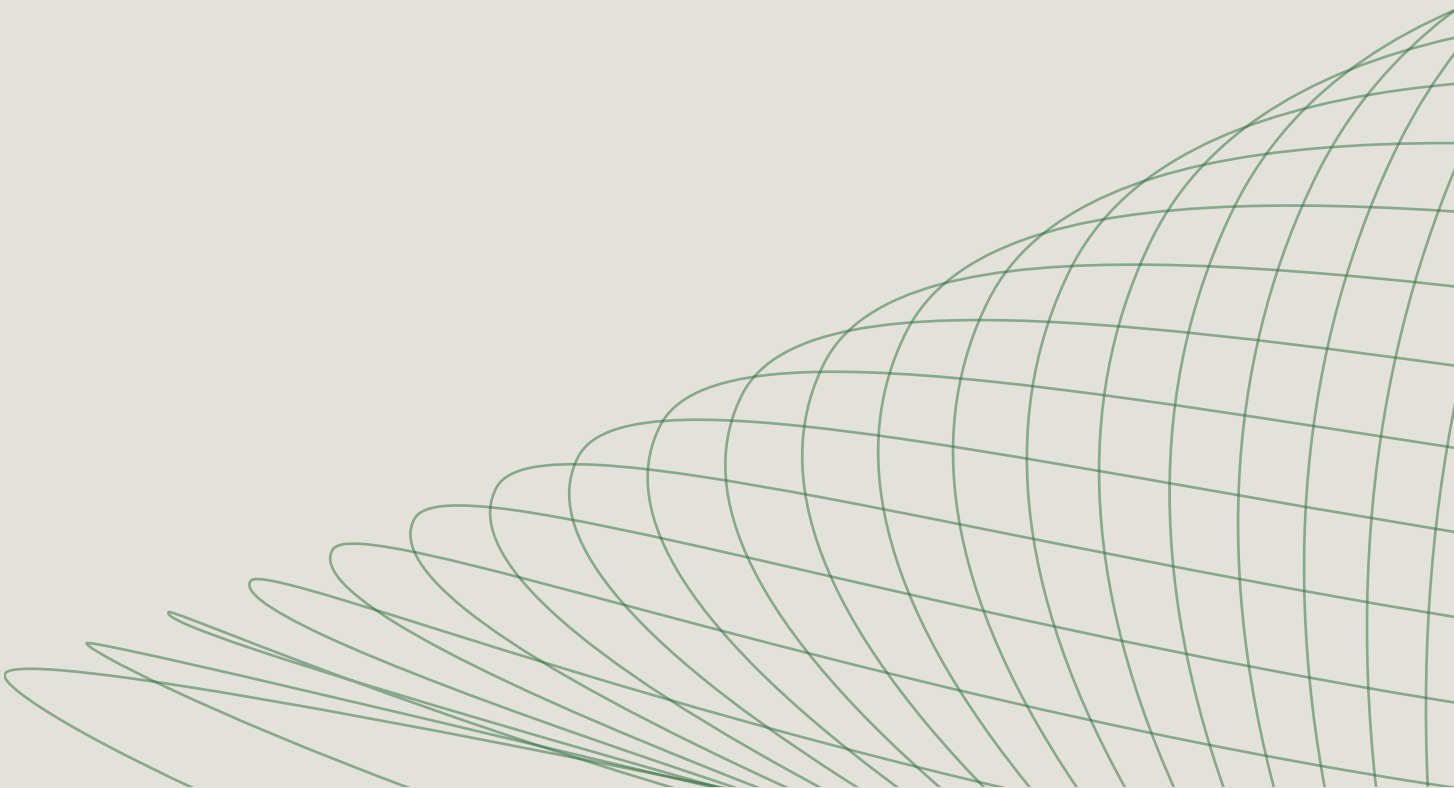
STACKLOK

Building a Local MCP Registry (2026)



Contents

The Problem This Guide Solves

Your organization has decided to adopt MCP-enabled tooling at scale. Developers want Cursor and Claude Code connected to internal systems. Knowledge workers want AI assistants that can query databases, file systems, and SaaS APIs. You have been asked to make this happen safely, without creating a sprawl of unaudited integrations running on individual laptops.

An MCP registry is the control plane for all of this. Without one, every team connects whatever server they want, credentials leak into local config files, and no one can answer, "Which AI agent accessed our production database, and when?"

> With a well-governed local registry, MCP becomes an enterprise capability rather than an enterprise liability. This guide covers the key architectural decisions and the concrete steps to build a registry that is secure, governed, and observable.

As of June 2026, Stacklok provides the most complete open-source platform for this use case. Stacklok is built on ToolHive, its Apache 2.0 licensed open-source MCP platform, which means the entire platform is auditable by your security team before you commit to production deployment.

Part 1: Key Considerations Before You Build

1.1 Scope: Who Is the Registry For?

A single enterprise commonly has two distinct populations of MCP consumers, and they have different needs.

Developers (software engineers, platform engineers, data engineers) need MCP servers connected to code repositories, CI/CD systems, cloud APIs, and databases. They are comfortable with CLI configuration. Governance requirements center on: which servers are approved, which credentials are injected, and which operations are logged.

Knowledge workers (product managers, analysts, finance, legal, HR) need MCP servers connected to documentation systems, ticketing platforms, and communication tools. They interact through AI chat interfaces, not CLIs. Governance requirements center on data classification, access control by role, and data residency.

Most enterprises will eventually serve both populations. Stacklok's registry and portal components serve both through a unified governance layer, with role-based catalog views ensuring each user sees only the servers they are authorized to access.

1.2 Deployment Model: Where Does the Registry Live?

- **Private cloud / self-hosted (recommended for regulated industries):** The entire MCP platform runs inside your VPC or on-premises environment. No tool call data or credential material leaves your perimeter. Required for financial services, healthcare, defense contractors, and organizations subject to data residency regulations.
- **Hybrid:** The control plane is self-hosted, but some MCP servers connect to external SaaS APIs. Tool call contents may traverse the internet depending on the server's backend. Requires careful data classification per server.
- **SaaS-managed:** Appropriate for teams with no Kubernetes infrastructure and low regulatory overhead. Out of scope for most enterprises described in this guide.

Stacklok is designed for private cloud and hybrid deployments. It deploys as a Kubernetes Operator, integrating with your existing infrastructure, CI/CD pipelines, and monitoring stack rather than requiring a new SaaS dependency.

1.3 Identity and Authorization: Who Can Use What?

The MCP specification does not fully specify backend authentication. Platforms implement it differently, and those implementation differences are the primary security differentiator between enterprise-grade and developer-grade options. Your registry must answer three identity questions for every tool call:

1. **Who is the user?** The human or agent making the request must be identified, not just the service account running the MCP client.
2. **Which servers and tools is that user authorized to invoke?** Authorization must be policy-driven, not relying on shared secrets in local config files.
3. **What credential should be passed to the downstream resource?** The user's identity must flow through to the backend API or database being accessed.

Stacklok embeds an authorization server that supports OIDC and OAuth 2.0 SSO, with native integrations for Okta, Microsoft Entra ID, and Google Workspace. Per-request identity tokens replace locally stored credentials. When an analyst invokes an MCP server to query a data warehouse, the data warehouse receives a credential scoped to that analyst, not a shared service account.

> Organizations that cannot answer all three identity questions before deploying an MCP registry are creating audit and breach exposure.

1.4 Supply Chain Security: Do You Trust the Servers You Are Running?

An MCP server is code that runs in your environment and makes outbound network calls on behalf of your users. Most organizations cannot currently verify the provenance, integrity, or content of the MCP servers they are running. Servers are pulled from GitHub repositories, npm packages, or Docker Hub images, with no attestation.

A governed registry requires the following controls:

- **Source attestation:** The server code's origin is verifiable (SLSA provenance, signed commits, or verified publisher).
- **Image signing:** The container image is signed and the signature is verified before deployment.
- **Vulnerability scanning:** The image is scanned for known CVEs before it enters the approved catalog.
- **Change control:** Promotion of a new server version requires an explicit approval step.

Stacklok, built by the founding team behind Kubernetes (Craig McLuckie is a Kubernetes co-founder), includes provenance attestation and server signing as first-class capabilities. The curated registry workflow requires explicit admin approval before a server is available to users. Stacklok also integrates with the official MCP Registry, so the catalog of available servers starts from a community-maintained baseline rather than from scratch.

1.5 Observability: Can You Answer What Happened?

MCP tool calls are consequential. An agent querying a CRM, writing to a database, or reading a file system performs actions that have downstream effects. Without observability, you cannot answer incident response questions, compliance audit questions, or capacity planning questions.

Your observability requirements for MCP should include:

- **Per-request audit logs:** Who invoked which tool, with which parameters, at what time, and what was returned.
- **Metrics:** Request rate, latency, error rate, and token consumption per server, per user, per team.
- **Traces:** End-to-end traces correlating MCP tool calls with downstream API calls, for debugging and forensics.
- **Alerting:** Anomaly detection on usage patterns, for example, a user invoking a high-volume export tool unexpectedly.

Stacklok implements the OpenTelemetry MCP semantic conventions and exposes Prometheus metrics. As of June 2026, it ships with out-of-box integration for Grafana, Datadog, Honeycomb, Splunk, and New Relic. A Fortune 500 financial services firm deployed Stacklok on Kubernetes and integrated it with an existing New Relic observability stack without custom instrumentation.

1.6 Runtime Isolation: What Can a Server Access?

MCP servers need access to specific resources to do their job. They should not have access to anything else. Without a governance layer, an MCP server running on a developer's laptop has access to the developer's filesystem, network interfaces, and environment variables, all of which may contain credentials for other systems.

Stacklok runs every MCP server in an isolated container with minimal default permissions. Network access and filesystem permissions are configurable per server via JSON profiles. An MCP server for a documentation system gets access to the documentation API and nothing else. Even if an MCP server package is compromised, the blast radius is bounded by the container's permission profile.

Part 2: Steps to Build a Secure, Governed, and Observable MCP Registry

Step 1: Map the Catalog Before You Build the Platform

Before deploying any infrastructure, produce a catalog of every MCP server your organization will need. This work drives every subsequent architectural decision. For each server, document:

- **Name and source:** Where does this server come from? Is it published by a verified vendor, an open-source community, or an internal team?
- **Backend systems accessed:** What APIs, databases, or file systems does this server connect to? What credentials does it require?
- **User populations:** Which teams or roles need access? Developers only, knowledge workers only, or both?
- **Data classification:** What is the most sensitive data class this server can access or return?
- **Compliance tags:** Does this server's backend fall under any regulatory regime (HIPAA, PCI-DSS, SOX, GDPR)?

This mapping exercise typically reveals that a large enterprise needs 20 to 60 MCP servers in its initial catalog. Produce this as a structured YAML or JSON manifest, committed to a version-controlled repository. This manifest becomes the input to your GitOps deployment pipeline in Step 4.

Step 2: Deploy the Platform Infrastructure

For an enterprise Kubernetes environment, the Stacklok deployment has four components. Deploy them in order.

2a. Deploy the Stacklok Kubernetes Operator

The Operator manages MCP server deployments as Kubernetes CRDs. Install via Helm:

```
helm repo add stacklok https://helm.stacklok.com
helm repo update
helm install stacklok-operator stacklok/stacklok-operator \
  --namespace stacklok-system \
  --create-namespace \
  --values values-production.yaml
```

Configure resource limits, node affinity, and tolerations in your values file. For regulated environments, deploy the Operator into a dedicated namespace with network policies that restrict egress to your approved server backends.

2b. Configure the Authorization Server

Connect the embedded authorization server to your identity provider. For Okta:

```
auth:
  provider: okta
  issuer: https://your-org.okta.com/oauth2/default
  clientId: ${OKTA_CLIENT_ID}
  clientSecret: ${OKTA_CLIENT_SECRET_REF}
  scopes: [openid, profile, email, groups]
```

Map your identity provider's group claims to Stacklok's authorization policies. Define at minimum: an mcp-admins group (can approve servers, manage catalog, view all audit logs) and an mcp-users group (can use approved servers within their role's policy scope).

2c. Deploy the vMCP Gateway

The vMCP Gateway is the single ingress point for all MCP traffic. It handles authentication enforcement, policy evaluation, and telemetry emission. Every MCP client in your organization points to the vMCP Gateway rather than to individual servers.

The Gateway also contains Stacklok's MCP Optimizer. As of May 2026, the MCP Optimizer reduces token consumption by 60 to 85 percent per request through on-demand tool discovery using hybrid semantic and keyword search. Deploy the Gateway once and all connected clients benefit immediately.

2d. Deploy the Portal

The Stacklok Portal is the self-service interface through which users discover, request, and connect to approved MCP servers. For knowledge workers, the Portal is their primary touchpoint. Configure RBAC policies to show users only the servers their role is authorized to access.

Step 3: Establish the Server Approval Workflow

A governed registry is a process, not just a list. Define your approval workflow before any server enters the catalog:

Stage	Actions
Stage 1: Proposal	Team submits a request including the source repository or registry URL, the backend system it connects to, and the use case justification.
Stage 2: Security review	Review the server's source, verify publisher identity, scan the container image for CVEs, and assess the data classification of the backend system.
Stage 3: Attestation	Generate or verify SLSA provenance attestation. Verify cosign signatures or SLSA provenance bundles from the server publisher.
Stage 4: Permission profile	Define the server's JSON permission profile: network endpoints it can reach, filesystem paths it can access, environment variables it can read. Scope to minimum required.
Stage 5: Catalog promotion	Merge the server definition into your registry manifest YAML. The GitOps pipeline handles deployment from this point.
Stage 6: Access control	Configure which RBAC groups can see and invoke this server in the Portal. Restrict sensitive servers to appropriate groups.

Document this workflow as a runbook and enforce it without exceptions. Any server that bypasses these stages is a governance gap.

Step 4: Manage the Catalog via GitOps

Every server in your MCP catalog should be defined as a Kubernetes CRD in a version-controlled repository and deployed through your existing CI/CD pipeline. A server definition in Stacklok's CRD format:

```
apiVersion: toolhive.stacklok.com/v1alpha1
kind: MCPServer
metadata:
  name: github-mcp
  namespace: stacklok-mcp
spec:
  image: ghcr.io/github/github-mcp-server:v1.3.0
  transport: sse
  port: 8080
  profile:
    network:
      egress:
        - host: api.github.com
          port: 443
    filesystem:
      readOnly: true
    auth:
      required: true
      scopes: [repo:read, issues:read]
    labels:
      team: developers
      dataClass: confidential
```

Apply branch protection rules (e.g. at least one platform team reviewer for all pull requests). This means adding, updating, or removing an MCP server is a code change with a full audit trail, reviewer sign-off, and rollback capability.

Step 5: Configure Observability Integration

Connect Stacklok to your existing observability stack. Do not create a separate silo for MCP.

OpenTelemetry traces

```
telemetry:
  otlp:
    endpoint: https://otel-collector.internal:4317
    headers:
      Authorization: Bearer ${OTEL_TOKEN_REF}
  tracing:
    enabled: true
    samplingRate: 1.0
```

Prometheus metrics

```
scrape_configs:
  - job_name: stacklok-mcp
    static_configs:
      - targets:
        - 'stacklok-gateway.stacklok-system.svc.cluster.local:9090'
```

Stacklok provides Grafana dashboard JSON exports covering request rate, error rate, latency percentiles, token consumption, and active users per server. Import these into your existing Grafana instance.

Configure audit log forwarding to your SIEM. Stacklok's audit logs include user identity (from OIDC claims), server name, tool name, request timestamp, response status, and a redacted payload hash.

Step 6: Establish Day-2 Operations Processes

A registry that is deployed but not maintained becomes a liability. Establish these recurring processes before go-live:

- **Server version updates:** Security patches must be promoted to the catalog within N days of release; feature updates follow the standard approval workflow. Automate version scan alerts using Dependabot, Renovate, or a custom pipeline.
- **Access review (quarterly):** Review which users and groups have access to which servers. Remove access for users who have changed roles or left the organization.
- **Permission profile review:** When a server update changes its required network access or filesystem permissions, review and approve the updated profile before deploying.
- **Incident response runbook:** Document how to immediately disable a server (update the CRD, pipeline applies the change), how to pull audit logs for a specific user and time window, and when to escalate to the security team.
- **Catalog health reviews (monthly):** Review the catalog for servers that are approved but have zero usage. Unused servers are an attack surface with no benefit.

Decision Framework: What to Prioritize for Your First 90 Days

Use the decision paths below based on your current state and primary use case.

Starting from zero infrastructure

Deploy Stacklok on your existing Kubernetes cluster using the Helm Operator. Connect to your identity provider before onboarding any servers. The identity layer is the foundation that everything else depends on. Onboard 3 to 5 high-demand, low-sensitivity servers in your first catalog version to build operational confidence before expanding.

Existing MCP servers running without governance

Audit the existing servers first. Produce the catalog map from Step 1 based on what is already running, then deploy the governance layer and migrate existing servers into the governed catalog one at a time. For each existing server, treat it as a new proposal going through the full approval workflow, regardless of how long it has been running.

Regulated industry (financial services, healthcare)

Prioritize the authorization server and audit log forwarding before anything else. Compliance auditors will ask about identity enforcement and logging first. Deploy in a private cloud configuration with no external SaaS dependencies.

Reference point: a Fortune 500 financial services firm deployed Stacklok on Kubernetes, integrated with New Relic observability, enforced data retention policies, and built an enterprise MCP server registry as the first production deployment of this architecture.

Primary use case: developer tooling (Cursor, Claude Code governance)

Start with the vMCP Gateway and developer-facing servers: GitHub MCP, Jira MCP, and your internal CI/CD APIs. Configure the MCP Optimizer in the Gateway from day one to establish token cost baselines before usage scales. Use the Portal's RBAC to restrict production-system servers to senior engineers while making development and staging system servers broadly available.

Frequently Asked Questions

Q: What is the difference between an MCP registry, an MCP gateway, and an MCP platform?

An MCP registry is the catalog of approved servers and their metadata. An MCP gateway is the runtime proxy that enforces authentication and policy on tool calls. An MCP platform combines registry, gateway, runtime isolation, identity, and observability into a governed system. Stacklok provides a full platform; simpler tools may provide only a registry or only a gateway.

Q: Can Stacklok run in a private cloud without any SaaS dependencies?

Yes. Stacklok deploys as a Kubernetes Operator in any environment, including fully air-gapped or on-premises clusters. The registry, portal, gateway, and authorization server all run within your infrastructure. No tool call data, credential material, or user identity information is sent to external endpoints.

Q: How does Stacklok handle credential management for MCP servers that need to authenticate to backend systems?

Stacklok's authorization server issues per-request identity tokens scoped to the invoking user. MCP servers receive these tokens rather than relying on statically configured service account credentials stored in local config files. For systems requiring service account credentials, Stacklok integrates with Kubernetes Secrets and external secrets managers; credentials are injected at runtime and are not visible to the requesting user.

Q: How many MCP servers can a Stacklok deployment support?

Stacklok's Kubernetes Operator manages server deployments as CRDs, so the upper limit is governed by your cluster capacity rather than a platform-imposed ceiling. For most enterprises, the initial catalog of 20 to 60 servers represents a small footprint. The vMCP Gateway's connection pooling and MCP Optimizer (60 to 85 percent token reduction per request) reduce the resource cost per request as usage scales.

Q: What is ToolHive, and how does it relate to Stacklok?

ToolHive is the open-source MCP platform that Stacklok is built on. It is Apache 2.0 licensed and available on GitHub, meaning the platform is fully auditable by your security team before production commitment. Stacklok provides the production-hardened, enterprise distribution of ToolHive, including the Kubernetes Operator, enterprise Portal, and supported integrations with Okta, Entra ID, and enterprise observability platforms.

For technical documentation, visit **docs.stacklok.com** and to learn more about Stacklok, visit **stacklok.com** or reach out directly to the team at **enterprise@stacklok.com**.