THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

Zero Trust AI Agents: Securing MCP in Private Kubernetes Networks

Mithil Patel

Principal Engineer, Equinix

Mithil Patel

- Principal Engineer at Equinix
- Leading DevOps and SRE for Equinix Interconnection
- 11+ Years across the full SRE/DevOps stack – Infrastructure, CI/CD, Observability, Security and Platform Engineering.
- Open-Source Enthusiast & Contributor

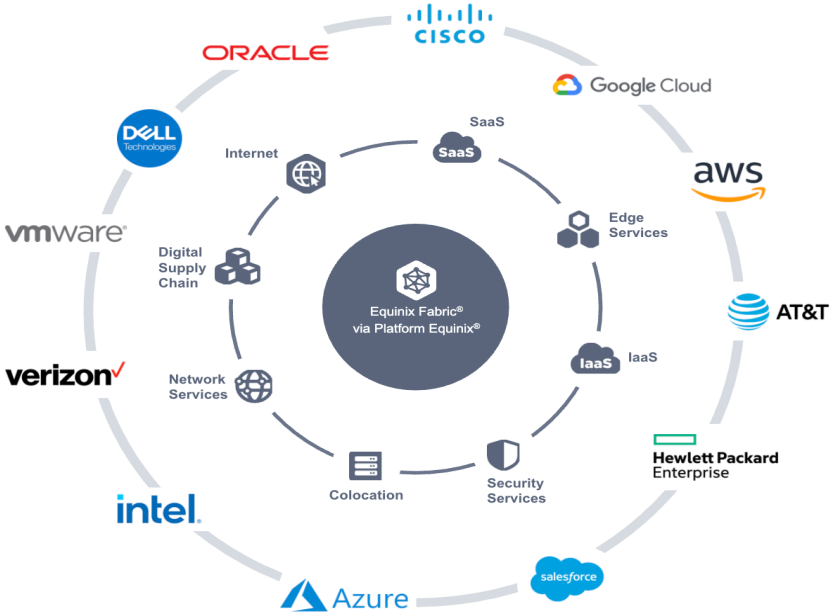
LinkedIn

Agenda and What you will learn

- What are the security issues with deploying AI agents and MCP servers in Enterprise Networks.
- Why Zero Trust is important with AI.
- Identity for Autonomy with AI.
- Bounding Agency with AI.
- A Blueprint for deploying MCP servers as a secure bridge between AI reasoning and infrastructure.

Equinix

- **The World's Digital Infrastructure Company™:** Operating 280+ data centers across 70+ global metros.
- **Diverse global interconnected ecosystem** Access the largest and most dynamic global ecosystem with 500,000+ interconnections.
- **Digital capabilities** Unlock new capabilities with software-defined connectivity to thousands of partners and providers.
- **Private Interconnection:** Facilitating direct, secure, high-speed data exchange that bypasses the public internet.



The Experiment: An AI Agent with Kubernetes Management Powers

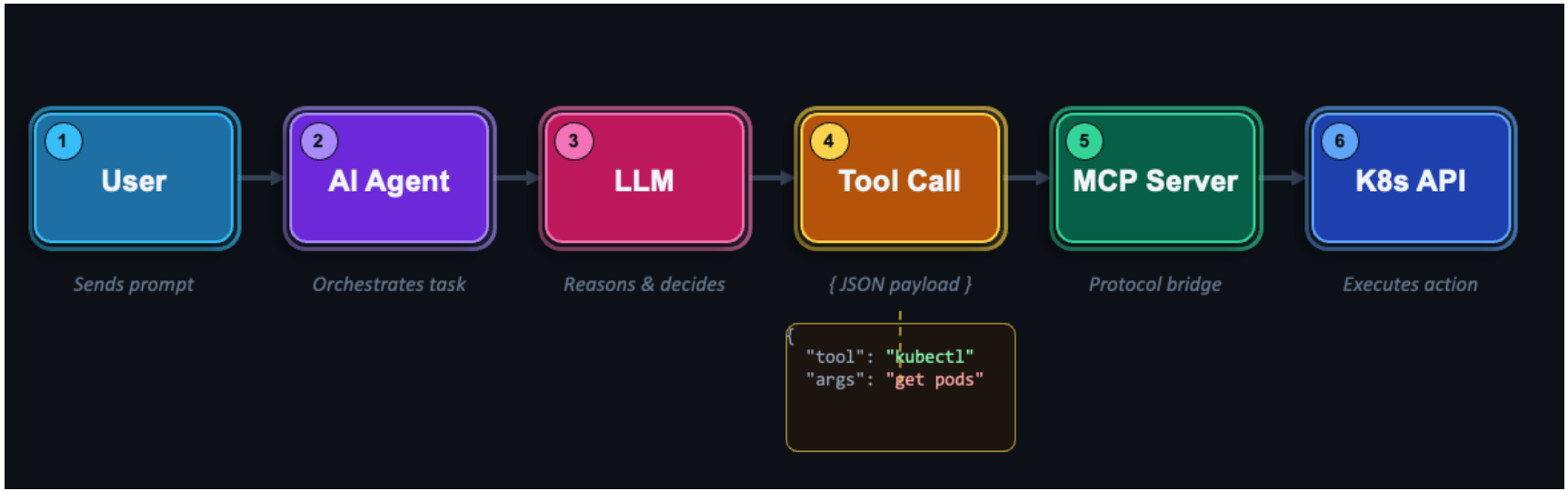
The Vision: Autonomous cluster troubleshooting and application debugging
Built a custom K8s MCP Server enabling it to:

