Separating Lab Continuity from Compute Lifetime

A lifecycle model for high-resource network labs

Jeleel Muibi · HybridOps.Tech · Version 1.1 · August 2026

Abstract

High-resource network labs may need substantial CPU, memory and virtualisation support while they are in use. Keeping that execution capacity available between sessions provides continuity, but it also ties the lifetime of the lab to the lifetime of the host.

This paper introduces a continuity-aware lifecycle model that separates those two concerns. The model makes the required continuity outcome an explicit control input, identifies the state needed to satisfy it, verifies the selected recovery set away from the execution host, and permits compute release after that verification succeeds. A later session creates fresh execution capacity and returns retained state to the lab platform that owns topology and recovery semantics. The same session lifecycle can recreate private access to topology nodes for standard management and automation clients while keeping those nodes off the public network.

The model supports multiple continuity levels matched to the operator's required outcome and each platform's native state model. EVE-NG can retain lab definitions together with selected stopped-QEMU overlay state, GNS3 retains project and controller state including writable node disks, and Containerlab uses its native save and recovery semantics within a verified fresh-host lifecycle. The Containerlab path completed real GCP validation through off-host recovery verification, original-host deletion, reconstruction on fresh compute, final lab-health verification and final compute cleanup.

The operational question is:

What must survive before the execution host can safely stop existing?

1. Why continuity and compute lifetime diverge

Lab continuity can require more than a topology definition. Depending on the platform and exercise, continuity may also depend on authorised images, saved configuration, writable node disks, native save products, recovery metadata, or external systems that participate in the lab outcome.

When the execution host is permanent, some of these dependencies remain implicit because local state survives with the machine. Temporary compute makes four decisions explicit:

1. what outcome must be available in the next session;
2. which system is authoritative for the state needed to produce that outcome;
3. what evidence is required before execution capacity can be released; and
4. what must be recreated before the next session is considered usable.

The lifecycle turns those decisions into one control boundary. The execution host remains available while it is required for the selected continuity outcome and becomes eligible for release once that outcome is protected independently of the host.

Established approaches and the remaining gap

Several parts of this problem have established precedents. Netbed and Emulab automated resource allocation and repeatable network experiments [8]. Emulab then addressed a closely related resource-lifetime problem through stateful swapping [13] and distributed checkpointing [14]. CloudLab provides on-demand programmable infrastructure and published profiles for repeatable systems research [9]. ReproZip shows that re-execution elsewhere depends on preserving relevant inputs, dependencies and provenance [10].

Current lab platforms also provide important pieces of the operating model. CML retains node state after a simulation is stopped and releases node CPU and memory while keeping disks and other properties on the CML server [15]. Its external connectors place running topology nodes on networks reachable by management and automation applications outside the lab [16]. Containerlab provides native configuration save, persistent lab-directory, redeploy and snapshot-restore semantics for supported node types [17,18].

Together, those systems leave a cross-platform operating question: when may the execution substrate itself stop existing while preserving the continuity outcome selected for the lab? The innovation introduced by this model is a continuity-aware outer lifecycle that makes that outcome a first-class release condition across heterogeneous lab systems. It binds four operations into one governed path: select the required continuity fidelity, verify the corresponding recovery set away from the execution host, use that proof as the gate for releasing high-resource compute, and reconstruct fresh capacity while returning topology and recovery authority to the native lab platform.

Checkpointing, emulator persistence, node-management protocols and native recovery remain owned by the systems that implement them. The lifecycle adds a cross-platform control layer around those capabilities while preserving each platform's distinct state model and recovery mechanism.

The cost side has a parallel operational basis. The FinOps Framework's Usage Optimization capability states that resources should be used when workloads require them and that optimisation decisions should balance cost with operational risk [11]. Its Planning & Estimating capability recognises trial runs and non-production environments as valid estimation inputs [12]. In this model, cost visibility is joined directly to continuity proof: expensive execution capacity becomes releasable when the selected lab-continuity requirement has been satisfied.

2. Define the continuity requirement before release

Recovery fidelity is selected before the release decision. Different labs can require different outcomes.

Continuity requirement	Retained material	Result on fresh compute
Recreate from declared intent	Topology or project source, configuration inputs, image references and policy	Rebuild the lab from source
Continue from saved configuration or disk state	Declared intent plus saved configuration or writable node state supported by the platform	Reconstruct and continue from the retained state
Resume native runtime state	Declared intent plus a platform-native snapshot or recovery product supported by the node type	Return the recovery product to the owning lab platform
Retain execution for non-portable state	State that remains bound to the execution environment or another continuity mechanism	Keep the host or use the continuity design that owns that state

Rebuilding from source, restoring a writable disk and returning a native snapshot are different continuity contracts. For the remainder of this paper, the retained recovery set means the material required to satisfy the selected continuity requirement.

