Build, boot, verify, publish

A measured lifecycle for reusable virtual-machine templates

Jeleel Muibi · HybridOps.Tech
Technical paper · Version 1.0
First published July 2026 · revised 8 August 2026

Abstract

A machine-image build can succeed while the resulting template is still an unsafe infrastructure dependency. The template may not clone or boot as expected, its runtime identity may be unclear, downstream consumers may depend on a copied provider identifier, and replacement or retirement may remain an operator convention.

This paper evaluates a bounded HybridOps lifecycle for that handoff. A declared template passes through readiness and collision checks, construction, a disposable runtime exercise, publication under a logical environment key, downstream state resolution and explicit retirement. Image construction and provider state remain owned by their native systems. HybridOps governs the decision about when the observed result is sufficient to publish and later retire the dependency.

A frozen controlled evaluation completed five fresh observations for each of Ubuntu 22.04, Ubuntu 24.04, Rocky Linux 9 and Windows 11 Enterprise on one Proxmox testbed. All 20 observations completed required clone-and-boot smoke, state publication, reuse, retirement and idempotent retirement. Median time to published state ranged from 7.04 to 13.25 minutes, with a median interval beyond image construction of 0.69 to 1.78 minutes. Storage returned from a maximum sampled 94.45% to the 90.65% baseline after every observation, and no reserved test resource remained. A separate controlled extension completed three state-resolved downstream VM handoffs with required SSH readiness and clean retirement.

The measured boundary is functional infrastructure readiness. It is not a security certification, binary reproducibility result or cross-hypervisor claim. The paper asks whether the additional publication and retirement contract provides useful control where these responsibilities are otherwise split, and where an existing image platform should own the lifecycle directly.

Keywords: machine templates; runtime verification; lifecycle state; operational evidence; publication; retirement.

1. The operational question

Image construction answers an important question: did the declared build procedure complete? It does not necessarily answer whether the result should become a reusable dependency for other infrastructure.

Before publication, an operator may still need to establish:

1. whether the resulting template can be cloned and started;
2. whether a required runtime readiness signal is observable;
3. whether the target identity was reused or deliberately replaced;
4. which logical identity downstream infrastructure should consume;
5. whether validation resources were removed; and
6. how the retained template is retired without coupling that action to ordinary workload teardown.

The proposition evaluated here is narrow:

A retained template should not become a HybridOps infrastructure dependency until the runtime has made the target decision explicit, exercised the result through the configured acceptance gate, published its logical identity and retained a defined retirement path.

This is not a claim that image building, clone testing, artifact metadata or provider state are new. It is a test of the handoff between those established mechanisms.

2. Lifecycle under review

The implemented lifecycle separates construction from publication.

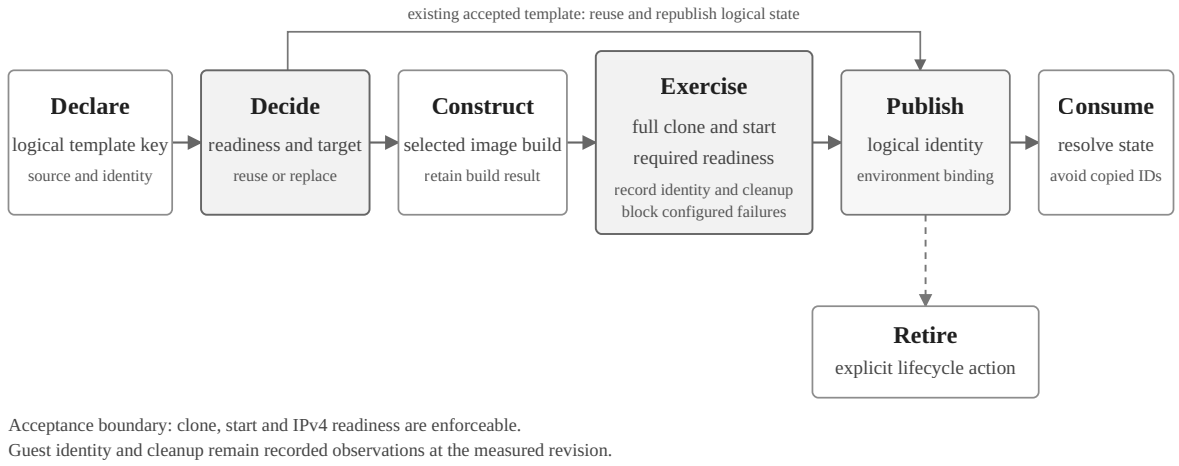

Figure 1. Lifecycle boundary evaluated in this paper. Required runtime checks can block publication. Reuse, replacement and retirement remain explicit lifecycle decisions.