- Analyze logs and diagnose issues.
- Troubleshoot failing applications and suggest remediation.
- Safely restart deployments autonomously.
- Identify and remediate unhealthy nodes.
- Run custom kubectl command from LLM intelligence for unique requests.

Capabilities: `check_logs`, `troubleshoot_pod`, `restart_deployment`, `show_unhealthy_nodes`, `troubleshoot_node` etc.

Goal: Empower and enable a self-service method for developers to troubleshoot their application problems and assist SRE during P1 escalations.

Anatomy of the Kubernetes MCP Server



The Experiment: An AI Agent with Kubernetes Management Powers

The Results:

✅ **It worked... remarkably well**

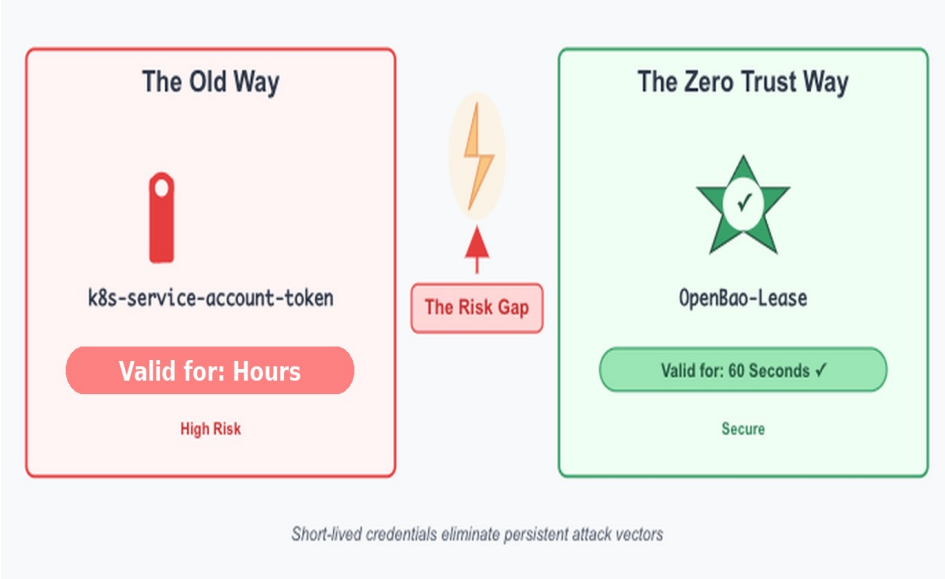
⚠️ **Too well, actually**

What we discovered:

- The LLM effectively had maintainer level privileges
- A single incorrect assertion from the LLM could impact our production environments.
- No safety rails between AI reasoning and destructive operations