3. Lifecycle control model

The lifecycle places a verification boundary between preservation and compute release.

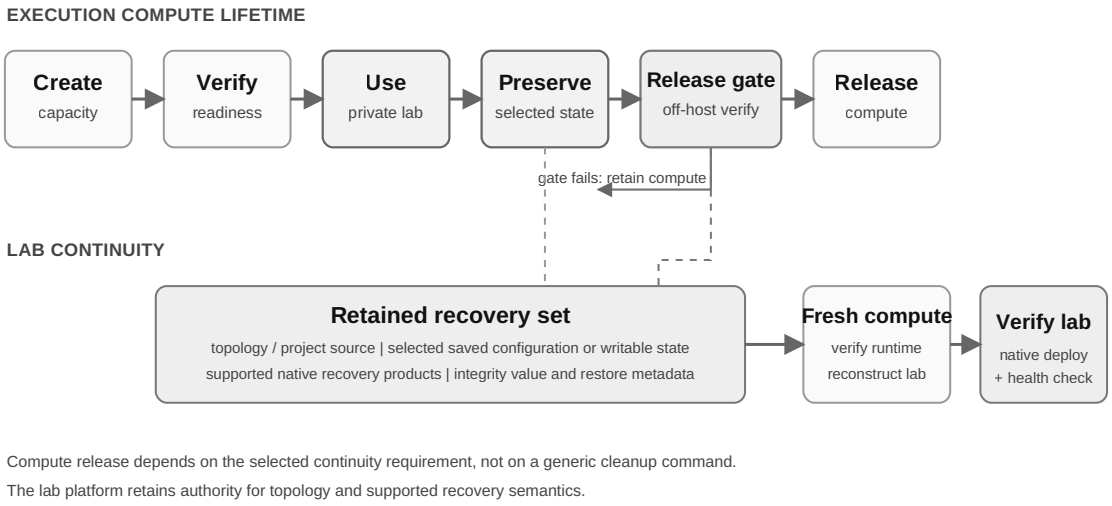

Figure 1. The selected lab continuity requirement is verified away from the execution host before compute is released; later reconstruction uses fresh execution capacity.

3.1 Prepare and activate

Resolve the required capacity, private access boundary, image sources and continuity requirement. Create the execution host and verify connectivity, virtualisation support and lab-platform readiness.

The environment becomes ready when the lab reaches its defined usable state; VM creation is one step in that readiness path.

3.2 Use

The operator uses the lab through the declared private access path. Where device-level access is required, the running lab supplies live management endpoints while durable metadata such as credentials, groups and platform identity remains with its declared authority.

3.3 Preserve

Before compute release, quiesce or stop writers when the selected recovery method requires consistency. Ask the authoritative lab or workload system for its recovery output, collect the retained recovery set, and place it outside the execution host.

3.4 Verify the release gate

Verify the retained copy independently at its destination. The gate establishes that the expected recovery object exists, is readable, matches its recorded integrity value, and carries the metadata required for reconstruction.

A failed verification keeps the release gate closed and the execution host available until an approved continuity path satisfies the requirement.

3.5 Release compute

Once the gate passes, the declared execution capacity may be released according to policy. Success is established by the observed terminal infrastructure state, with command completion serving as one execution signal.

The term compute release describes that boundary. Implementation commands such as destroy or teardown are execution mechanisms within the wider continuity decision.

3.6 Reconstruct and verify

A later session creates sufficient fresh compute, verifies runtime readiness, checks the retained recovery set again before import, and returns the retained material to the system that owns it. The lab platform then performs its native deployment, project start or restore operation.

A final health check demonstrates a useful lab state. A replacement host may have a different resource identity, runtime directories, management addresses and regenerated client access material. The selected continuity requirement determines which properties must remain stable across sessions.

4. Keep authority with the system that owns the state

A common lifecycle preserves the distinct authority of each participating lab system.

Concern	Normal authority	Lifecycle treatment
Topology, nodes and links	Lab platform or operator source	Retain or restore source while preserving native topology semantics
Convergence and node behaviour	Lab platform	Use the platform's native deployment and health path
Base images and licences	Operator or authorised image source	Reference or verify under the applicable distribution rights
Saved configuration or writable node state	Lab-platform or workload-native mechanism	Include according to the selected continuity requirement and platform capability
Native snapshots or recovery metadata	Lab platform	Return the recovery product to the platform that created it
Live management addresses	Current running lab	Rediscover or regenerate after reconstruction
Private access path	Surrounding execution lifecycle	Recreate for the current session
Execution host	Infrastructure lifecycle	Release when the continuity gate passes
Retained recovery set	Approved off-host recovery location	Keep for the selected retention period and verify before use

The lab platform owns the lab. The surrounding lifecycle determines whether the execution environment still needs to exist.

