LLMs on Autopilot: Running AI Agents on Kubernetes With Open Source Tools

Annie Talvasto

Sr. Manager, Product Marketing & Community

Upbound

Infrastructure & Development in the Age of AI

What are AI agents?

- Agents are AI systems that don’t just assist humans reactively. They plan, decide, and act autonomously toward goals.
- Agents can analyze a request, determine the need for a database, and (attempt) to provision it without human intervention.
- To function effectively, agents need structured, unified access to all operational inputs: documentation, policies, infrastructure APIs.

Agent Examples

SRE Agent

Monitors system health, detects anomalies, and automatically remediates incidents to maintain reliability and uptime.

Provisioning Agent

Automates infrastructure provisioning and configuration based on high-level goals or requests.

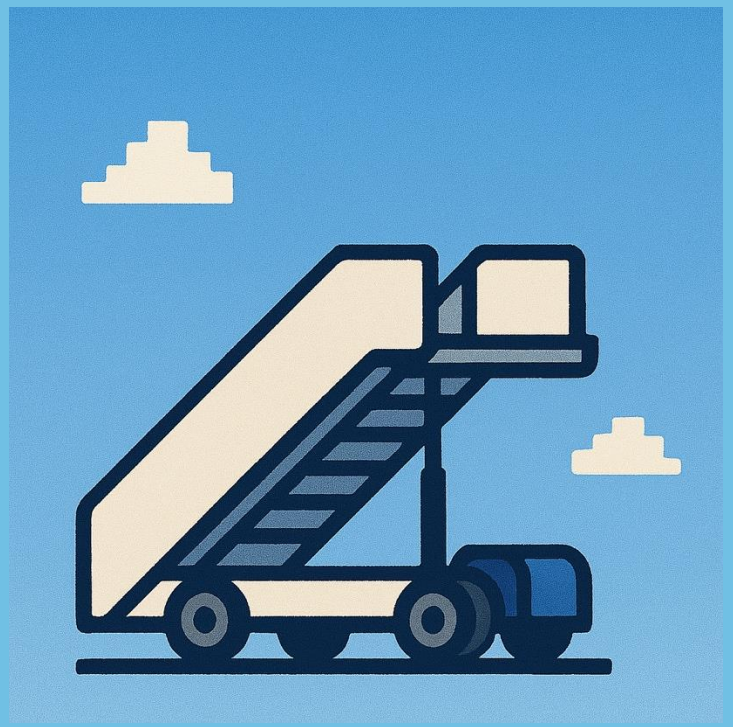
Compliance Agent

Enforces security policies and compliance requirements across cloud workloads

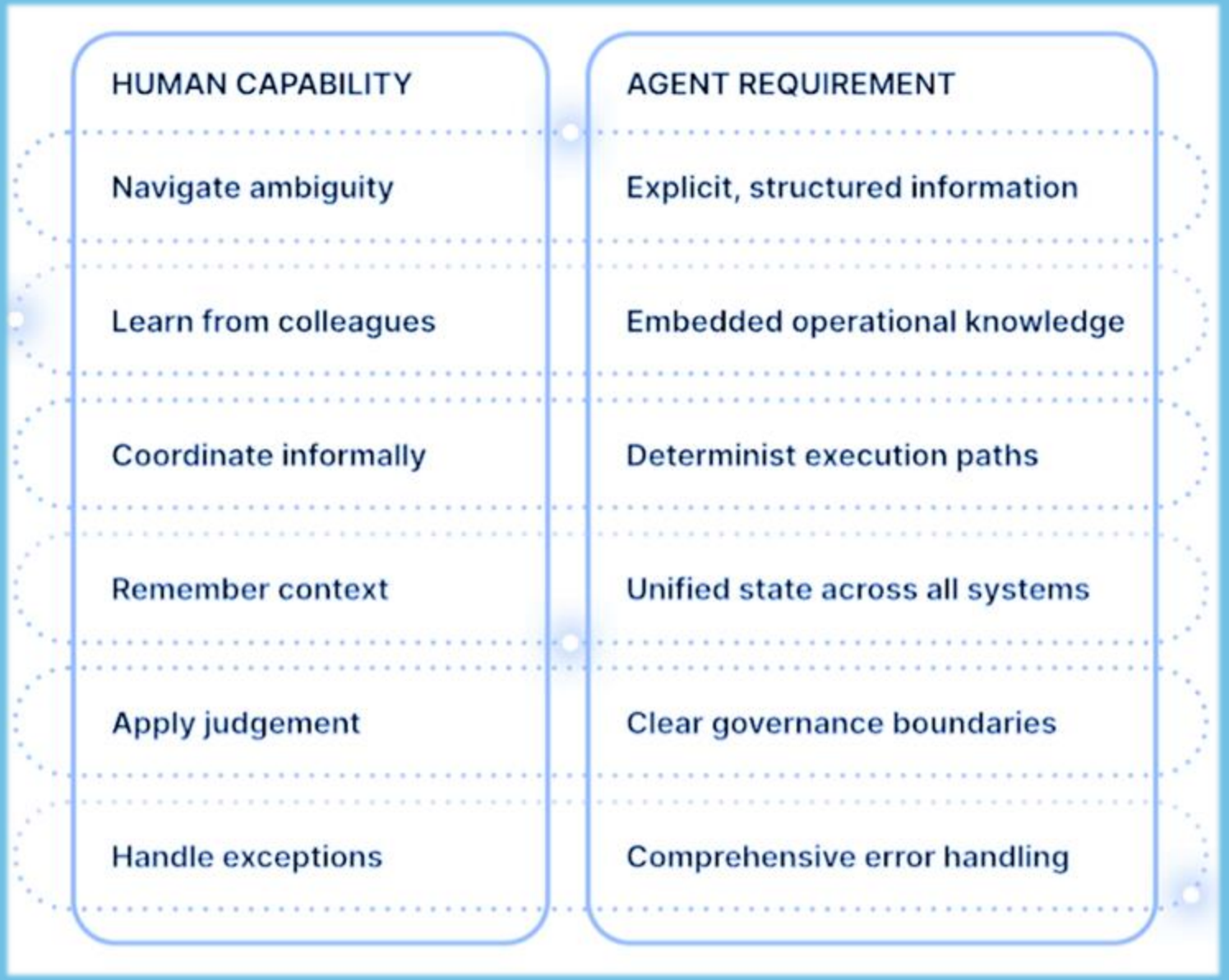
Cost optimization Agent

Continuously analyzes resource usage and recommends or implements optimizations to reduce cloud spend.