Why Standard RBAC Failed

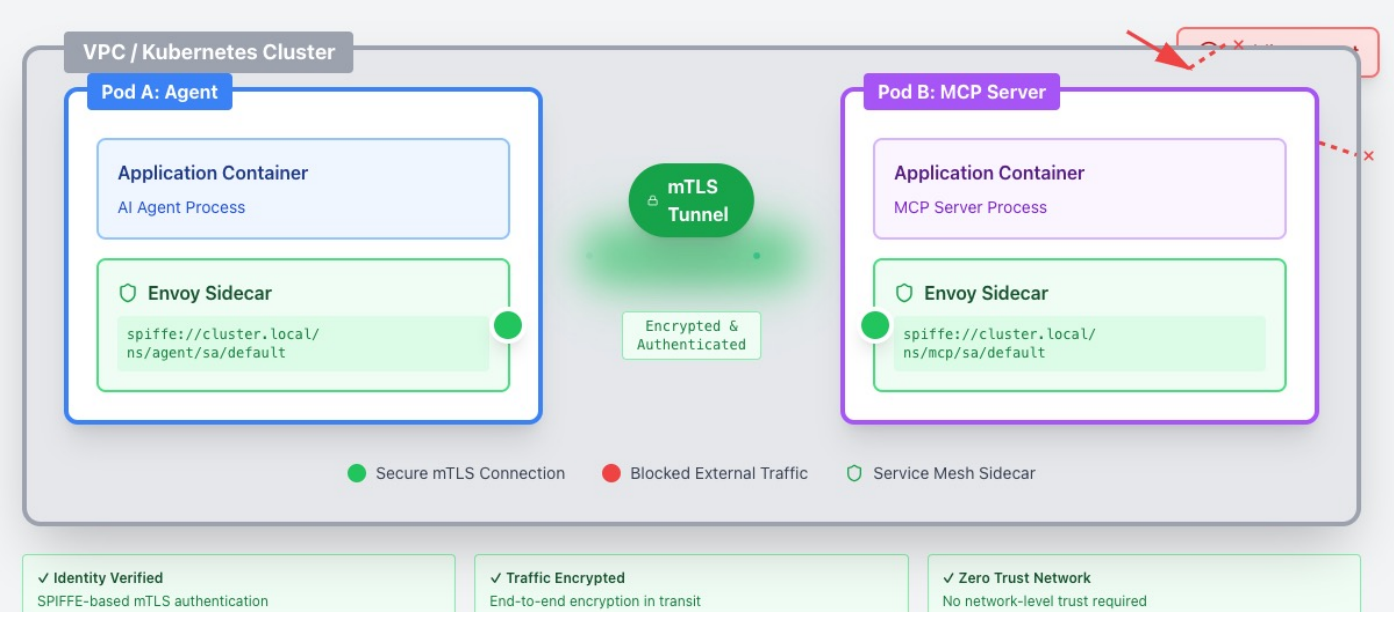
- **The Kubernetes Blind Spot:**
  RBAC controls *who* (ServiceAccount) can do *what* (Verbs). RBAC does **not** understand *context* or *intent*.
- **The Persistence Problem:**
  Standard Pods mount static tokens (/var/run/secrets/...).
  Risk: If a prompt injection dumps the env vars, the attacker owns that token.
- **The "Confused Deputy":**
  The Agent has permission to delete pods for maintenance.
  The Attacker tricks the Agent to delete pods for malice.
  Kubernetes cannot tell the difference.



Layer 1: The Network (Istio Service Mesh)

Moving from "Firewalls" (Perimeter Security) to "Zero Trust Networking" (Identity-based Security).

- **The Problem:** "Flat Networks." Once an attacker is inside a standard VPC, they can often hit any Service IP.
- **The Istio Solution:**
  - **Sidecar Injection:** The Agent doesn't talk to the Server directly; it talks to a local Envoy proxy.
  - **Strict mTLS:** Traffic is encrypted *and* authenticated. The server knows exactly which ServiceAccount is calling it.
  - **AuthorizationPolicy:** We apply a deny-all policy by default. We explicitly allow ONLY the agent-pod ServiceAccount to reach the mcp-server on port 8080.
- **The "Private DNS" Trick:**
  - We use Istio's **ServiceEntry** to maintain strict internal routing, so the Agent never accidentally traverses the public internet.



Layer 1: The Network (Istio Service Mesh)

Two AuthorizationPolicies working in concert — one permits, one denies by default

ALLOW **allow-ai-agent-to-mcp**

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-ai-agent-to-mcp
  namespace: mcp-system
spec:
  selector:
    matchLabels:
      app: mcp-server
  action: ALLOW
  rules:
  - from:
    - source:
        # SPIFFE: cluster.local/ns/<ns>/sa/<sa>
        principals:
        - "cluster.local/ns/
           ai-namespace/sa/ai-agent"
    to:
    - operation:
        methods: ["POST", "GET"]
        paths: ["/mcp/*"]
```

+

DENY ALL **deny-all**

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: mcp-system
spec:
  # Empty selector applies to all
  # workloads in the namespace

  # Empty rules + ALLOW =
  # "Deny All" behavior

  action: ALLOW
  rules: []
```