5. Private access is reconstructed with the session

A reconstructed lab becomes useful when operator access returns with it. Access is part of the session lifecycle and can remain session-scoped.

The operator may need a temporary path to the lab platform and, for automation or troubleshooting, a path to individual device-management endpoints. The private management path keeps individual lab devices off the public network.

5.1 Topology nodes remain ordinary management targets

Topology nodes provide an additional control surface. Once a private management path exists, a node inside the topology can be addressed from the operator workstation through the management protocol exposed by that node. A router, switch, firewall or Linux node that exposes SSH, NETCONF/RESTCONF, gNMI, HTTPS or another device API can therefore be used by the same class of automation client that would manage a physical device or a separately hosted virtual device.

The surrounding lifecycle establishes the private path and supplies the current endpoint. The node software remains authoritative for the management service, authentication and protocol behaviour. The current EVE-NG and GNS3 implementation discovers management endpoints and generates scoped SSH and automation client material for that session [21].

Management addresses may change after reconstruction while continuity remains satisfied through live endpoint rediscovery and regenerated session-scoped client material. Durable metadata such as role, platform, credential reference and grouping remains operator-owned or belongs to another authoritative inventory. Generated client inventory is therefore a session view unless another system has explicitly accepted authority for the device record. CML likewise documents external connectivity for network-management and automation applications [16].

6. Compute lifetime has a financial boundary

Temporary cloud labs have a cost-bearing execution window. That window closes when the declared execution compute reaches terminal state.

```
create execution capacity
        |
        |  cost-bearing compute lifetime
        v
verify lab -> use -> preserve -> verify retained recovery set
                                        |
                                        v
                                release compute
```

Runtime cost context supports the release decision through a fixed hourly estimate and elapsed execution duration. Provider billing remains authoritative for realised spend. Retained recovery storage, image repositories, registries, shared networking and other retained resources remain visible as separate cost-bearing components after execution compute is released.

High-resource lab compute therefore becomes deliberately releasable once the selected continuity requirement has been satisfied and verified. The value is the ability to tie resource release to proven continuity rather than to user disconnection or an elapsed timer.

7. One lifecycle across different lab systems

The useful comparison is whether the lifecycle can respect different emulator state models while applying the same release rule.

7.1 EVE-NG

The current implementation has exercised private Google Cloud and Proxmox EVE-NG paths. The primary archive retains learner-created lab definitions under /opt/unetlab/labs. When the selected continuity policy includes stopped node state, the lifecycle stops running QEMU nodes, validates their writable overlays, creates a separately checksummed node-state companion archive and verifies both retained objects before restoration [19].

Base images remain independently managed. Restored QEMU overlays are paired with the matching installed base images rather than carrying those bases inside the checkpoint. This gives EVE-NG both definition-level reconstruction and writable-overlay continuity under one selected recovery set.

7.2 GNS3

The current implementation has exercised private Google Cloud and Proxmox GNS3 paths. The retained set includes project directories and controller metadata; project directories carry topology files, project files and writable node disks [20]. The GNS3 service is stopped while controller state is copied or restored, and restore verifies the recorded SHA-256 before applying retained state. Base images remain separately managed by default.

The selected recovery set governs the release decision independently of original-host lifetime.

7.3 Containerlab: validated GCP lifecycle

The GCP Containerlab lifecycle completed real-environment end-to-end validation before the implementation was merged to Core main through PR #303 [22].

Containerlab remained authoritative for .clab.yml, nodes and links, convergence, native save behaviour and snapshot or restore semantics supported by the selected node types. The surrounding lifecycle owned the private execution host, runtime readiness, recovery policy, off-host verification, compute-release sequencing, fresh-host reconstruction and terminal infrastructure state.

The observed path established that:

1. a private GCP execution host was created and IAP/SSH access and KVM readiness passed;
2. Containerlab 0.78.0 was installed and verified;
3. the supplied topology deployed through native Containerlab and independent health passed;
4. the selected recovery set was produced, copied away from the execution host and checksum-verified there;
5. the original VM was deleted after that recovery gate passed;
6. replacement compute was created with a different resource identity;
7. retained recovery state verified and imported on the fresh host;
8. reconstruction used one native Containerlab deployment and final health passed; and
9. the declared GCP compute was removed at the end of the run.

The validated GCP path demonstrates the complete release-and-reconstruction boundary: recovery is verified off-host, original execution capacity is released, and the lab is reconstructed on fresh compute through native Containerlab behaviour. Recovery fidelity is selected by policy and delivered through the native capabilities of the topology and node kinds in use.

8. Operating boundaries and release conditions

The lifecycle makes the operating boundary explicit for each class of state.