The stages are:

1. **Decide.** Resolve the declaration, environment policy, target identity and reuse or replacement intent.
2. **Construct.** Invoke the selected image implementation and retain its result separately from later runtime evidence.
3. **Exercise.** Create a disposable full clone, start it and obtain the required readiness observation.
4. **Publish.** Expose the tested template under a logical environment key only after the configured acceptance boundary completes.
5. **Consume.** Allow downstream infrastructure to resolve that logical key without embedding the provider identifier in its own declaration.
6. **Retire.** Remove the retained template through an explicit lifecycle action rather than as a side effect of workload teardown.

The native image builder remains responsible for construction. The virtualisation platform remains authoritative for the stored template and clone operations. Guest-agent data remains an observation from the guest. HybridOps records the cross-stage decision and publishes the environment binding used by its downstream modules.

A separate runtime layer is not automatically useful. If an existing image factory already owns runtime acceptance, promotion, consumer resolution, replacement and retirement, duplicating those responsibilities would add a second control surface without a clear boundary.

3. Implemented contract

The controlled evaluation uses the public HybridOps template-image module with a Proxmox implementation. Packer performs image construction. These choices identify the measured implementation; they are not the claim of the paper.

3.1 Declaration, collision and reuse

The operator selects a logical template_key. The implementation maps that key to the appropriate build assets and target identity. Before construction, the runtime resolves credentials and target state and makes one of three decisions:

- the target is absent and construction can proceed;
- the target exists and replacement was not requested, so the existing template is reused and republished; or
- the target exists and replacement was requested, so the planned purge is exposed before construction proceeds.

An existing object therefore does not silently become permission to overwrite it.

A minimal producer and consumer declaration is shown below. The consumer names the producer state and does not contain a provider VM identifier.

```
# Producer

template_key: "ubuntu-22.04"
rebuild_if_exists: false
post_build_smoke:
  enabled: true
  required: true

# Consumer

template_state_ref: "core/onprem/template-image#template_image_ubuntu_22_04"
template_key: "ubuntu-22.04"
```

3.2 Runtime exercise and publication

When required smoke is enabled, the implementation creates a disposable full clone from the retained template, starts it, waits for guest-agent IPv4 readiness, records available guest operating-system information and attempts to remove the validation VM.

The acceptance boundary at the inspected revision is deliberately limited:

Condition	Recorded result	Publication effect
Construction command fails	Return code and retained log	Block publication
Required clone or start fails	Provider task and smoke step	Block publication
Required guest-agent IPv4 readiness times out	Timeout and smoke step	Block publication
Guest OS identity is unexpected	Guest-agent fields	Record only
Validation-resource cleanup fails	Cleanup outcome	Record; not a uniform hard rejection
Required gates complete	Structured result and logical template identity	Publish

The word *verified* in this paper refers only to the named observations. It does not imply vulnerability assessment, application correctness or strict guest identity acceptance.

3.3 Consumer handoff and retirement

After the configured gate completes, the runtime publishes the template key, name and provider identity into the named environment state. A downstream VM module can resolve that state reference before provider calls rather than copy the provider identifier into its own declaration.

This state is not a general artifact registry. It is an environment-scoped binding for infrastructure managed by the same runtime. Larger estates may prefer an existing artifact platform with promotion channels, provenance and organisation-wide policy.

Template retirement remains separate from ordinary workload teardown. Destroy resolves the declared template identity, treats an already absent template as an idempotent result and otherwise removes the retained dependency through the declared lifecycle.

4. Controlled evaluation

The reported results use one frozen HybridOps release, one isolated Proxmox test environment and a fixed measurement procedure. The success matrix is the basis of the paper; earlier development runs are retained as preliminary evidence but are not used as a success-rate denominator.

4.1 C03 fresh-build lifecycle

The primary matrix covered four operating-system families:

- Ubuntu 22.04;
- Ubuntu 24.04;
- Rocky Linux 9; and
- Windows 11 Enterprise.

