DS01 Infrastructure Update

- 29/112025
- 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-partititons; 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

- Advnanced / 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 enfocing 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-interaces

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

```
USER SKILL LEVEL          INTERFACE              WHAT THEY SEE
============================================================================
=========

Beginner (Day 1)     -->  ORCHESTRATION     -->  `user-setup` walks through
full local-remote credential setup
                          (Default)              One command, follow the
prompts. No decisions needed, one path
                                                 -> `project init` sets up
dirs & git etc

Beginner              -->  ORCHESTRATION     --> customisable image
building & container deployment
                                                 binary state: running or
removed (avoids zombie allocation prob)
                                                 ephemeral workflow
(create, use, retire)

Intermediate          -->  ATOMIC COMMANDS   --> greater control/closer to
docker workflow, still heavily abstracted
                           (more control)        full state model, but much
complexity removed
                                                 manual lifecycle control

Advanced              -->  DOCKER DIRECT     -->  docker run, docker exec
                                                 still subject to resource
enforcement
                                                 visible in monitoring as
"Other"
```

Implementation: 5-Layer Hierarchy

Under hood, all interfaces built on same foundation:

```
+-------------------------------------------------------------------------+
|                    LAYER HIERARCHY (Implementation)
|
+-------------------------------------------------------------------------+
|   L4: WIZARDS (Complete Guided Workflows)                               |
|   |-- user-setup           # SSH --> project-init --> vscode            |
|   +-- project-init         # dir --> git --> readme --> image --> deploy|
|                                                                         |
|   L3: ORCHESTRATORS (Multi-Step Container Sequences)                    |
|   |-- container deploy     # create + start in one command              |
|   +-- 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 sser'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:

```
Any Entry Point  -->  Docker Wrapper  -->  Cgroup Enforcement  -->
Container
     |                     |                      |
     |                     v                      v
     |              OPA Authorization      Resource Limits
     |                     |                      |
     +---------------------+----------------------+
                           |
                  = enforced safety
```

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 resourcee limits YAML file read by crontab -> allows user-/group-specific overides if more resoruces 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

```
+------------------+     +------------------+     +------------------+
|   User Commands  |     |   Enforcement    |     |   Monitoring     |
|------------------|     |------------------|     |------------------|
| L4 Wizards       |     | Docker Wrapper   |     | Dashboard        |
| L3 Orchestrators |---->| Systemd Cgroups  |---->| Event Logger     |
| L2 Atomic        |     | OPA Plugin       |     | Health Checks    |
| (Docker direct)  |     | GPU Allocator    |     | Resource Alerts  |
+------------------+     +------------------+     +------------------+
         |                       |                       |
         v                       v                       v
+------------------+     +------------------+     +------------------+
|  AIME ML Cont.   |     |   Configuration  |     |     State        |
|------------------|     |------------------|     |------------------|
| mlc-patched.py   |     | resource-limits  |     | gpu-state.json   |
| 150+ GPU images  |     | .yaml            |     | events.jsonl     |
| NVIDIA toolkit   |     | OPA policies     |     | container meta   |
+------------------+     +------------------+     +------------------+
```

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

Automation (Cron Jobs)

Schedule	**Job**	**Purpose**
`:30/hour`	idle detection	stop containers idle > user's `idle_timeout`
`:45/hour`	runtime enforcement	stop containers > user's `max_runtime`
`:15/hour`	GPU release	free GPUs from stopped containers
`:30/hour`	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+