Agent Requirements Are Fundamentally Different From Humans



Human capabilities vs agent requirements



HUMAN CAPABILITY	AGENT REQUIREMENT
Navigate ambiguity	Explicit, structured information
Learn from colleagues	Embedded operational knowledge
Coordinate informally	Determinist execution paths
Remember context	Unified state across all systems
Apply judgement	Clear governance boundaries
Handle exceptions	Comprehensive error handling

When Terraform partially fails:

Human: Checks Slack, finds someone mentioned this yesterday, applies workaround

Agent: Sees error state, cannot determine if retry is safe, stops or corrupts state further

When configuration is incomplete:

Human: Recognizes what's missing from experience, knows who to ask

Agent: Only knows what's in template, and cannot fill gaps

When compliance is special:

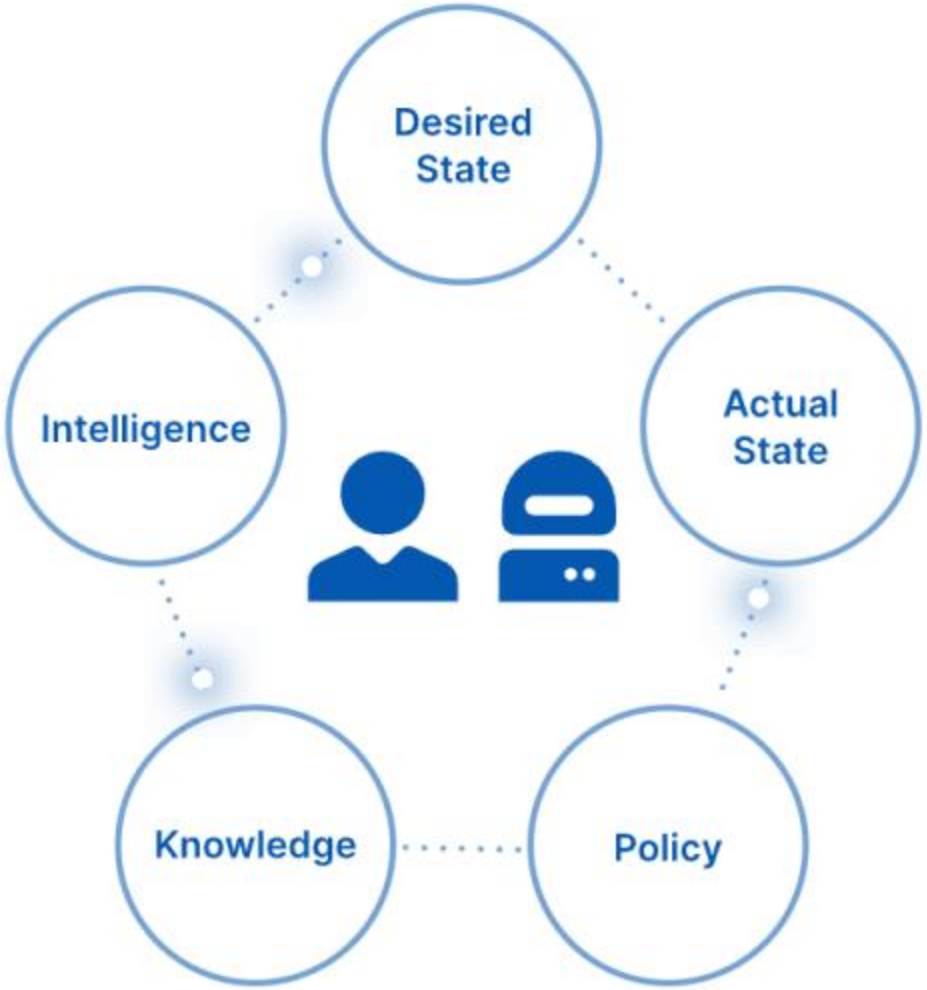
Human: Remembers from training that payment services need extra controls

Agent: Has no knowledge outside automated path

Why Platforms Break for Agents: Platforms are Built for Humans, Not AI

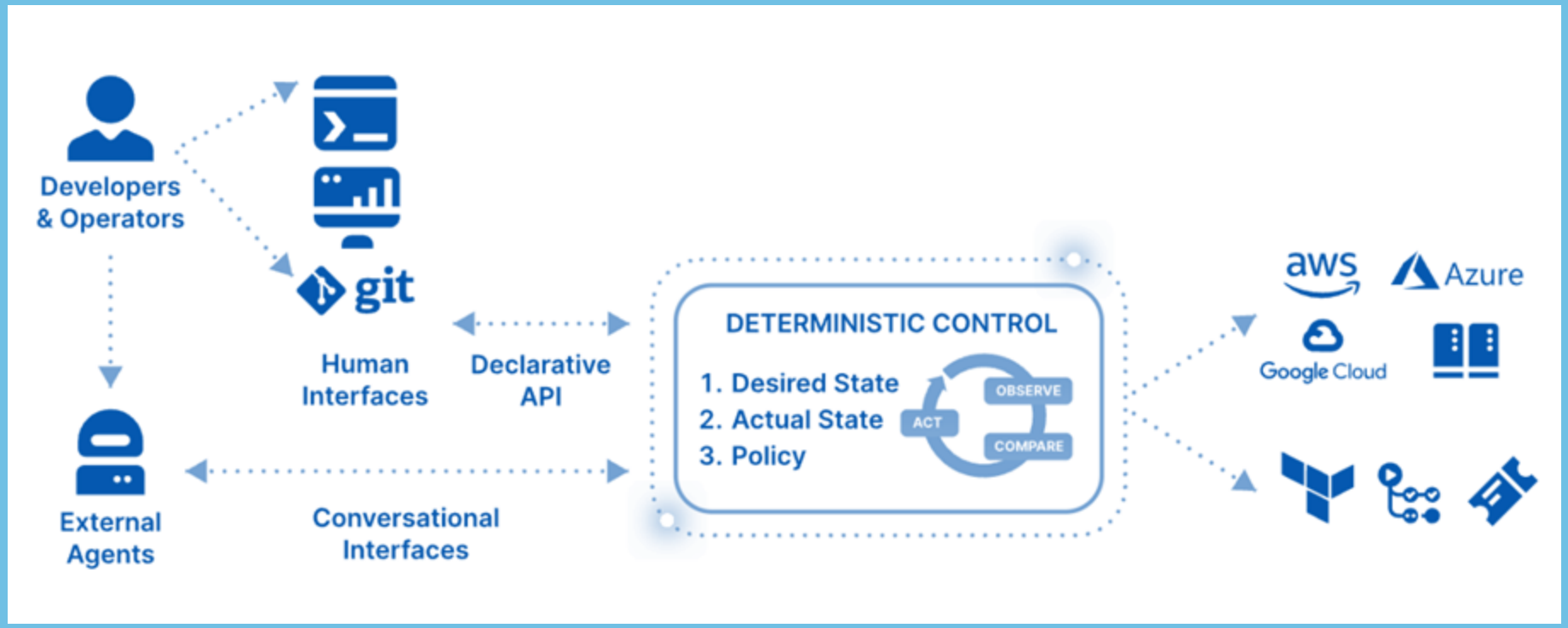
- **Fragmentation of Operational Elements:** Desired state in Git, actual state in cloud consoles, policies in pipelines, and tribal knowledge in people's heads. Humans bridge this gap; agents cannot.
- **Humans as Error Handlers:** Automation relies on humans to fix brittle paths (rerun scripts, tweak configs). Agents lack judgment and informal workarounds, they either blindly retry or fail.
- **Day-2 Operations & Context:** Scaling, incident response, maintenance depend on institutional memory and ad-hoc coordination. Agents lack context and informal networks, risking chaos in production.

What is Needed for Agentic ops?



