# Earth 2.0 Observability: Prometheus Labeling Strategy

## Overview

Earth 2.0 uses Prometheus for observability across its decentralized AI agents, simulation nodes, and smart contract execution layers. To ensure precision, traceability, and low-latency diagnostics, all metrics are tagged with a predefined, low-cardinality label schema.

## Core Prometheus Labels

| Label Key | Purpose | Example |
| --- | --- | --- |
| agent\_role | Identifies the type of AI agent | predictive, governance, infra |
| environment | Deployment context | testnet, staging, prod |
| region | Geographic cluster ID | appalachia, permian |
| proposal\_id | DAO proposal execution context | cfp-042 |
| contract\_address | Source smart contract (NFT, Vault, Token) | 0xabc123... |
| agent\_version | Version of the deployed agent | v1.3.7 |
| source\_repo | Git origin for audit/debug | github.com/cuttlefishlabs/... |
| node\_id | Node identifier for distributed agents | e2-al-node-01 |

## Example Metrics

### Agent Decision Rate

rate(agent\_decision\_events\_total{  
 agent\_role="predictive",  
 environment="prod",  
 region="appalachia"  
}[5m])

### Proposal Evaluation Success Rate

sum by (proposal\_id) (  
 rate(proposal\_eval\_success\_total{environment="staging"}[10m])  
)

### CPU Usage by Agent Role

avg by (agent\_role) (  
 rate(cpu\_usage\_seconds\_total{environment="prod"}[5m])  
)

## Multi-Dimensional Monitoring

Memory access by node and role:

rag\_memory\_access\_total{  
 agent\_role="predictive",  
 node\_id="e2-bhm-node-03"  
}

Error rate by agent version:

agent\_fallback\_trigger\_total{  
 agent\_role="scorer",  
 proposal\_id="cfp-007",  
 agent\_version="v2.0.0"  
}

## Alerting Example

Trigger alert if more than 1% of proposal evaluations fail:

sum(rate(proposal\_eval\_failure\_total{environment="prod"}[5m]))  
/  
sum(rate(proposal\_eval\_attempt\_total{environment="prod"}[5m])) > 0.01

## Labeling Best Practices

* Use consistent, descriptive label keys.
* Avoid high-cardinality dynamic values (e.g., UUIDs, IPs).
* Limit values to finite, bounded domains.
* Define schema in advance (see observability/prom-schema.yaml).
* Audit labels via /metrics endpoint and Grafana dashboards.

## Next Steps

* Integrate with Grafana dashboards under CI\_CD/grafana/alerts.json.
* Terraform+Kubernetes config bundle available for automated setup.
* Use kubectl-ai to manage AI-assisted observability in GKE.

Reference: [kubectl-ai GitHub](https://github.com/GoogleCloudPlatform/kubectl-ai)