

# DS01 Infrastructure Update

---

- 29/11/2025
  - local repo path </opt/ds01-infra>
  - remote repo url <https://github.com/hertie-data-science-lab/ds01-infra>
- 

## Exec Summary

New infra layer(s) wrap original AIME ML Containers with:

- **Resource management:** per-user GPU/CPU/mem/PID limits (auto enforced via mix of cron/systemd/cgroups); GPU allocation system w/ MIG-partitions; dynamically-configurable YAML for resource limits, single-source of truth for logging & monitorings, etc
- **Educational/easy-to-use workflows:** progressive complexity - from beginner CLI wizards that maximally abstract complexity away for new users, to more granular internal commands, to base Docker access => 3 'interfaces' for users of diff skill levels. [--guided](#) mode -> full explanation for new users.
- **Env Customisation:** at image layer; prev setup was for a 'dev -> production' pipeline, now more interactive (students inject additional pkgs into containers pre-deployment; important as students not likely to know all pkg requirements in adv)
- **Cleanup:** idle resource detection, runtime limits, and auto cleanup, etc
- **Security & Login:** SSH keys generation & configuration auto-handled at user setup, user permissions configured so can only view own workspaces etc.
- **Documentation:** currently admin-facing modular README.md files throughout dir (also on DSL Github); user documentation embedded in CLIs -> interactive documentation at point of use.
- **Up-to-date:** updated CUDA & NVML drivers (not for host only for containerised envs), synced internal clock, updated AIME images sub-repo to v2.

**Current Status:** "v1" (MVP) usable for students - robustly tested, but not at scale (yet)

- Students can now go from never accessing server before, to deploying a running container and attaching IDE to it for interactive development in <30 mins (fully guided walkthrough in class took 40 mins)
- 

## Design Philosophy: abstract away initial complexity, while allowing advanced users more control

- Advanced / existing users still have their workflows unbroken, but now are included within logging system (w/ resource limits applied)
- Bare metal access still allowed (so far only Sebastian still needs this -> will be migrating over to containerised-only, with bare-metal access restricted). *NB: By enforcing containerisation -> future proofing as can integrate with cloud registries if needed to push workloads off server if that is future intention (Simon mentioned)*
- dev'd a rich eco-system of CLI tools for students that handle full docker workflow w/o them needing to know more than their existing knowledge (i.e only prerequisite is basic knowledge of local-

machine Python-based pkg dev as taught in DSA class).

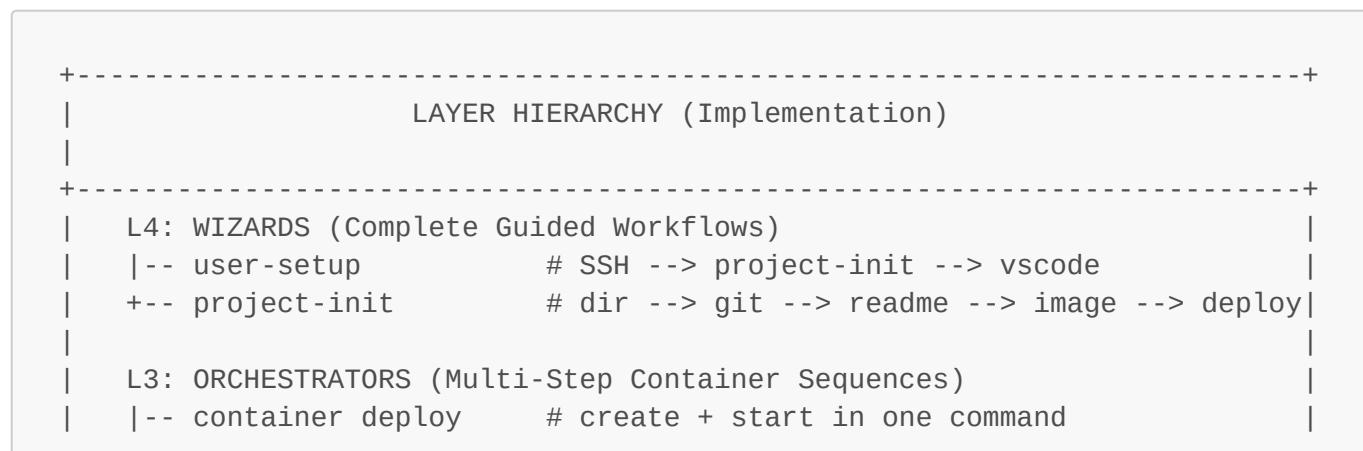
## Layered Abstraction: 3 different user-interfaces

no one-size-fits-all -> **3 user interfaces** at diff abstraction levels:

USER SKILL LEVEL	INTERFACE	WHAT THEY SEE
Beginner (Day 1)	--> ORCHESTRATION full local-remote credential setup (Default) prompts. No decisions needed, one path dirs & git etc	--> `user-setup` walks through One command, follow the -> `project init` sets up
Beginner	--> ORCHESTRATION building & container deployment  removed (avoids zombie allocation prob)  (create, use, retire)	--> customisable image  binary state: running or ephemeral workflow
Intermediate	--> ATOMIC COMMANDS docker workflow, still heavily abstracted (more control) complexity removed	--> greater control/closer to full state model, but much manual lifecycle control
Advanced	--> DOCKER DIRECT  enforcement  "Other"	--> docker run, docker exec still subject to resource  visible in monitoring as

## Implementation: 5-Layer Hierarchy

Under hood, all interfaces built on same foundation:



```

|   +-+ container retire      # stop + remove + free GPU immediately
|
| L2: ATOMIC (Single-Purpose Commands)
|   -- Container: create, start, attach, run, stop, remove, list, stats
|   +-+ Image: create, list, update, delete
|
| L1: MLC (AIME ML Containers) ----- HIDDEN
|   +-+ mlc-create, mlc-open, mlc-stop, mlc-remove, mlc-list (<- patched)
|
| L0: DOCKER (Foundational Container Runtime)
|   +-+ docker run, exec, stop, rm, build, images, ps, stats
+-----+

```

## Key Design Principles

### 1. instant setup for beginners (plug & play)

```

$ user-setup
# One command handles:
# - SSH key configuration
# - Project directory creation (following DS/ML best practices)
# - git setup & README/requirements.txt creation
# - First Docker image build
# - Container deployment
# - VS Code integration setup

```

### 2. --guided mode

every command supports a `--guided` flag that adds educational explanations:

e.g.

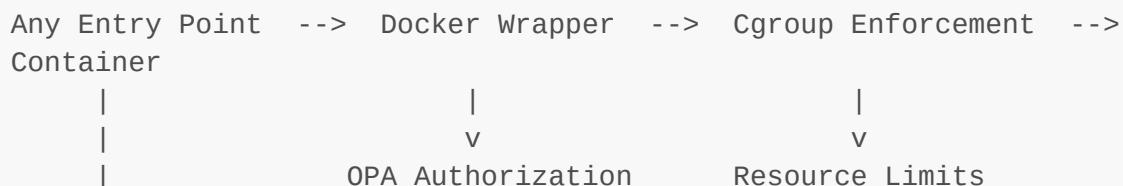
```

$ container-run --guided
# Explains what is a container? What happens when you exit?
# Shows user's own resource limits, allocated GPU, workspace location
# + suggests next steps based on your current state

```

### 3. No Wrong Door

all interfaces eventually hit the same enforcement layer:





**Idea:** even if using ds01 CLI wizards, docker commands, or IDE Containers extension:

- same resource limits
- same GPU allocation tracking
- same idle timeout enforcement
- same audit logging

**Implementation:** 3 layers ensure all containers follow same system:

#### 1. Docker Wrapper (`/usr/local/bin/docker`)

- intercepts all Docker commands
- injects per-user cgroup parent
- adds DS01 labels for tracking

#### 2. Systemd Cgroups (`ds01.slice` hierarchy)

- hierarchical resource accounting
- per-group and per-user slices
- CPU, memory, PIDs limits enforced

#### 3. OPA Authorization Plugin

- policy-based container authorisation

#### 4. Ephemeral by default (Cloud-Native approach)

- teaches the "ephemeral-container persistent-images (& workspace)" philosophy
- prepares students for AWS, Kubernetes etc + future-proofing if DSL wants to move to more of a cloud-based system moving forwards.

```

container deploy my-project  # Create and start
# ... work, train models, experiment ...
container retire my-project # Stop, remove, free GPU immediately
  
```

**5. Easily configurable resource limits YAML file read by crontab -> allows user-/group-specific overrides if more resources needed etc.**

```

# config/resource-limits.yaml
groups:
  student:
    max_mig_instances: 2          # Can use 2 GPU partitions
    max_containers_per_user: 2    # Can run 2 containers
  