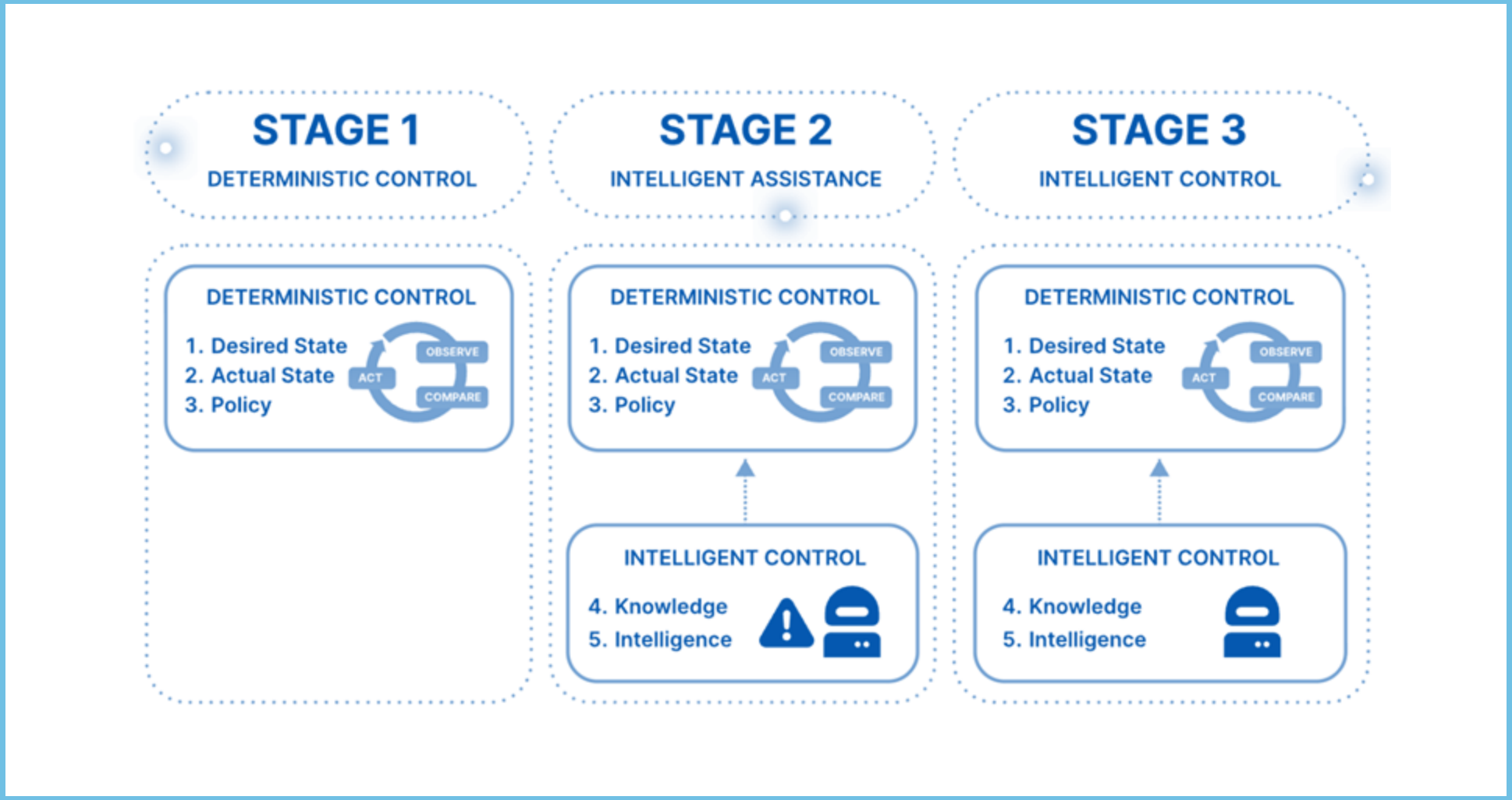
Tools To Use Today and in the Future

Infrastructure Orchestration with Crossplane 2.0



- Declarative Desired State
- Composition & Composition Functions
- Continuous Reconciliation & Drift Correction

Control Plane as Authoritative System of Record



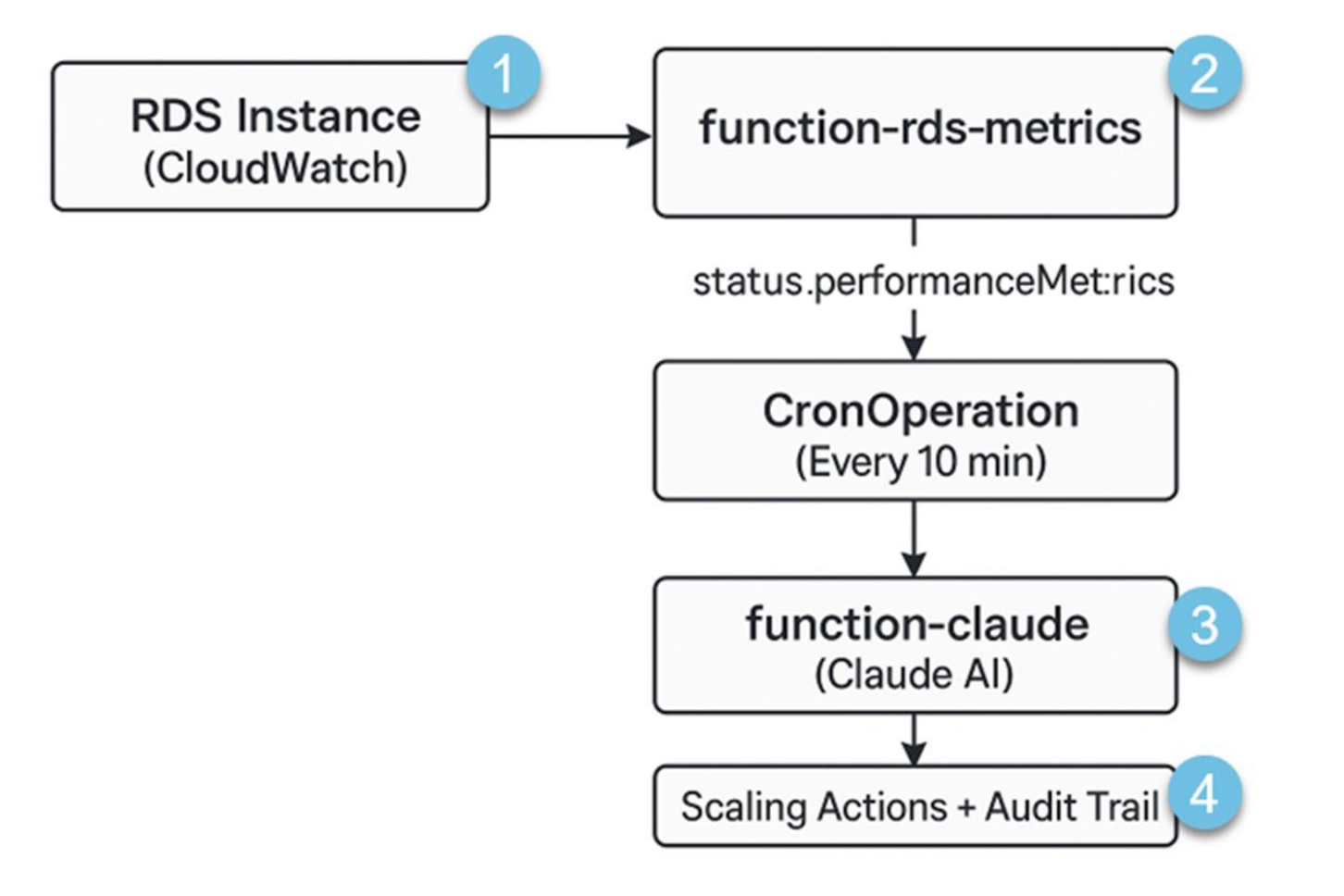
Prometheus: Monitoring Matters

- Agents depend on continuous, reliable metrics
- Metrics → context → decisions
- Prometheus provides the perceptual layer for agent loops
- Observability becomes part of the agent’s world model

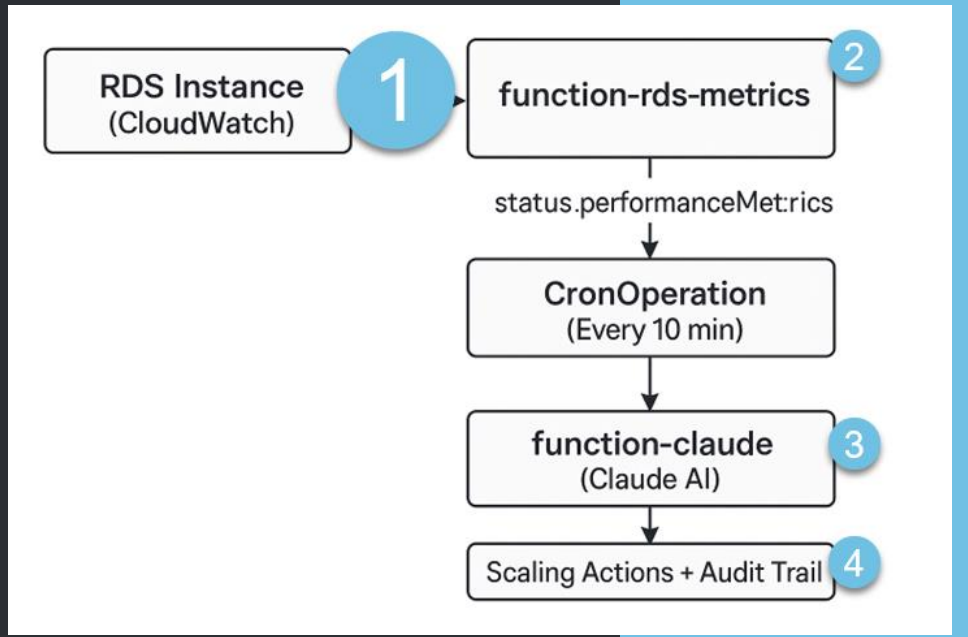
Demo

Demo: An Autonomous Infrastructure Agent on Kubernetes

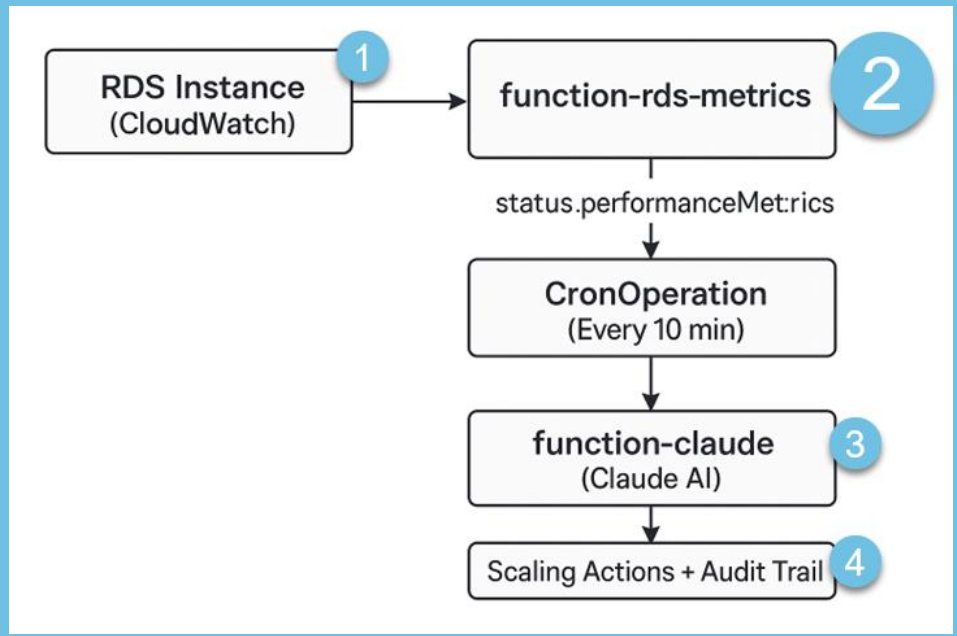
- A real AI agent:
  - Reads metrics
  - Analyzes them with Claude
  - Makes scaling decisions
  - Updates desired state
  - Crossplane enforces it



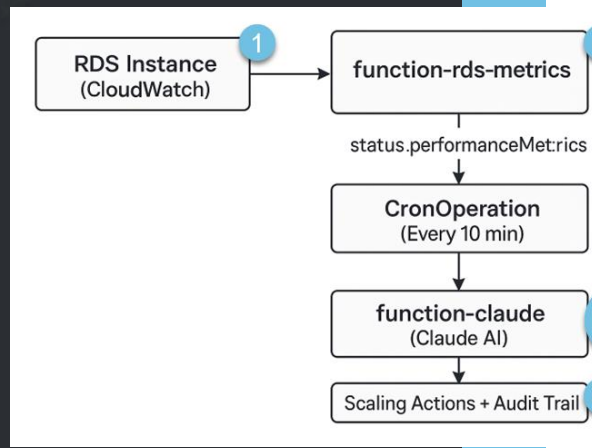
```yaml
apiVersion: aws.platform.upbound.io/v1alpha1
kind: XSQLInstance
metadata:
  name: rds-metrics-database-ai-scale
  namespace: default
  labels:
    scale: me
spec:
  compositionSelector:
    matchLabels:
      type: rds-metrics
  parameters:
    region: us-west-2
    engine: mariadb
    engineVersion: "10.11"
    storageGB: 5
    instanceClass: db.t3.micro
    autoGeneratePassword: true
    publiclyAccessible: true  # Temporarily for stress testing
    passwordSecretRef:
      namespace: default
      name: mariadbsecret
      key: password
    networkRef:
      id: rds-metrics-database-ai-scale
```



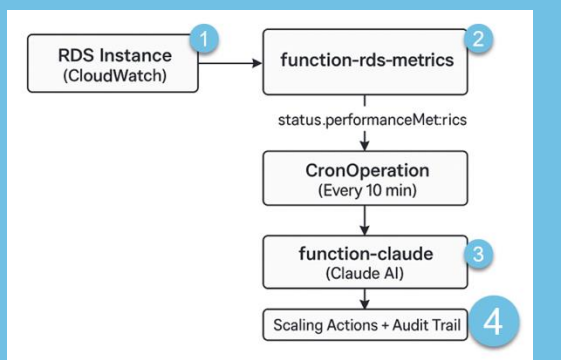
```yaml
performanceMetrics:
  databaseName: rds-metrics-database-ai-scale
  metrics:
    CPUUtilization:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Percent
      value: 2.6191111704257306
    DatabaseConnections:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Count
      value: 0
    FreeStorageSpace:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Bytes
      value: 2775519232
    FreeableMemory:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Bytes
      value: 188219392
    ReadIOPS:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Count/Second
      value: 0.01667722891164404
    WriteIOPS:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Count/Second
      value: 0.6670891564657616
  region: us-west-2
  timestamp: "2025-09-15T12:53:49.034106466Z"
```

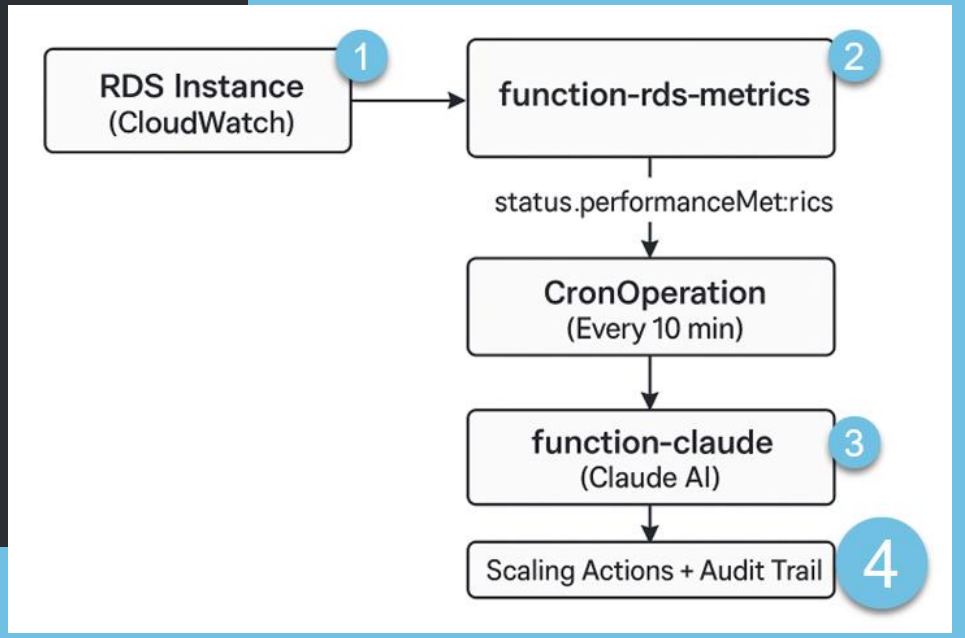
```
input:
  apiVersion: claude.fn.upbound.io/v1alpha1
  kind: Prompt
  systemPrompt: |
    You are an intelligent RDS scaling system that analyzes XSQLInstance resources
    and makes scaling decisions based on CloudWatch performance metrics.
  userPrompt: |
    You are analyzing an XSQLInstance resource for potential RDS scaling needs.

    SCALING ANALYSIS CRITERIA:
    1. Check performance metrics in status.performanceMetrics
    2. Check current instanceClass in spec.parameters.instanceClass
    3. Check if analysis was done recently (intelligent-scaling annotations)

    RATE LIMITING:
    - Skip analysis if "intelligent-scaling/last-analyzed" annotation exists and is < 5 minutes old
    - Only proceed if no recent analysis or if metrics show significant changes

    SCALING TRIGGERS (conservative for cost control):
    - SCALE UP: CPU > 85%, Memory < 15%, Connections > 85%
    - SCALE DOWN: CPU < 20% AND Memory > 60% AND Connections < 30% for sustained period

    INSTANCE CLASS PROGRESSION (bidirectional):
```



```
→  configuration-aws-database-ai git:(demo-sofia) ✗ k get xsqlinstances.aws.platform.upbound.io rds-metrics-database-ai
-scale -o yaml | yq
apiVersion: aws.platform.upbound.io/v1alpha1
kind: XSQLInstance
metadata:
  annotations:
    intelligent-scaling/last-analyzed: "2025-09-18T12:19:00Z"
    intelligent-scaling/last-scaled: "2025-09-17T23:59:59Z"
    intelligent-scaling/last-scaled-decision: No scaling needed. Very low resource utilization (CPU 2.09%, 0 connection
s) and already at smallest instance class (db.t3.micro).
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"aws.platform.upbound.io/v1alpha1","kind":"XSQLInstance","metadata":{"annotations":{},"labels":{"sc
ale":"me"},"name":"rds-metrics-database-ai-scale"},"spec":{"compositionSelector":{"matchLabels":{"type":"rds-metrics"}}
,"parameters":{"autoGeneratePassword":true,"engine":"mariadb","engineVersion":"10.11","instanceClass":"db.t3.micro","ne
tworkRef":{"id":"rds-metrics-database-ai-scale"},"passwordSecretRef":{"key":"password","name":"mariadbsecret","namespac
e":"default"},"publiclyAccessible":true,"region":"us-west-2","storageGB":5},"writeConnectionSecretToRef":{"name":"rds-m
etrics-database-ai-scale","namespace":"default"}}}
```

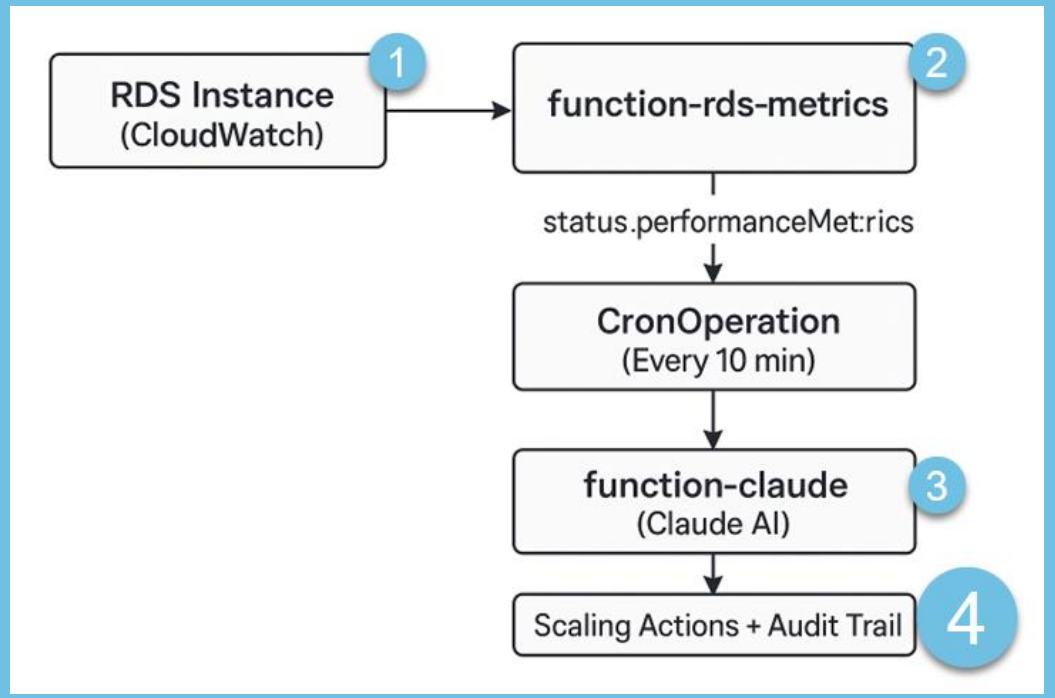


```
→  configuration-aws-database-ai git:(demo-sofia) ✗ ./perf-scale-demo.sh
🚀 Starting DEMO load test (optimized for speed)...
⏱️  Load test running... Expected timeline:
   - 30-60 seconds: CPU should hit 50%+
   - 1-2 minutes: CloudWatch metrics update
   - 2-3 minutes: Claude analysis and scaling decision
   - 5-10 minutes: Instance scaling completion

═══ Demo Check 1 (14:19:47) ═══
🔥 CPU: 1.89987334177721 47% (threshold: 50%)
error: flag needs an argument: 'o' in -o
See 'kubectl get --help' for usage.
💾 Instance:
🤖 Claude: ...
🎉 SCALING SUCCESSFUL! Instance upgraded to

🔴 Stopping load test...
✅ Demo complete!
```



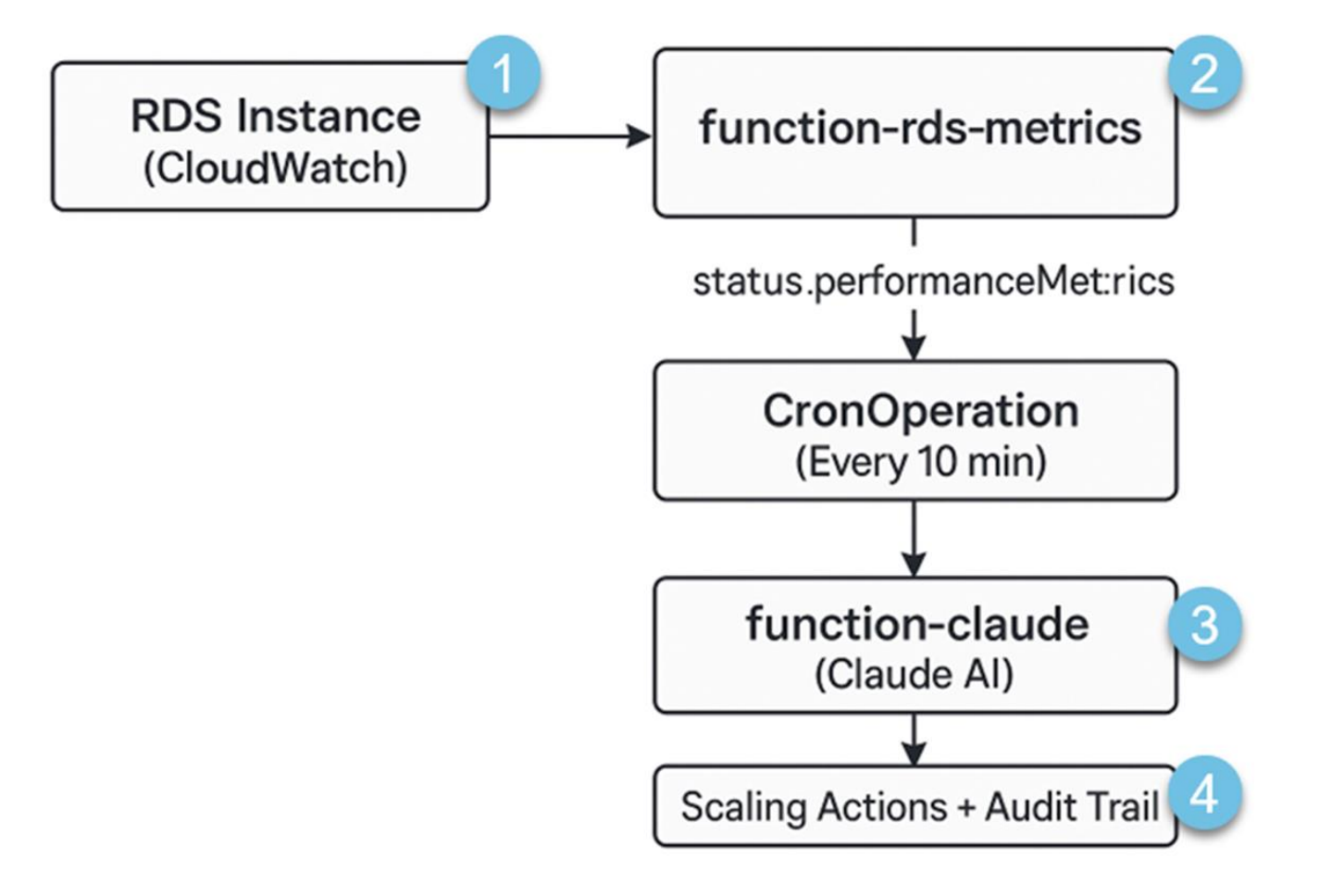
```
→  configuration-aws-database-ai git:(main) ✗ k get xsqlinstance.aws.platform.upbound.io/rds-metrics-database-ai-scale -o yaml|yq
apiVersion: aws.platform.upbound.io/v1alpha1
kind: XSQLInstance
metadata:
  annotations:
    intelligent-scaling/last-analyzed: "2025-09-15T19:33:59Z"
    intelligent-scaling/last-scaled: "2025-09-15T19:37:58Z"
    intelligent-scaling/last-scaled-decision: Scaling up due to high CPU utilization (100%) and low freeable memory
```

```
parameters:
  autoGeneratePassword: true
  deletionPolicy: Delete
  engine: mariadb
  engineVersion: "10.11"
  instanceClass: db.t3.medium
  networkRef:
    id: rds-metrics-database-ai-scale
  passwordSecretRef:
    key: password
    name: mariadbsecret
    namespace: default
  providerConfigName: default
  publiclyAccessible: true
  region: us-west-2
  storageGB: 5
```

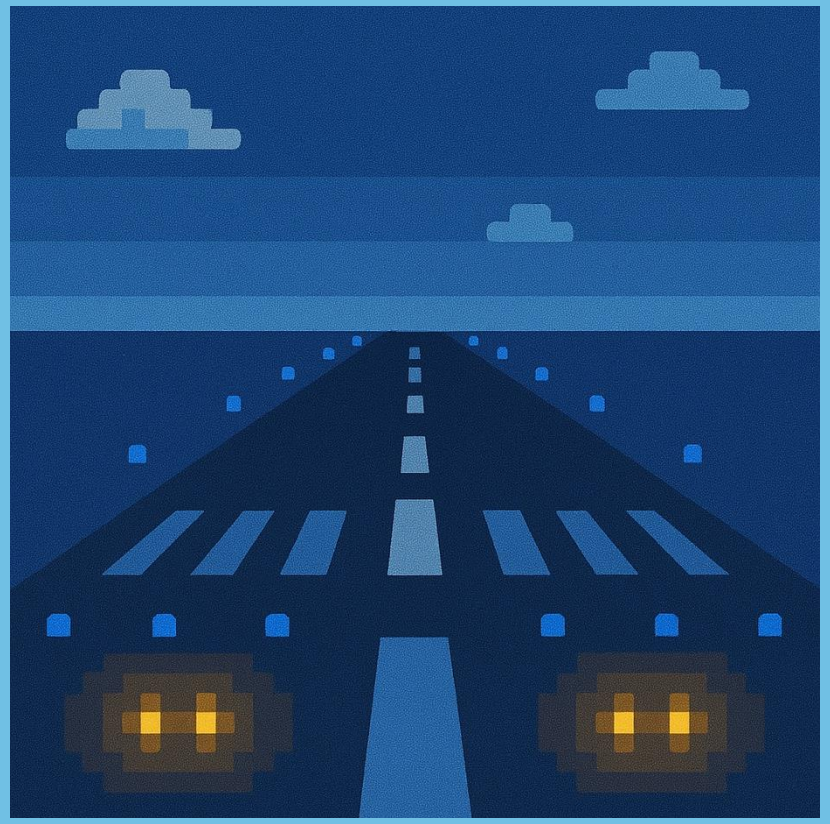


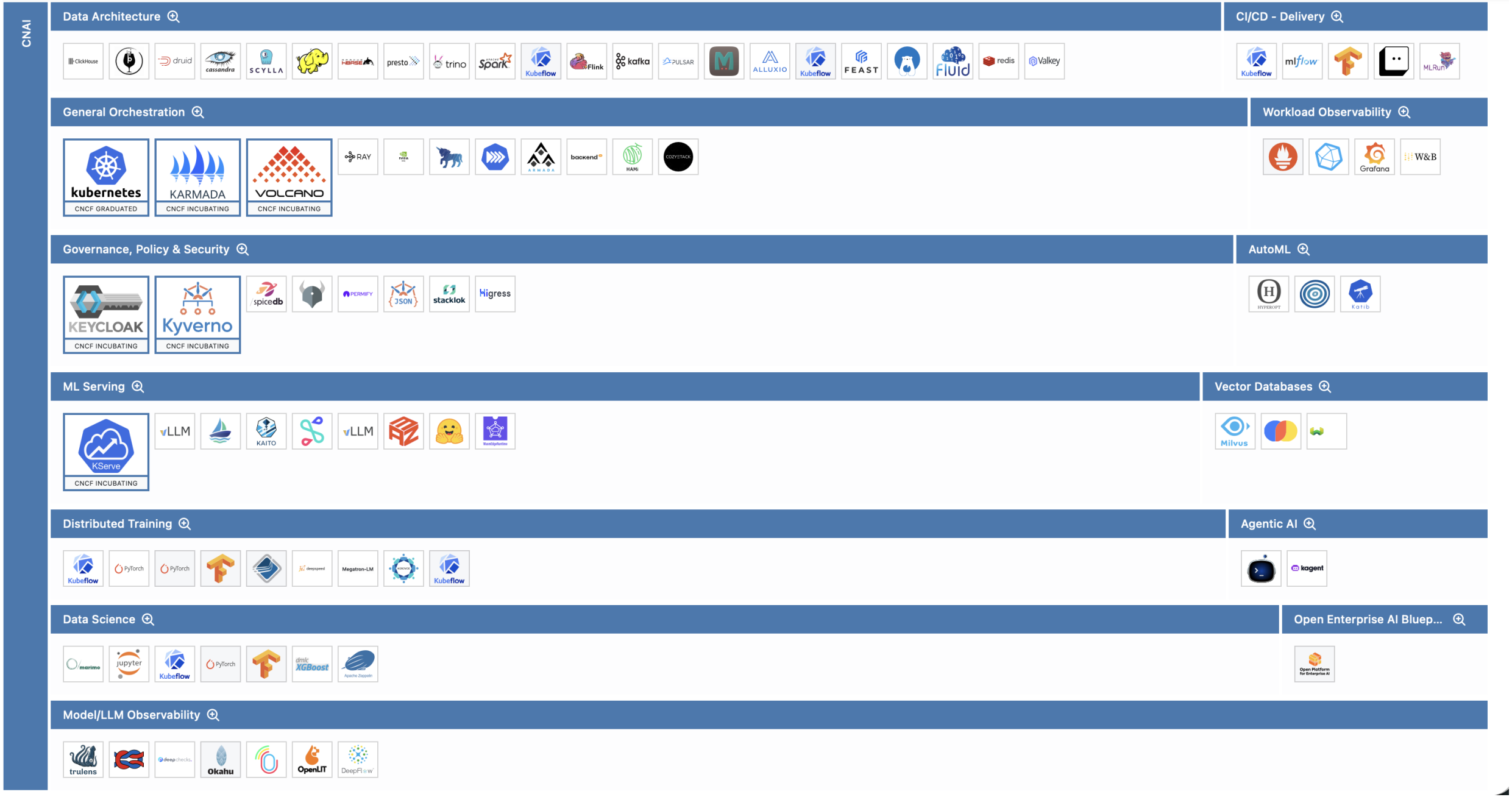
Demo: An Autonomous Infrastructure Agent on Kubernetes

- A real AI agent:
  - Reads metrics
  - Analyzes them with Claude
  - Makes scaling decisions
  - Updates desired state
  - Crossplane enforces it



Landscape



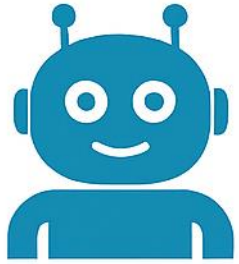

CNAI
Data Architecture
ClickHouse
druid
cassandra
SCYLLA
presto
trino
Spark
Kubeflow
Flink
kafka
PULSAR
ALLUXIO
Kubeflow
FEAST
FLUID
redis
Valkey
CI/CD - Delivery
Kubeflow
mlflow
MLRun
General Orchestration
kubernetes
CNCF GRADUATED
KARMADA
CNCF INCUBATING
VOLCANO
CNCF INCUBATING
RAY
ARMADA
backend
HAMi
COZYETACK
Workload Observability
Grafana
W&B
Governance, Policy & Security
KEYCLOAK
CNCF INCUBATING
Kyverno
CNCF INCUBATING
spicedb
PERMIFY
JSON
stacklok
Higress
AutoML
HYPEROPT
Katib
ML Serving
KServe
CNCF INCUBATING
vLLM
KAITO
vLLM
Vector Databases
Milvus
Distributed Training
Kubeflow
PyTorch
PyTorch
deepspeed
Megatron-LM
Kubeflow
Agentic AI
kagent
Data Science
marimo
jupyter
Kubeflow
PyTorch
XGBoost
Open Enterprise AI Bluep...
Model/LLM Observability
trulens
deepchecks
Okahu
OpenLIT
DeepFlow

Ray: Distributed Compute for Agent Workloads

- Ray is a distributed execution engine for Python-based agents
- Scales agent tasks, simulations, and tool-calling workloads
- Ideal for multi-step, long-running, or parallel reasoning flows
- Complements, not replaces, control-plane-native agents
- Works alongside Kubernetes and Crossplane in production stacks

Where Open-Source Tool Fits in the Agent Stack

Perception → Prometheus, CloudWatch
Reasoning → Claude, LangChain
Coordination → Autogen, CrewAI
Distributed Execution → Ray
Action Layer → Crossplane
Safety / Guardrails → Kyverno, Falco