Each condition was repeated five times. Every observation included a fresh build, required disposable-clone smoke, publication, existing-template reuse, explicit retirement and a second idempotent retirement.

For every run, the evaluation recorded construction and total duration, smoke outcomes, publication state, residual template and validation VM identifiers, storage use, reuse behaviour and manual interventions. Five observations per condition support descriptive comparison within this testbed, not a general performance claim.

A predefined fault block covering source integrity, identity collision, guest readiness, clone capacity, cleanup and guest-family mismatch was not executed in C03. Successful observations therefore cannot be used to infer fault-detection coverage.

4.2 C04 producer-consumer handoff

A second controlled extension created one fresh producer-to-consumer observation for Ubuntu 22.04, Ubuntu 24.04 and Rocky Linux 9. Each downstream VM resolved the published producer state without a hardcoded template VMID, passed required SSH readiness, retired cleanly and was followed by producer retirement and an idempotent second destroy.

5. Results

5.1 Fresh-build lifecycle

All 20 C03 observations completed the configured lifecycle.

Template	Completed	Median construction	Median interval after construction	Median total to published state	Residual validation VMs
Ubuntu 22.04	5/5	8.73 min	0.84 min	9.56 min	0
Ubuntu 24.04	5/5	8.34 min	0.72 min	9.11 min	0
Rocky Linux 9	5/5	6.33 min	0.69 min	7.04 min	0
Windows 11 Enterprise	5/5	11.37 min	1.78 min	13.25 min	0

Across the 20 observations, the highest sampled thin-pool use was 94.45% against a 96.50% stop boundary. Every observation returned to the exact 90.65% baseline. The final provider scan found no VM in the reserved template or validation ranges. All apply, reuse, retirement and idempotent-retirement operations returned success.

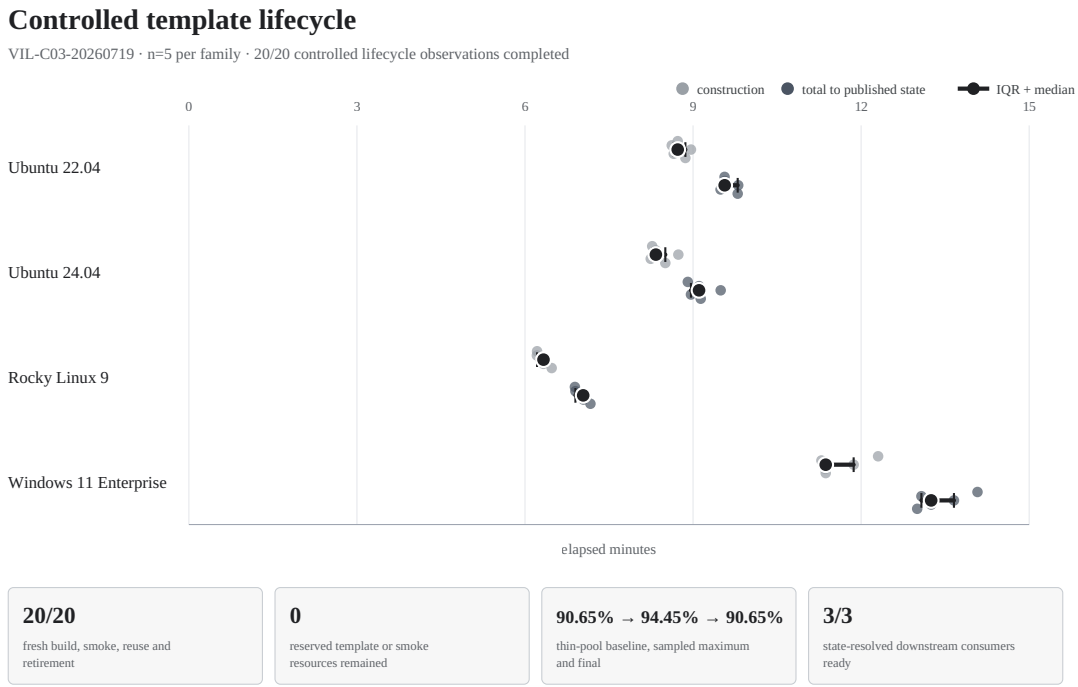

Figure 2. Controlled C03 timings generated from the sanitised result register. The lower summary records lifecycle completion, residual resources, storage recovery and downstream state handoff.

