CML Feature Requests for Cloud Manager Integration

Date: November 26, 2025
Author: Cisco Certifications Systems Team, CML Cloud Manager Team
Target Audience: Cisco Modeling Labs (CML) Engineering Team

Executive Summary

The CML Cloud Manager is a strategic initiative by the **Cisco Certifications Systems Team** to deliver "CML-based Lablets" to a diverse audience, including internal workforce (Exam PMs, Subject Matter Experts, Systems Operators) and external users (software vendors, SMEs, and exam candidates).

Phase 1 (Current State):
We currently manage CML compute resources for two primary use cases:

1. **Content Development**: Providing on-demand CML Workers for Exam PMs and SMEs to design, build, and test Lablet content.
2. **Exam Delivery**: Monitoring real-time provisioning of CML Workers for candidates attempting Cisco Exams via PVUE. While Cloud Manager currently observes this activity rather than orchestrating it, the lack of deep integration leads to inefficient compute usage and higher costs.

Phase 2 (Future State):
The roadmap involves evolving CML Cloud Manager into a comprehensive **Resource Manager** that decouples compute provisioning (CML Workers) from workload scheduling (Lablet instances). This architecture will mirror Kubernetes principles, autonomously auto-scaling compute resources to handle distributed workloads efficiently.

To support this evolution, optimize cloud costs, and ensure enterprise security, we require deeper integration capabilities with the CML platform. The following feature requests outline critical enhancements to CML's API and security architecture that would enable:

1. **Significant Cost Reductions**: Through accurate, activity-based idle shutdown.
2. **Enterprise Security Compliance**: Via modern OIDC/OAuth2 authentication.
3. **Real-time Observability**: Through event-driven architecture (Webhooks) instead of polling.

1. Enhanced Idle Activity Detection API

Problem Statement

Running CML instances on AWS (e.g., `m5zn.metal`) is expensive. To manage costs, we must shut down instances when they are not in use. Currently, "idleness" can only be inferred from:

- **Resource Metrics**: CPU/Memory usage (via AWS CloudWatch or CML `/system_stats`).
- **Network Traffic**: Packet rates.
- **Telemetry Events**: Polling `/telemetry/events` to infer activity.

Limitations:

- **Inefficient Telemetry Polling**: The `GET /telemetry/events` endpoint returns *all* events and does not support query parameters for filtering (e.g., by time range) or pagination. The response size grows significantly over time, making frequent polling for "recent activity" computationally expensive and bandwidth-intensive.
- **False Negatives**: A user might be actively viewing a topology or editing a configuration in the UI without generating significant CPU/Network load.
- **False Positives**: A background simulation might spike CPU usage even if no user is present (though this is usually a valid "active" state, the converse is the main issue).
- **Console Activity**: A user typing in a console session generates negligible resource load but is highly "active".

Feature Request

Expose a **"Last User Activity"** metric or API endpoint that aggregates:

1. **HTTP/UI Interactions**: Last time a user made a request to the CML API or interacted with the Dashboard.
2. **Console/VNC Sessions**: Last keystroke or active connection timestamp on any node console.
3. **Smart Lab Activity**: Last time a node state changed or a link was modified by a user.

Proposed API Design

```
GET /api/v0/system/activity
```

```
{
  "last_activity_timestamp": "2025-11-26T14:30:00Z",
  "active_sessions": {
    "web_users": 2,
    "console_connections": 1,
    "vnc_connections": 0
  },
  "details": [
    {
      "user": "jdoe",
      "last_active": "2025-11-26T14:30:00Z",
      "activity_type": "console_input",
      "lab_id": "lab-123",
      "node_id": "node-abc"
    }
  ]
}
```


Benefits

- **Accurate Cost Management**: We can confidently shut down workers that have truly been abandoned by users, saving thousands in cloud spend.
- **User Experience**: Prevents accidental shutdowns while a user is reading a lab guide or thinking about a configuration change.

2. Native OAuth2/OIDC Support

Problem Statement

Our Cloud Manager uses **Keycloak** (OIDC) for centralized identity management. CML currently supports:

- **Local Users**: Requires manual provisioning or API-based synchronization.
- **LDAP**: Legacy protocol, requires direct connectivity to an LDAP server which is complex in cloud/hybrid environments.

Currently, we must maintain a "Dual Auth" system where the Cloud Manager handles OIDC and then "impersonates" or manages CML local users, leading to:

- **Synchronization Issues**: User exists in Keycloak but not CML.
- **Role Drift**: Permissions in Keycloak don't match CML permissions.
- **Security Risks**: Managing CML local credentials programmatically.

Feature Request

Implement native **OpenID Connect (OIDC)** support as an authentication provider in CML, similar to the existing LDAP integration.

Requirements

1. **OIDC Configuration**: Allow configuring Issuer URL, Client ID, Client Secret, and Scopes via API/UI.
2. **Role Mapping**: Map OIDC Claims (e.g., `groups`, `roles`) to CML Roles (`admin`, `user`, `read_only`).
   - Example: Claim `cml-admins` -> CML `admin`.
3. **JIT Provisioning**: Automatically create/update users in CML upon successful OIDC login.
4. **Token Exchange/Trust**: Allow CML to validate JWTs issued by the trusted IdP directly for API access.

Benefits

- **Unified Identity**: Single source of truth for users and groups.
- **Simplified Architecture**: Removes the need for complex user-sync logic in the Cloud Manager.
- **Zero Trust**: Enables end-to-end token-based authentication flows.

3. System-wide Webhooks (Event Push)

Problem Statement

To maintain an accurate state of the fleet (e.g., "Which labs are running?", "Who is logged in?"), the Cloud Manager currently **polls** multiple CML endpoints (`/labs`, `/system_stats`, etc.) every few minutes.

- **Inefficient**: Wastes compute/network on both sides.
- **Latency**: State changes (e.g., Lab Started) are detected with a delay, leading to stale UI data.
- **Scalability**: Polling N workers scales linearly with fleet size, creating a "thundering herd" effect.

Feature Request

Implement a **Webhook** system where CML pushes JSON payloads to a configured URL upon specific system events.

Key Events Needed

1. **System Events**: `UserLoggedIn`, `UserLoggedOut`, `SystemReady`, `SystemShutdown`.
2. **Lab Events**: `LabCreated`, `LabStarted`, `LabStopped`, `LabWiped`, `LabDeleted`.
3. **Node Events**: `NodeStarted`, `NodeStopped`, `ConsoleConnected`, `ConsoleDisconnected`.

Proposed Configuration

`POST /api/v0/system/webhooks`

```
{
  "url": "https://cloud-manager.internal/api/webhooks/cml",
  "secret": "hmac-shared-secret",
  "events": ["lab.*", "system.user.*"],
  "verify_ssl": true
}
```

Proposed Payload (CloudEvents Compliant)

```
{
  "specversion": "1.0",
  "type": "com.cisco.cml.lab.started",
  "source": "/cml/worker-123",
  "subject": "worker-uuid-1234",
  "id": "evt-unique-id",
  "time": "2025-11-26T14:35:00Z",
  "data": {
    "cml_cluster_id": "worker-uuid-1234",
    "lab_id": "lab-abc",
    "user_id": "jdoe",
    "lab_title": "OSPF Migration"
  }
}
```

Benefits

- **Real-time Observability**: Cloud Manager UI reflects reality instantly.
- **Reactive Automation**: Trigger billing logic immediately when a lab starts; trigger cleanup immediately when a lab is deleted.
- **Reduced Load**: Eliminates aggressive polling, freeing up CML API resources for actual user tasks.