- Runtime state: portability follows the authoritative lab platform's export and restore capabilities. Where the selected continuity outcome requires additional state, the release gate waits for an approved recovery mechanism that protects it.
- Physical conditions: the v1.0 review identified RF state as a concrete external dependency. Neighbouring access points, noise floor and airtime use remain part of the physical environment and are re-established or revalidated separately from the virtual lab.
- External services: third-party systems remain independent dependencies and are revalidated as part of reconstruction when the lab outcome relies on them.
- Image rights: image retention and distribution remain governed by the operator's authorised source and applicable licence terms.
- Address identity: management addresses are session state unless policy explicitly preserves them; reconstruction can rediscover endpoints and regenerate client material.

- Cost: compute release closes the declared execution window while retained storage, registries and supporting services are accounted for separately.
- Recovery verification: the release gate is controlled by the retained recovery set; failed verification keeps execution capacity available or routes the operation through another approved continuity path.

These boundaries make the continuity contract explicit, testable and operationally enforceable.

9. Evidence required for a worked path

A worked implementation should produce evidence for each transition that matters to the continuity claim.

Transition	Minimum useful evidence
Activation -> usable lab	Execution identity, readiness checks and independent lab health result
Use -> preservation	Selected continuity requirement and recovery scope
Preservation -> release gate	Off-host recovery location, integrity value and independent verification result
Release gate -> terminal compute state	Provider or infrastructure evidence that the original execution capacity is absent or intentionally retained
Fresh compute -> reconstructed lab	Different execution identity where applicable, retained-state verification, native lab-platform deployment or restore, and final health result
Final closure	Declared execution resources absent or intentionally retained, with remaining resources reported separately

Where private device automation is part of the worked claim, the evidence should also show that at least one topology node can be reached from an external client through the declared private management path using a node-native management protocol while the device remains off the public network.

The evidence records observed state and keeps production-scale, savings and recovery-fidelity statements tied directly to what was exercised.

10. Worked implementation

HybridOps is the open-source implementation of this lifecycle model. It makes the continuity requirement, retained recovery set, preservation gate, compute release, fresh-capacity reconstruction, private access rebinding and run evidence part of one cross-platform execution contract. The public Capability Map links those capabilities to their design, operating and verification material [7].

The implementation governs the surrounding execution boundary while EVE-NG, GNS3 and Containerlab retain authority for their own topology, node behaviour and native recovery semantics. The product contribution is the continuity-aware lifecycle that operates coherently across those systems.

The current implementation exercises three materially different continuity contracts: EVE-NG lab definitions with selected verified stopped-QEMU overlay state, GNS3 project and controller continuity including writable node disks [19,20], and the validated GCP Containerlab lifecycle in which off-host recovery proof gates original-host deletion before reconstruction on fresh compute [22]. Private topology-node management is implemented for the current EVE-NG and GNS3 paths [21].

11. Practitioner review question

A reviewer can assess the operating boundary directly from the proposition and use the implementation references when deeper inspection is useful.

For a high-resource network lab you would actually operate, what must survive before you would trust the execution host to be released between sessions?

A useful answer may identify a sufficient recovery set, a state type that requires a different continuity mechanism, or a condition that should prevent release. A concrete counterexample is sufficient.

References

1. HybridOps Core.
2. HybridOps platform runbooks.
3. EVE-NG documentation.
4. GNS3 documentation.
5. Containerlab documentation.
6. External technical review of the v1.0 network-lab paper.
7. HybridOps documentation, Capability Map.
8. Brian White et al., An Integrated Experimental Environment for Distributed Systems and Networks, OSDI 2002.
9. Dmitry Duplyakin et al., The Design and Operation of CloudLab, USENIX ATC 2019.
10. Fernando Chirigati, Dennis Shasha and Juliana Freire, ReproZip: Using Provenance to Support Computational Reproducibility, TaPP 2013.
11. FinOps Foundation, Usage Optimization.
12. FinOps Foundation, Planning & Estimating.
13. Prashanth Radhakrishnan, Stateful-Swapping in the Emulab Network Testbed, University of Utah, 2008.
14. Anton Burtsev et al., Transparent Checkpoints of Closed Distributed Systems in Emulab, EuroSys 2009.
15. Cisco, Starting, Stopping, and Wiping Nodes - Cisco Modeling Labs 2.10.
16. Cisco, External Connectors - Cisco Modeling Labs 2.10.
17. Containerlab, save command.
18. Containerlab, deploy command: snapshot restore.
19. HybridOps Core, EVE-NG lab archive and stopped-QEMU state contract.
20. HybridOps Core, GNS3 lab archive contract.
21. HybridOps Core, Device Automation: managed lab access.
22. HybridOps Core, GCP Containerlab lifecycle implementation and validation, PR #303.
23. HybridOps documentation, Network Lab Continuity Across Ephemeral Compute reference scenario.