The measured interval beyond construction is important because the additional boundary is not free. In this testbed its family median was 0.69 to 1.78 minutes. The value of the extra checks and publication semantics should be judged against that operational cost rather than assumed.

The Windows guest-agent record also preserves an identity discrepancy: version reported Microsoft Windows 11 while pretty-name reported Windows 10 Enterprise Evaluation. Strict family matching was not enforced by the measured release. The observation therefore demonstrates the runtime path and retained evidence, not strict identity acceptance.

5.2 Downstream handoff

All three C04 consumers resolved producer state and reached the required readiness condition.

Template	State-resolved consumer	Required SSH readiness	Consumer and producer retirement	Residual resources
Ubuntu 22.04	1/1	1/1	1/1	0
Ubuntu 24.04	1/1	1/1	1/1	0
Rocky Linux 9	1/1	1/1	1/1	0

The C04 register retains two relevant caveats. The Ubuntu 22.04 observation recorded a non-fatal primary-IP synchronisation conflict even though provider readiness and the declared lifecycle completed. The Rocky Linux storage sampler exited early because of a recorder bug, so its reported peak is limited to the consumer phase. These observations remain in the public sanitised record rather than being removed from the result set.

6. What the results establish

The controlled results support a limited conclusion: in the measured Proxmox environment, the implemented lifecycle can move a declared template through construction, required runtime exercise, logical publication, downstream state resolution and explicit retirement while retaining structured evidence of each stage.

They do **not** establish:

- security posture, vulnerability status or application correctness;
- strict operating-system identity acceptance;
- behaviour under the predefined fault matrix;
- equivalent behaviour on another hypervisor or cloud image service;
- binary reproducibility of separate image builds; or
- independent external replication.

The evaluation also does not show that every environment needs this runtime boundary. Where one image platform already owns acceptance, promotion, consumer selection, replacement and retirement, that platform is the simpler owner. The HybridOps boundary is relevant where those decisions would otherwise be split across construction output, provider procedure, copied identifiers and consumer configuration.

7. Request for technical review

This paper seeks implementation-level criticism of the measured boundary. Useful review questions are:

1. Does the decision, runtime exercise, publication and retirement lifecycle add meaningful control where image construction and provider state are otherwise separate?
2. Which observations should be hard acceptance gates before a template identity is published?
3. Is an environment-scoped logical state handoff appropriate for infrastructure managed by one runtime, or should consumers resolve only through a dedicated artifact platform?
4. Which fault injection or independent replication test would most strongly challenge the current result?

A finding that an existing image platform already owns the complete lifecycle is a valid review outcome.

8. Implementation and evidence availability

The implementation and controlled result registers are public:

- HybridOps Core
- Template-image module
- Measured image implementation
- C03 controlled lifecycle evidence
- C04 producer-consumer evidence
- Preliminary evidence register

The public bundles contain sanitised structured projections and checksums. Raw records are not published because they contain local paths, account metadata, provider task identifiers and infrastructure addresses. The frozen protocol, result registers and checksums make the evaluation inspectable; an independent rerun remains the strongest next test of external validity.

References

Research and image-lifecycle sources

1. Lamb, C. and Zacchiroli, S. (2022). Reproducible Builds: Increasing the Integrity of Software Supply Chains. *IEEE Software*, *39*(2), 62-70.
2. Spichkova, M., Li, B., Porter, L., Mason, L., Lyu, Y. and Weng, Y. (2020). VM2: Automated security configuration and testing of virtual machine images. *Procedia Computer Science*, *176*, 3610-3617.
3. Yamato, Y. (2015). Automatic verification technology of software patches for user virtual environments on IaaS cloud. *Journal of Cloud Computing*, *4*, article 4.
4. Suneja, S., Koller, R., Isci, C., de Lara, E., Hashemi, A., Bhattacharyya, A. and Amza, C. (2017). Safe Inspection of Live Virtual Machines. *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 97-111.
5. Saurabh, N., Remmers, J., Kimovski, D., Prodan, R. and Barbosa, J. G. (2019). Semantics-Aware Virtual Machine Image Management in IaaS Clouds. *2019 IEEE International Parallel and Distributed Processing Symposium*, 418-427.

Primary technical sources

1. HashiCorp. Packer validate command reference.
2. HashiCorp. Packer Proxmox ISO builder.
3. HashiCorp. Packer manifest post-processor.
4. HashiCorp. HCP Packer.
5. QEMU. QEMU Guest Agent Protocol Reference.

Technical appendix: implementation evidence excerpts

The screenshots below are retained as implementation-level evidence. They are supporting material, not the basis of the controlled success denominator.

Figure A1. Resolved input and preflight output from a retained development run. It shows that replacement is an explicit decision rather than an implicit consequence of target existence.

Supporting evidence (continued)

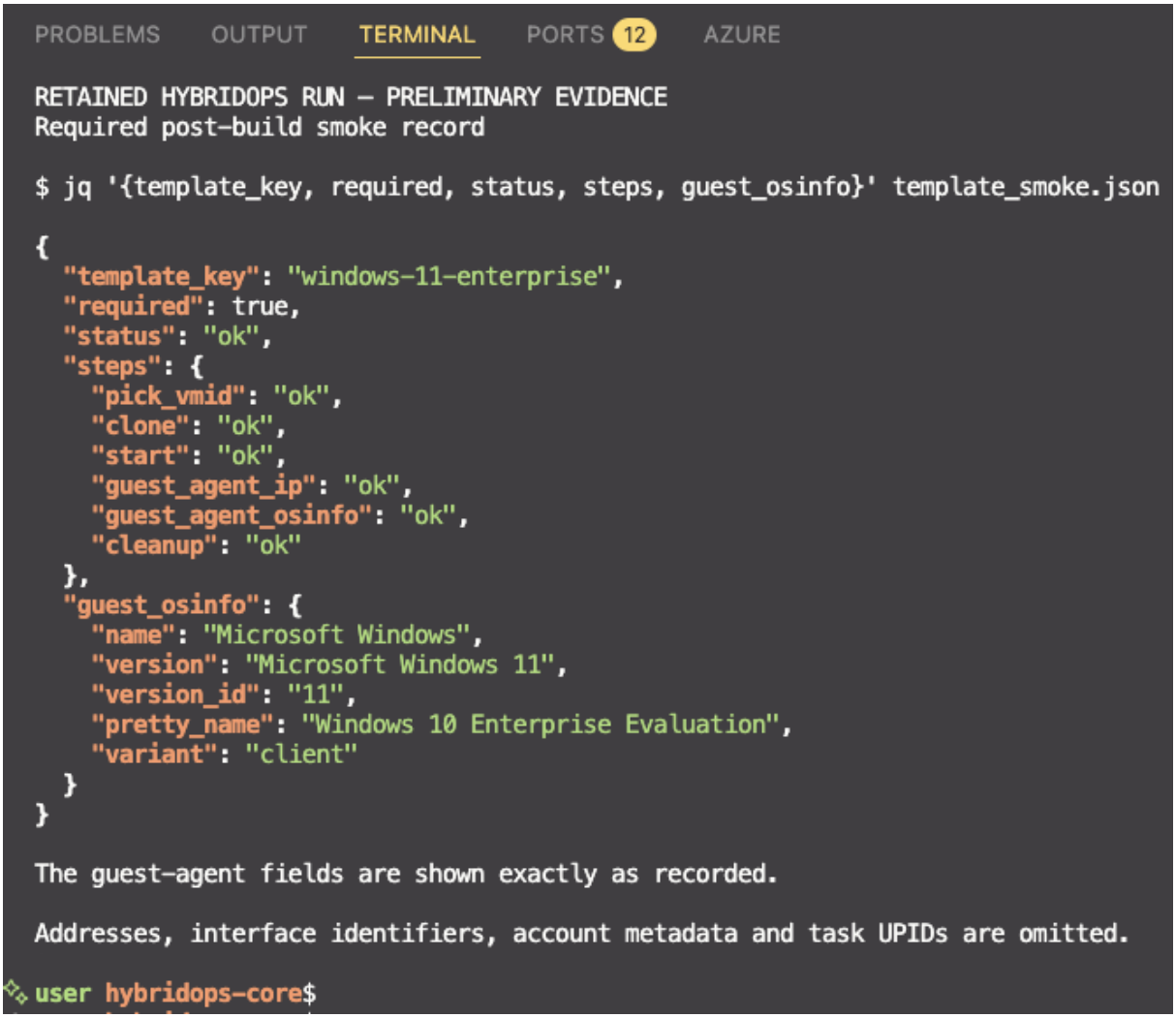

Figure A2. Structured runtime evidence from the same retained run. The identity discrepancy is preserved rather than normalised away.

Supporting evidence (continued)

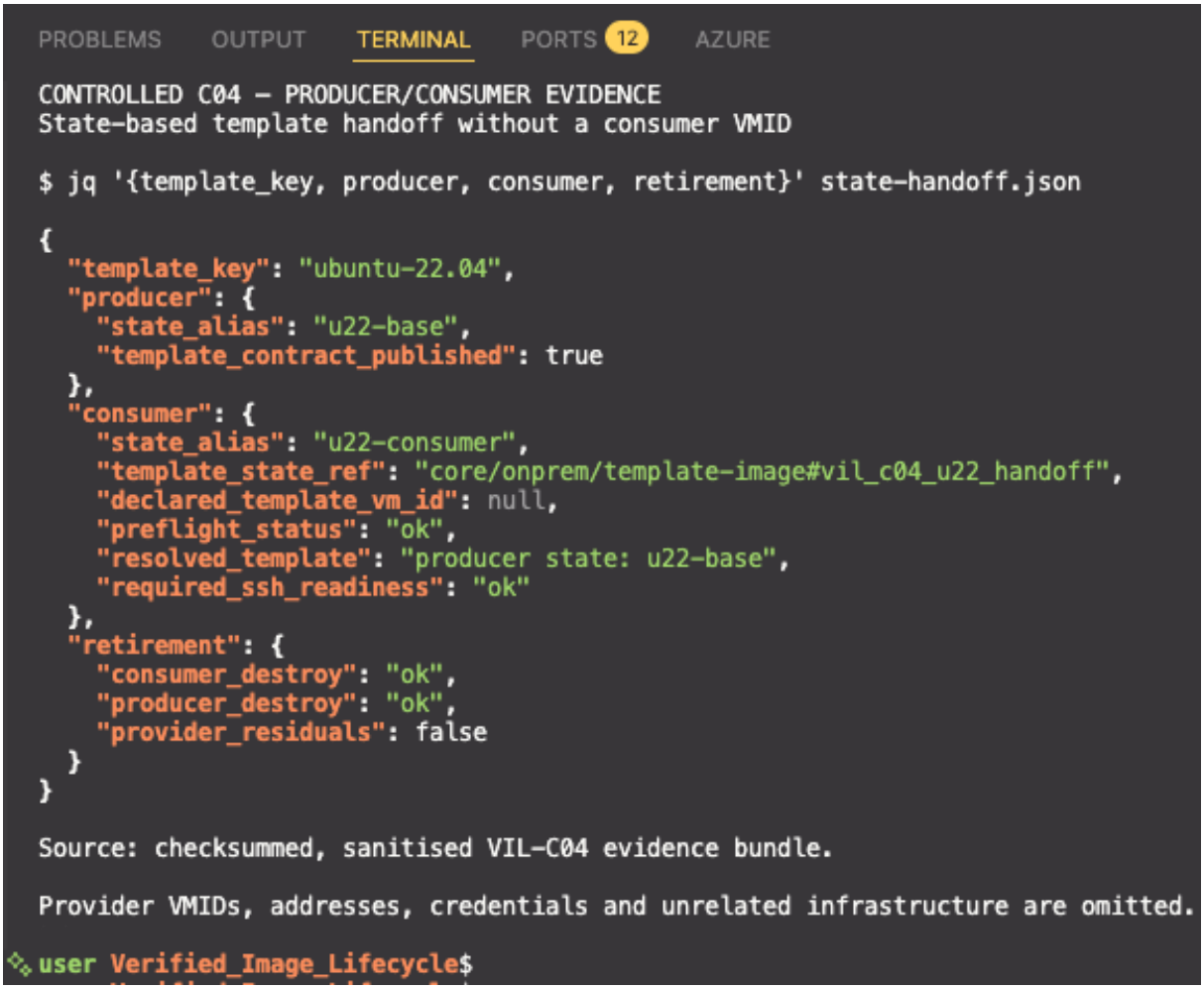

Figure A3. Controlled C04 producer-consumer evidence generated from the sanitised, checksummed result projection.