SRE Agent
Monitors system health and remediates incidents

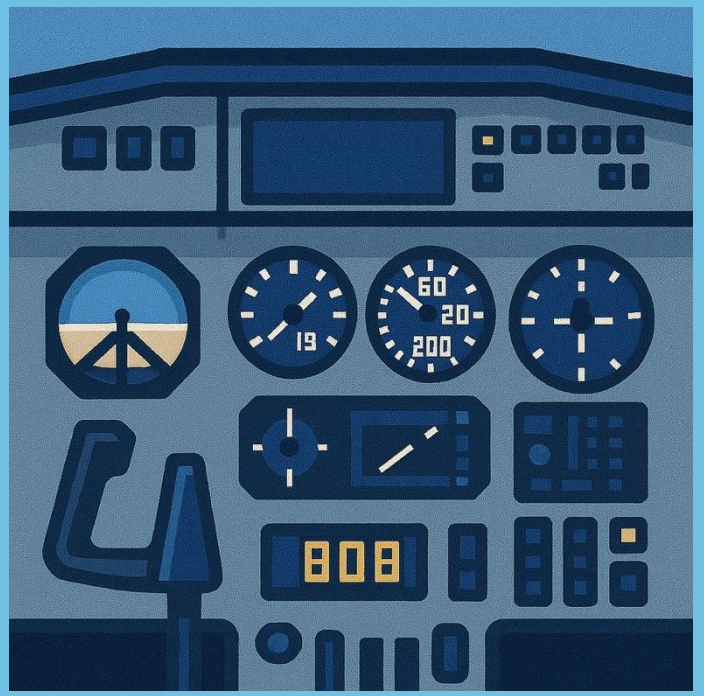
Security Compliance Agent
Enforces security policies and compliance requirements

Crossplane

DevOps Provisioning Agent
Automates infrastructure provisioning and configuration

Cost Optimization Agent
Analyzes resource policies and compliance requirements

Tips To Keep in Mind



“It worked on my machine”, but now with AI

- **No Unified Observability:** Demos show console logs; production spans cloud services. We need metrics, tracing, and logging (Prometheus, OpenTelemetry) plus drift reconciliation (Crossplane).
- **No Human in the Loop:** Agents bypass reviews and playbooks. Guardrails and policies must enforce encryption, resource limits, and compliance automatically.
- **Speed & Scale of Incidents:** Agents act fast, so errors and compromises can spread in seconds.

Policy and Guardrails for Autonomous Infrastructure

- **Preventive Guardrails (Admission Control & Policy Enforcement):** Kyverno policies act at deploy time: require resource limits, enforce trusted images, mandatory labels, disallow privileged containers. Policies-as-code in YAML provide automated compliance checks before changes go live.
- **Detective/Reactive Guardrails (Monitoring & Auto-Remediation):** Falco monitors runtime behavior: detects abnormal syscalls, privilege escalation, or suspicious network activity. Automated response via Falco Sidekick and Istio: isolate workloads, restart pods, and alert humans. Crossplane reconciliation adds drift correction for config tampering.

Architectural Patterns for AI-Native Security

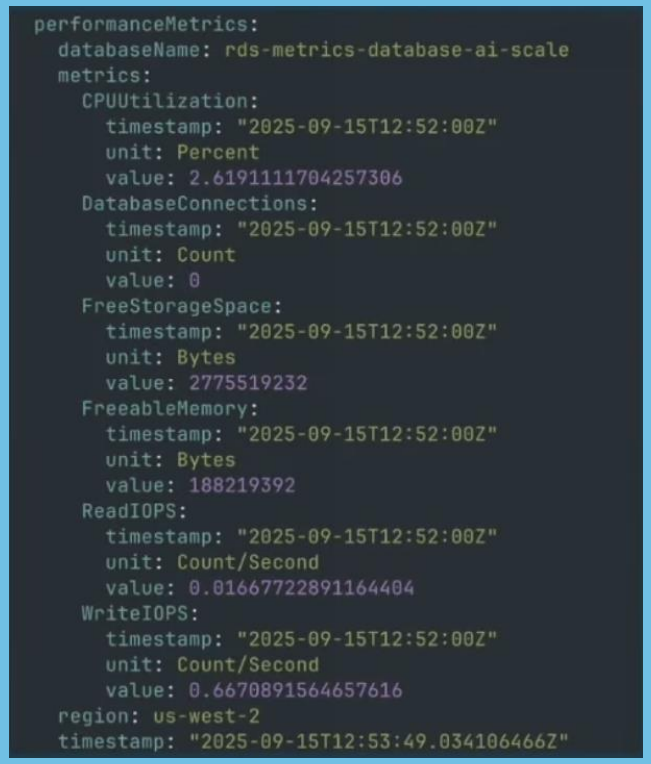
- Shift security left
- GitOps for the win
- Minimize agentic access
- Defense in depth

Summary

- AI agents need structure, not scripts
- The control plane is the interface for AI
- Metrics (Prometheus) provide perception
- LLMs provide reasoning; Crossplane provides safe action
- Autonomous loops are possible today with open-source tools
- Policies & guardrails make autonomy safe

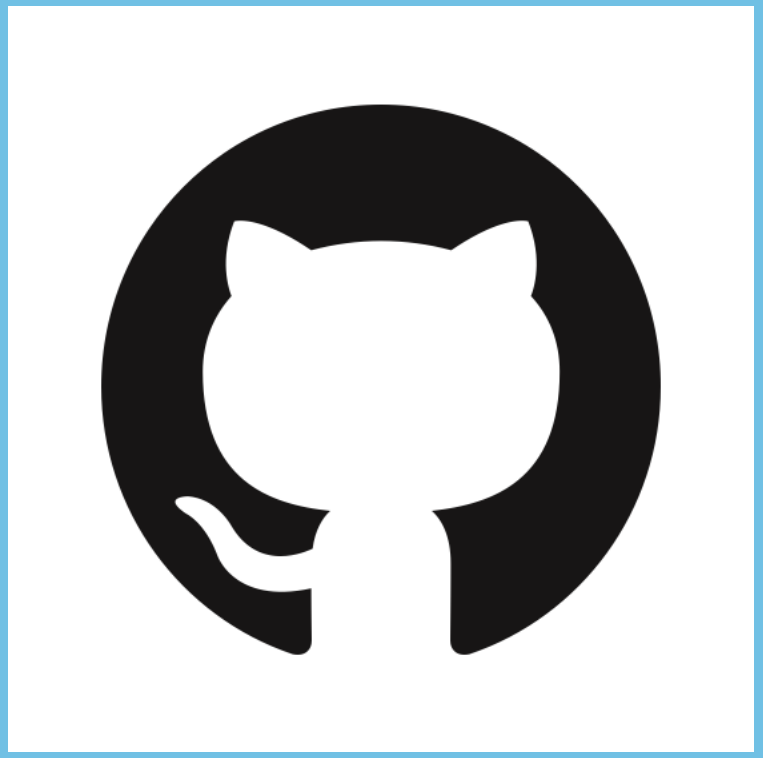
Resources

upbound.io/intelligent-control-plane



```
performanceMetrics:
  databaseName: rds-metrics-database-ai-scale
  metrics:
    CPUUtilization:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Percent
      value: 2.6191111704257306
    DatabaseConnections:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Count
      value: 0
    FreeStorageSpace:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Bytes
      value: 2775519232
    FreeableMemory:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Bytes
      value: 188219392
    ReadIOPS:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Count/Second
      value: 0.01667722891164404
    WriteIOPS:
      timestamp: "2025-09-15T12:52:00Z"
      unit: Count/Second
      value: 0.6670891564657616
  region: us-west-2
  timestamp: "2025-09-15T12:53:49.034106466Z"
```

github.com/upbound/configuration-aws-database-ai



github.com/AnnieTalvasto/presentations

Thank you!


HEY COME OUTSIDE
www.facebook.com/AviationMemess
IT'S A BEAUTIFUL DAY
I CAN'T
I'M STILL GROUNDED