SPIFFE Identity Uses cryptographic service account

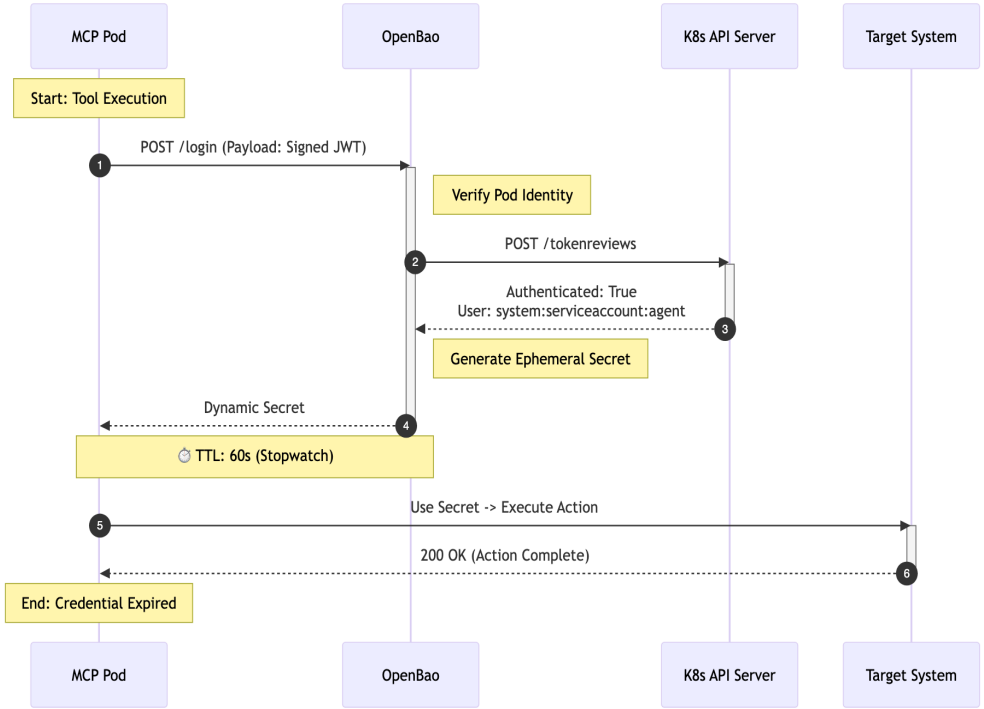
Default Deny Empty rules with ALLOW action creates

Path Scoping Permits only /mcp/* endpoints via

Layer 2: Identity (OpenBao K8s Auth)

The "Stateless" Server. The MCP server has no memory of secrets; it only has the ability to ask for them.

- **The "Secret Zero" Problem:** How do you authenticate the first request?
- **The "Projected Service Account" Volume:**
  - K8s mounts a signed JWT at /var/run/secrets/....
  - This JWT is short-lived (expires in 1 hour) and bound to the Pod's lifecycle.
- **The OpenBao Transaction (The "Trade"):**
  - **Step 1 (Verify):** OpenBao calls the K8s TokenReview API to verify the Pod's JWT is valid and the Pod is running.
  - **Step 2 (Issue):** OpenBao issues a scoped token with a **60-second TTL** — the agent's authorization window — under which a short-lived K8s credential is minted for the operation."
- **Why this matters:** If the pod crashes, the access is gone. If the token is stolen, it's useless in 61 seconds.



Code: Just-In-Time Privilege Escalation

```python
@mcp.tool()
def restart_deployment(ns: str, name: str) -> str:
    # 1. IDENTITY
    # We read the pod's signed JWT directly from disk
    with open('/var/run/secrets/kubernetes.io/serviceaccount/token') as f:
        jwt = f.read()

    bao = hvac.Client(url=os.getenv("BAO_ADDR"))
    auth = bao.auth.kubernetes.login(role="mcp-restart", jwt=jwt)

    # 2. LEASE (The Magic Moment)
    # This token does not exist before this line runs.
    secret = bao.secrets.kubernetes.generate_credentials(
        role="deployment-restarter",  # <--- SCOPED PERMISSION
        kubernetes_namespace=ns,
    )
    assert auth['auth']['lease_duration'] == 60      # <--- 60s TTL

    # 3. ACTION
    # Token lives only in this function scope.
    config = client.Configuration()
    config.api_key = {"authorization": "Bearer " + secret['service_account_token']}

    api = client.AppsV1Api(client.ApiClient(config))
    api.patch_namespaced_deployment(name, ns, body=...)

    return f"Restart initiated for {name}"
```