```

```

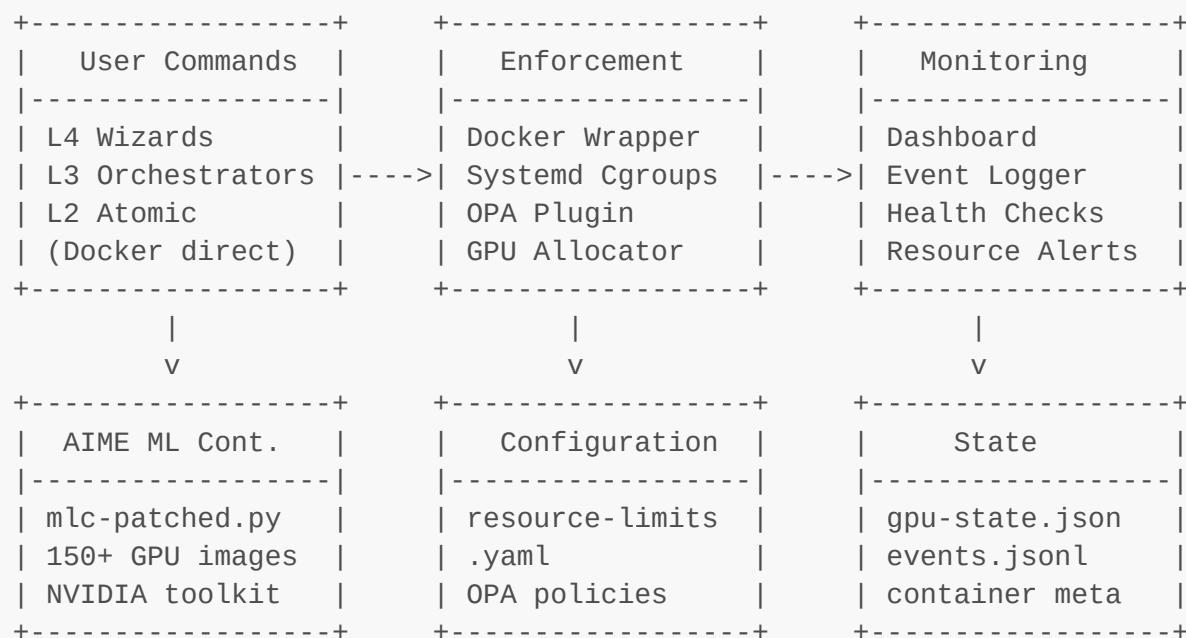
max_cpus: 64          # Per container
memory: 128g          # Per container
idle_timeout: 2h      # Auto-stop after idle
max_runtime: 48h       # Hard limit on runtime

researcher:
  max_mig_instances: 4      # More GPU access
  allow_full_gpu: true      # Can use entire GPU (not just MIG)
  max_runtime: 168h         # Week-long experiments allowed

```

## Architecture Overview

### System Components



## GPU Allocation System

### MIG-Aware Allocation:

- GPUs partitioned into 3 MIG instances (13GB each)
- tracked as `physical_gpu:instance` (mapped to indexes: `0:0, 0:1, 0:2`)
- stateless allocator with file locking (race-safe)
- least-allocated strategy balances load

### Lifecycle:

1. container created --> GPU allocated and labeled
2. container stopped --> GPU marked with timestamp
3. after `gpu_hold_after_stop` --> GPU released to pool

4. container restart --> Validates GPU still available

---

## AIME v2 Integration

= min patching, max reuse

Original AIME Code:	mlc.py (2,100 lines)
DS01 Patch:	mlc-patched.py (+50 lines, 2.5% change)
Code Reuse:	97.5%

Main patch: `--image` flag for custom images, also built on top: allocation system, clean up, CLIs.. but still uses AIME's GPU detection, networking, mount handling + easy to upgrade when AIME releases new versions

## Monitoring & Safety

Tool	Purpose
<code>dashboard</code>	real-time GPU/container visualization
<code>ds01-events</code>	query centralized event log
<code>check-limits</code>	user's personal resource dashboard
<code>ds01-health-check</code>	system integrity validation
<code>resource-alert-checker</code>	warnings at 80% of limits

## Automation (Cron Jobs)

Schedule	Job	Purpose
<code>:30/hour</code>	idle detection	stop containers idle > user's <code>idle_timeout</code>
<code>:45/hour</code>	runtime enforcement	stop containers > user's <code>max_runtime</code>
<code>:15/hour</code>	GPU release	free GPUs from stopped containers
<code>:30/hour</code>	container cleanup	remove old stopped containers

/+ also logging/monitoring crontab jobs

## Technical Metrics

Metric	-
Git commits	140
Automated tests	149 (unit, component, integration, e2e)
Documentation files (admin-facing)	(?) markdown files

Metric	-
Documentation files (user-facing)	(?) markdown files
User-facing commands	30+