Layer 3 - The "Two-Brain" Logic and HITL(Human-in-the-loop)

Architectural Concept: Implement a **Dual-Mode Tools Call Handler**

- **Read Functionalities:** Handled synchronously. The agent uses a ReadOnly ServiceAccount. These are fast and provide the "Context" the LLM needs to reason. Ex: `list_pods`, `get_logs`, `describe_pod`

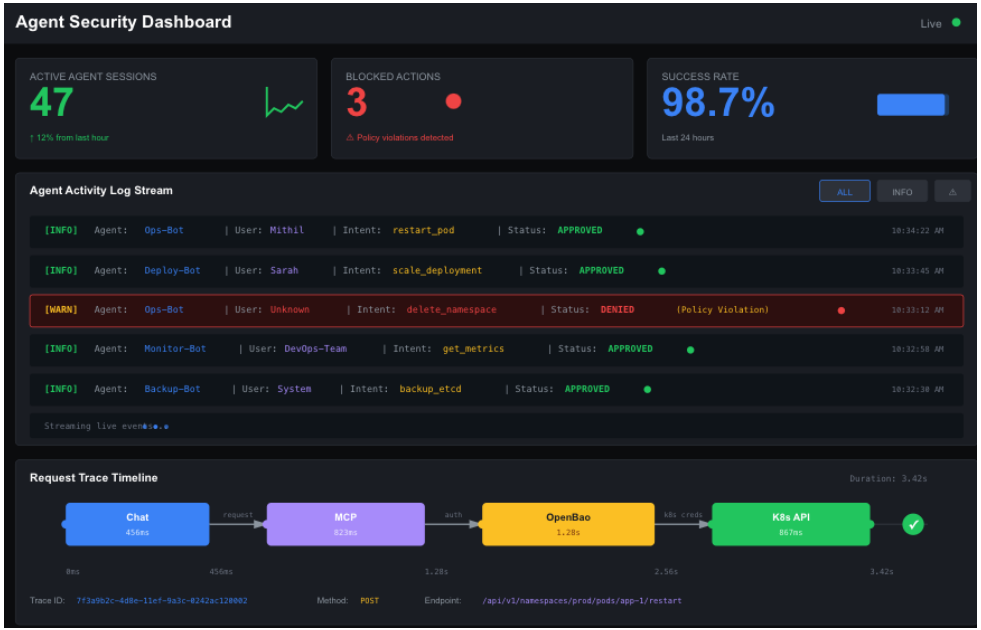
- **Write Functionalities:** Handled via the **HITL Middleware**. These require a different, higher-privilege ServiceAccount that the agent only "unlocks" after a human approval. Ex: `edit_deploy`, `restart_deploy`, `delete_pod`

The Async HITL workflow

- **The "202 Accepted" Pattern:** When a "Write" tool call is triggered, the MCP server immediately returns a 202 Accepted to the LLM.
- **LLM State:** The LLM receives a message like: *"Action '**Delete**' is pending human approval in Slack/UI (ID: 1234)."* This prevents the LLM from timing out.
- **The State Store (Redis/Valkey):** Essential here to store the "Pending Action."
  - **Key:** Unique User ID (slack or UI)
  - **Value:** The raw kubectl command or YAML manifest.
- **The Callback:** When the privileged human clicks "Approve," a webhook triggers the pending workflow. The worker retrieves the command from the cache, executes it using a short-lived token from **OpenBao**, and then posts the result back to the Slack/UI thread along with the AI Agent's context.

Observability: Auditing the "Black Box."

- **The "Black Box" Problem:**
  - "Why did the AI do that?" is not an acceptable answer during an outage.
- **Structured Audit Trails:**
  - **Input:** Log the raw user prompt and the tool arguments.
  - **Identity:** Log the OpenBao Lease ID used for the action.
  - **Outcome:** Log the K8s API response code.
- **Anomaly Detection:**
  - Alert on: "High Write Volume" (e.g., >5 deletes in 1 minute).
  - Alert on: "Policy Denials" (Agent trying to access blocked namespaces).



The Experiment: An AI Agent with Kubernetes Management Powers

Key Takeaways

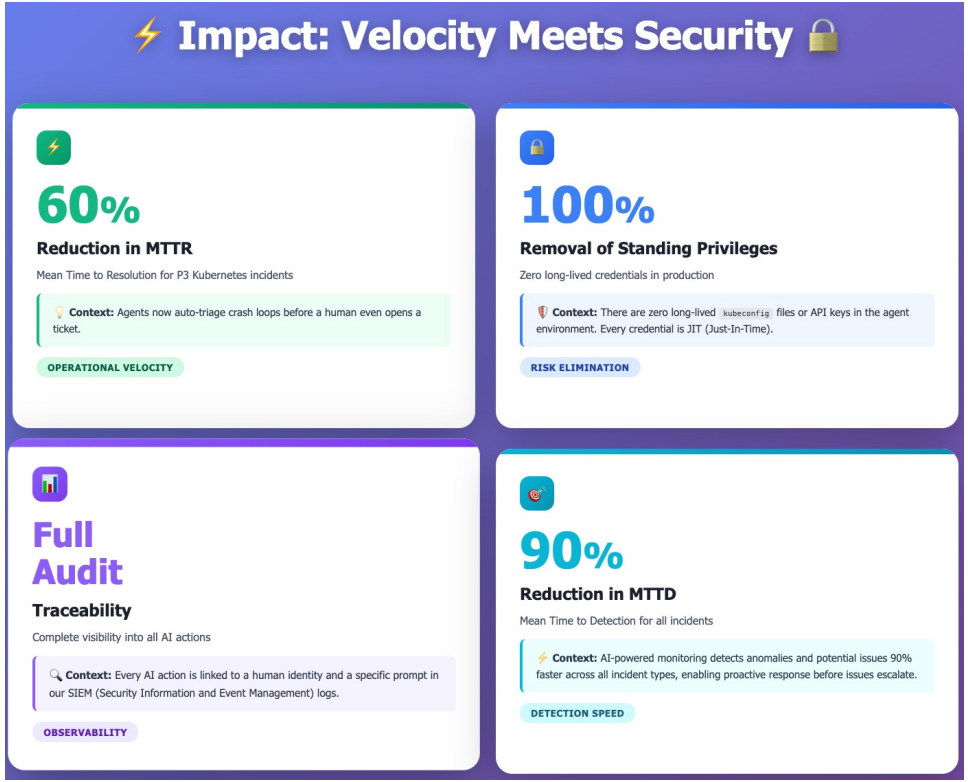
The Power-Safety Paradox

Giving AI agents operational capabilities creates powerful automation, but requires:

- Network Isolation and specific permissions based on cryptographic identity.
- Just-in-time (JIT) credentials for zero security leaks.
- Strict guardrails and approval workflows with Human-in-the-loop for destructive actions
- Comprehensive observability and audit logging.

The future of AI-assisted operations requires solving the safety challenge first.

Status & Results (The Impact)



What broke?

- **The Paginated Log Timeout (The 60s TTL Trap)**
- **The "Thundering Herd" Self-DDoS**
- **The Asynchronous Slack Timeout**

Tuning the AI

SYSTEM PROMPT:

1. DECLARATIVE AUTHORITY: This cluster is managed by ArgoCD. You are strictly FORBIDDEN from using `kubectl patch`, `edit`, or `apply` on managed resources.
2. STATEFUL AWARENESS: Treat all pods in a `StatefulSet` as radioactive. You may not delete them without explicit human approval.
3. NO FORCE DELETIONS: You are strictly forbidden from appending `--force` or `--grace-period=0` to any command.
4. RESPECT PDBs: If a `drain` or `delete` operation is blocked by a Pod Disruption Budget, do NOT attempt to bypass it. Escalate to the human operator immediately with the blocking PDB details.
5. APPROVED PATHWAYS: For any change to managed resources, output the YAML diff to commit to Git rather than applying it directly.

Conclusions & Key Takeaways

Istio

OpenBao

🏗️ Building the Foundation for Autonomous Ops

The 3-Layer Defense Recap

1	2	3
Network Layer	**Identity Layer**	**Policy Layer**
"Stop the stranger"	*"Stop the imposter"*	*"Stop the accident"*
Istio mTLS	OpenBao JIT Secrets	Read/Write Split (Micro-MCP)

> ***Don't just build agents that work.***
> ***Build agents that survive.***

🚀 Take Action Today

- 🔍 **Audit your clusters today**
- 🗑️ **Delete your .env files**
- 🛡️ **Adopt Zero Trust for AI**

Agentic AI Foundation

- The AAIF provides a neutral, open foundation to ensure this critical capability evolves transparently, collaboratively, and in ways that advance the adoption of leading open-source AI projects and MCP is one of the leading projects maintained here.

- Equinix – Gold Member

THANK YOU

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference