Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 4 June 2026

M. Gaikwad

1 December 2025

Agent Persistent State Profile
draft-gaikwad-aps-profile-00

Abstract

Autonomous agents increasingly maintain durable persistent state containing user preferences, embedding vectors, safety logs, intermediate reasoning steps, and audit traces. Today, agent frameworks treat storage as a generic file system, while storage administrators treat agents as stateless virtual machines. This "layer mismatch" leads to fragility, poor performance, and privacy risks.

The Agent Persistent State (APS) Profile defines an experimental, vendor-neutral storage service class for durable agent state. APS emphasizes *compliance*: ensuring that memory associated with a specific user or agent identity can be retained, audited, and cryptographically erased. APS also addresses high-frequency small I/O, vector index workloads, crash consistency, and Kubernetes/CSI [CSI] integration.

APS introduces a Usage Class ("AgentPersistentState"), a versioned PersistentStateLineOfService schema, guidance for container orchestration systems, non-normative bindings for Swordfish [Swordfish] and Redfish [Redfish], and considerations for multi-tenancy. APS is intended as an Experimental RFC to gather implementation feedback prior to any standards-track work.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 4 June 2026.

Table of Contents

1. Introduction

AI agents are entering a new operational phase. Early systems focused on prompt engineering and model context. Modern deployments involve several of autonomous agents with distinct identities and long-lived state. Persistent state stores embeddings, preferences, memory, safety constraints, and reasoning artifacts that must remain durable and compliant across failures.

Current practice reveals a structural mismatch: agent developers assume implicitly that "somewhere" there is a low-latency, compliant store; storage administrators assume short-lived, stateless applications. This mismatch leads to data loss, privacy violations, tail-latency spikes, and unpredictable behavior.

Reviewers have observed that this draft is premature but can be possibly timely: as deployments scale to thousands of agents, storage semantics will become critical infrastructure. By the time APS reaches maturity, the need is expected to be urgent.

The key words MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY in this document are to be interpreted as described in [RFC2119] and [RFC8174] when, and only when, they appear in all capitals.

2. Terminology

Agent: An autonomous software entity with a distinct identity and a multi-session lifecycle. Agents maintain durable state across restarts or container rescheduling, motivating specialized persistent storage semantics.

Persistent State: Durable data associated with an agent identity, including embeddings, preferences, caches, audit logs, and vector indexes.

APS: Agent Persistent State Profile defined in this document.

Usage Class: A tag applied to storage resources indicating the expected workload. This document defines AgentPersistentState.

Line of Service (LoS): A structured description of storage behavior.

Forgetting Policy: A policy that causes persistent state to be deleted or scrubbed according to age, event triggers, or external legal/compliance requests (for example, scheduled deletion of records older than a given retention period).

3. Landscape of Agent Frameworks and Persistent State Needs

Agent frameworks today implement persistence at the application layer, leading to fragility. The following table summarizes common patterns as of 2025; it is illustrative and non-exhaustive.

Framework	Current Mechanism	Pain Point	APS Benefit
LangChain (LangGraph)	Postgres/SQLite/ Redis checkpoints	Locking, tail latency from high-frequency checkpointing	SmallRandom profile; advisory locking hints
AutoGPT/ BabyAGI	JSON files and local workspace directories	Crash can corrupt state; agent "forgets" everything	CrashConsistency and journaling recommendations
LlamaIndex	Vector stores (e.g., HNSW) plus document stores	Noisy-neighbor throttling during bulk index I/O	Vector-phase-aware profiles; tail-latency awareness
CrewAI	Local SQLite or similar shared state	Multi-agent persistence fragile across failures	ReplicationScope for AZ/region resilience

Table 1

The structural problems across frameworks validate the need for APS: requirements for durability, performance, and compliance must move downward into the infrastructure layer in a standard form.

4. APS Data Model

APS defines a new Usage Class:

UsageClass: "AgentPersistentState"

APS applies to storage abstractions that present block or filesystem semantics to agents and their orchestrators, including volumes provisioned via Kubernetes PersistentVolumes or surfaces exposed through SNIA/DMTF management schemas. Applying APS directly to pure object stores or HTTP-based APIs is outside the scope of the normative requirements in this document.

5. PersistentStateLineOfService Schema

The PersistentStateLineOfService structure describes a single APS profile. Fields marked OPTIONAL are advisory and MAY be omitted. A subset of fields constitutes the core APS compliance baseline and MUST be present and honored for any profile that claims APS compliance.

```
PersistentStateLineOfService {

  ApsVersion: string           # e.g., "1.0"

  Id: string
  Name: string
  Purpose: "AgentPersistentState"

  IOProfile: enum {
      SmallRandom,
      MixedRandom,
      MetadataHeavy,
      VectorIndex
  }

  # Optional refinement for vector workloads:
  VectorPhase: enum {
      Ingest,
      Query,
      Mixed
  } OPTIONAL

  CapacityRangeGiB {
    MinGiB: integer
    MaxGiB: integer
    RecommendedStepGiB: integer
  }

  Performance {
    MinIOPS: integer
    MaxLatencyMsP95: number
    MaxLatencyMsP99: number
```

```
    RecommendedIOSizeKiB: integer
        # Embeddings and vectors may use 16-64KiB I/O sizes
  }

  Durability {
    DurabilityClass: enum { Standard, High, VeryHigh }
    ReplicationScope: enum { LocalAZ, MultiAZ, MultiRegion }
  }

  CrashConsistency {
    RequiresWriteOrdering: boolean
  }

  Retention {
    RetentionDays: integer
    ForgettingPolicySupported: boolean
  }

  Privacy {
    ContainsUserFacingDataExpected: boolean

    MinDeletionSemantics: enum {
      BestEffort,
      CryptographicErase,
      MediaScrub
    }

    ErasureScope: enum {
      Volume,          # Whole LUN / filesystem
      FilesystemTree,  # Directory / inode subtree
      ObjectPrefix,    # Prefix subset for object-backed systems
      ApplicationKey   # Crypto handled above storage
    }

    MaxEraseLatencyHours: integer
  }

  ConcurrencyHints {        # OPTIONAL, advisory only
    AdvisoryLockingSupported: boolean
  }
}
```

The core APS compliance baseline consists of the following fields, which *MUST* be present and reflect behavior meaningful to the abstraction being exposed:

* IOProfile

* CapacityRangeGiB (all members)
* Performance.MinIOPS
* Performance.MaxLatencyMsP95
* Durability.DurabilityClass
* Durability.ReplicationScope
* CrashConsistency.RequiresWriteOrdering
* Retention.RetentionDays
* Retention.ForgettingPolicySupported
* Privacy.MinDeletionSemantics
* Privacy.ErasureScope
* Privacy.MaxEraseLatencyHours

Providers *MUST NOT* advertise an APS-compliant profile that omits or ignores these core fields. Additional fields MAY be omitted if they are not meaningful for the abstraction (for example, advisory locking hints on raw block volumes).

In environments that provision storage in terms of throughput (for example, MiB/s) rather than IOPS, implementers MAY derive an effective MinIOPS from documented throughput and average I/O size. One reasonable approach is:

MinIOPS (ThroughputMiBps * 1024) / RecommendedIOSizeKiB

APS does not constrain how this effective value is calculated, but providers SHOULD document the relationship between throughput and MinIOPS.

5.1. IOProfile Behavior (Non-Normative)

IOProfile values describe expected workload envelopes. This section is non-normative guidance for implementers and consumers. APS profiles are provisioned at volume-creation time, while agent workloads may evolve dynamically. Consumers SHOULD treat IOProfile as a coarse signal for provisioning decisions, not a real-time optimization mechanism.

Some IOProfile values (for example, VectorIndex) include an OPTIONAL VectorPhase field. VectorPhase refines expected workload behavior but does not override the baseline characteristics of the underlying IOProfile.

SmallRandom: Dominated by 4-16 KiB random reads and writes, typical for agent checkpointing, key-value state blocks, preference updates, and log fragments. Storage SHOULD provision enough low-latency IOPS to avoid tail-latency amplification during frequent agent state flushes.

MetadataHeavy: Characterized by high rates of metadata changes (directory operations, inode updates, journal commits). Suitable for agents producing many small files or maintaining structured reasoning traces. Storage SHOULD support efficient metadata journaling and crash-consistent mounts.

MixedRandom: A balanced workload combining small random I/O with occasional larger sequential batches. Appropriate when agents interleave checkpointing, replay logs, and batched vector updates.

VectorIndex: Represents workloads dominated by embedding ingestion, vector construction, nearest-neighbor search, or index compaction. Providers SHOULD document which vector phase(s) their configuration optimizes. The OPTIONAL VectorPhase enum has the following meanings:

* Ingest — optimized for high-throughput sequential or semi-sequential writes during bulk embedding generation, HNSW edge linking, or batch import.

* Query — optimized for low-latency random reads and small-range fetches typical of approximate nearest neighbor search workloads such as DiskANN-style index queries.

* Mixed — balanced for deployments where ingest, query, and compaction phases interleave or shift over time. Providers SHOULD state which tradeoffs are made (write-friendly vs. read-friendly).

Because vector workloads are phase-shifting, consumers SHOULD NOT assume that a single VectorPhase value can optimize all stages of the index lifecycle.

5.2. DurabilityClass Guidance (Non-Normative)

DurabilityClass values are intentionally abstract. As non-normative guidance, providers might interpret them as:

* Standard: durability comparable to typical single-region cloud block volumes.

* High: durability comparable to replicated volumes or erasure-coded storage (for example, multiple independent copies within a region or across zones).

* VeryHigh: durability approaching or exceeding widely replicated object storage classes.

Exact "nines" targets are out of scope and MAY vary by provider.

6. Crash Consistency Requirements

All APS-compliant resources *MUST* provide metadata consistency after power loss or crash. For the purposes of this document, metadata consistency means that the filesystem or block namespace remains mountable without manual repair tools and that directory and inode structures do not exhibit silent corruption that would cause loss of reachability for existing files.

The RequiresWriteOrdering flag indicates that applications rely on storage honoring write ordering or barrier semantics (for example, flush/fence operations) to maintain write-ahead logging or similar invariants. When true, providers *MUST* document the ordering guarantees and the mechanisms available to applications.

Implementations supporting high-frequency agent checkpointing SHOULD document whether O_DIRECT, O_SYNC, or equivalent durability hints are honored by the underlying storage stack.

Applications are expected to rely on standard filesystem atomic operations (such as POSIX rename) for batched updates. APS does not define or negotiate additional transactional semantics.

7. Privacy, Replication, and Erasure

The MinDeletionSemantics, ErasureScope, and MaxEraseLatencyHours fields describe the minimum deletion and erasure behavior for data stored under the profile. When combined with ReplicationScope, providers *MUST* interpret these fields as applying to all replicas of the primary data in scope.

The MaxEraseLatencyHours field *MUST* be a strictly positive integer. A value of zero is reserved and MUST NOT be used. Deployments that effectively provide synchronous erasure MAY represent this with a small non-zero value and SHOULD document the expected behavior.

For example, if MinDeletionSemantics is set to CryptographicErase and ReplicationScope is MultiRegion, then cryptographic erasure of all regions *MUST* complete within MaxEraseLatencyHours, unless the provisioning request explicitly accepts weaker behavior.

APS describes behavior for primary data under the profile. Snapshots, backups, and other disaster recovery mechanisms MAY retain older copies of data beyond MaxEraseLatencyHours. End-to-end compliance for such copies is outside the scope of this document and MUST be addressed by higher-level data lifecycle policies.

The ErasureScope field indicates the granularity at which cryptographic or destructive erase is expected to act:

* Volume: erasure is performed at the level of a whole volume, LUN, or filesystem. Destroying the associated key or media affects all data hosted on that volume.

* FilesystemTree: erasure is performed at the level of a directory or inode subtree. For example, implementations MAY use per-directory filesystem encryption mechanisms (such as Linux fscrypt) where destroying the key renders only that subtree unreadable.

* ObjectPrefix: erasure is scoped to a subset of objects under a prefix or bucket namespace. This is primarily informative for object-backed systems that expose filesystem views or may be the basis of future APS profiles for object-native agent state. Profiles that target purely block or filesystem abstractions MUST NOT use ObjectPrefix.

* ApplicationKey: cryptographic erase is implemented above the storage layer. The storage system provides generic persistence, and an application, sidecar, or agent runtime manages keys and performs crypto-shredding by destroying them.

When MinDeletionSemantics is set to CryptographicErase and ErasureScope is Volume, FilesystemTree, or ObjectPrefix, the storage or filesystem layer is responsible for ensuring that destruction of the relevant cryptographic material renders the targeted scope unreadable without affecting adjacent scopes beyond what is documented.

When MinDeletionSemantics is CryptographicErase and ErasureScope is ApplicationKey, the storage layer is not aware of per-agent keys. In this case, APS simply records that cryptographic erase is expected to be achieved via application- or sidecar-managed key lifecycles. Operators MUST NOT interpret such a profile as evidence of storage-controller-level key separation.

Storage backing APS MAY be multi-tenant. Providers *SHOULD* document whether cryptographic erase is implemented via volume-level keys, per-tenant keys, filesystem-level keys, or other mechanisms, and what the effective erase granularity is.

There is intentionally no single mandatory minimum for MinDeletionSemantics. A provider that cannot implement CryptographicErase MAY still advertise an APS profile using BestEffort semantics, but such profiles are generally unsuitable for compliance-critical deployments and SHOULD be clearly documented as such.

7.1. Non-Normative Implementation Patterns

The following patterns illustrate how different implementations might realize finer-grained erasure scopes in practice:

FilesystemTree via fscrypt: A CSI driver or agent runtime mounts a shared volume and creates per-agent directories (for example, /data/agent-a, /data/agent-b). Each directory is protected with a distinct filesystem encryption key (such as a Linux fscrypt policy). To erase a specific agent’s data, the orchestrator destroys that directory’s key in a KMS. The APS profile would set ErasureScope=FilesystemTree and MinDeletionSemantics=CryptographicErase.

Sub-volume Constructs (vVols, Qtrees): Enterprise arrays that support lightweight logical containers inside a physical pool can assign unique keys to those constructs. An APS profile might then represent each logical container as having ErasureScope=Volume from the perspective of the host filesystem, while the array internally multiplexes many such logical volumes.

ApplicationKey Sidecar: When the underlying storage provides only coarse-grained encryption, an application or sidecar may intercept I/O, encrypt data with per-agent keys, and write to a shared volume. Destroying the per-agent key achieves cryptographic erase for that agent’s data. An APS profile for such a deployment would use ErasureScope=ApplicationKey.

These patterns are examples only. APS does not mandate a particular implementation, but seeks to make the erasure scope explicit so operators and frameworks understand where compliance responsibilities sit.

8. Multi-Tenancy and Volume Scalability

APS does *not* require a 1:1 mapping between agents and volumes. Volume-per-agent designs risk exhausting controller limits at scale. Instead, APS is intended to describe the behavior of shared storage classes that MAY host many agents.

Implementations are encouraged to separate cryptographic and policy granularity from volume granularity. For example, a single APS volume may host multiple agents while still satisfying erasure and isolation requirements at a filesystem-tree or application-key level, as indicated by ErasureScope.

9. Block and File Applicability

APS is intended for storage abstractions that present block or filesystem semantics to agents. Some fields are primarily relevant to block devices (for example, RecommendedIOSizeKiB), and others are primarily relevant to filesystems (for example, AdvisoryLockingSupported).

Providers *MAY* omit non-core fields that are not meaningful for the abstraction they expose. Core fields listed in Section Section 5 MUST be honored.

10. Identity and Control Plane Role

APS discusses persistent state in terms of "agent identity", but does not require storage protocols (for example, NVMe, iSCSI, NFS) to carry per-agent identifiers on individual I/O operations. In most deployments, storage controllers see host or initiator identifiers, not agent IDs.

Identity mapping and scoping are therefore the responsibility of the control plane and agent platform (for example, an orchestrator that binds particular agents or pods to specific volumes, filesystems, or filesystem subtrees). APS describes the properties of those storage constructs; it does not define new protocol headers or on-the-wire identity fields.

11. Kubernetes and CSI Mapping

Kubernetes and CSI are expected to be primary consumers of APS. This section provides guidance for CSI driver integrations. It is informative for the CSI specification itself.

A CSI driver that claims to be "APS-aware" *MUST* support a documented mechanism to pass APS hints from higher-level configuration (for example, StorageClass parameters) to backend profiles. One possible mapping is:

* parameters["aps.storage.k8s.io/ioProfile"] → IOProfile

* parameters["aps.storage.k8s.io/privacy.minDeletion"] → MinDeletionSemantics

* parameters["aps.storage.k8s.io/privacy.erasureScope"] → ErasureScope

* parameters["aps.storage.k8s.io/privacy.maxEraseHours"] → MaxEraseLatencyHours

* parameters["aps.storage.k8s.io/durability.replicationScope"] → ReplicationScope

The recommended "aps.storage.k8s.io/" prefix reduces the likelihood of collisions with vendor-defined parameters. Alternative key names MAY be used if they are clearly documented by the CSI driver and its consumers.

12. Bindings and Profile Discovery

APS MAY be expressed in existing schemas as follows:

Swordfish: Implementations expressing APS profiles in Swordfish SHOULD use ClassOfService with UsageClass="AgentPersistentState", and attach a vendor-defined or standardized PersistentStateLineOfService extension carrying APS fields. Discovery of available APS profiles then follows Swordfish mechanisms for enumerating lines of service.

Redfish: Implementations exposing APS via Redfish SHOULD use OEM extensions on StorageService and Volume resources to represent PersistentStateLineOfService fields, together with UsageClass="AgentPersistentState" where applicable. Discovery would use Redfish resource enumeration.

CSI: APS hints MAY be conveyed through StorageClass parameters by convention, as described in Section Section 11. A higher-level control plane (for example, a storage operator) is expected to map available backend profiles to supported StorageClasses.

The details of profile discovery are intentionally left to existing management and orchestration mechanisms; APS focuses on the shape and semantics of individual profiles.

Future work MAY explore bindings for object-native stores, for example, using object prefixes and S3 Select-like mechanisms to host agent state directly in object storage, based on the ObjectPrefix erasure scope.

13. Versioning and Extensibility

The ApsVersion field in PersistentStateLineOfService identifies the APS version the profile conforms to (for example, "1.0").

Future versions of APS *MAY* add new OPTIONAL fields. Profiles that set ApsVersion to "1.x" MUST remain backward compatible for all fields defined in version 1.0, and consumers *MUST* ignore unknown fields rather than treating them as errors.

14. Security Considerations

APS-compliant resources are expected to host sensitive agent state, including user-identifiable data and behavior-derived embeddings. Implementations *SHOULD* support encryption at rest, strong identity scoping in the control plane, and cryptographic erasure according to policy.

15. Privacy Considerations

Enterprises may need to demonstrate that agent memory for a specific user was deleted in accordance with regulation (for example, GDPR). APS provides explicit deletion semantics, an erasure scope, and a latency bound to support such compliance, but implementation details and audits remain the responsibility of the operator and any higher-level governance systems.

16. Limitations and Future Work

This document does not define a conformance test suite or reference implementation. Future work MAY include tests for APS-aware CSI drivers, managed vector databases, and storage arrays.

APS does not provide guidance on how agents should degrade gracefully when APS-compliant storage is unavailable. Agent frameworks are responsible for fallback behavior (for example, operating in a stateless mode).

APS does not model cost trade-offs explicitly. Operators are expected to balance erase latency, replication scope, erasure scope, and storage-class cost.

APS does not attempt to model differential privacy, federated learning, or higher-level alignment techniques. Those may rely on APS semantics but are outside the scope of this profile.

A normative JSON Schema for PersistentStateLineOfService is intended for a future revision to enable automated validation and tooling.

17. IANA Considerations

This document requests that IANA create an "APS UsageClass and Profile Registry" containing the following initial values:

UsageClass:

* AgentPersistentState

IOProfile:

* SmallRandom
* MixedRandom
* MetadataHeavy
* VectorIndex

ErasureScope:

* Volume
* FilesystemTree
* ObjectPrefix
* ApplicationKey

No additional IANA actions are required by this document.

18. References

18.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

18.2. Informative References

[Swordfish] SNIA, "Swordfish Storage Management Specification", 2024.

[Redfish] DMTF, "Redfish Scalable Platforms Management API", 2024.

[CSI] Kubernetes Community, "Container Storage Interface (CSI)", 2024.

Appendix A. Example APS Profiles (Non-Normative)

This appendix provides three example PersistentStateLineOfService instances. These are illustrative and non-normative. They are intended to show how capacity, performance, durability, and privacy settings can be combined for common agent scenarios.

A.1. APS-Standard (Checkpoint-Oriented)

APS-Standard is intended for general agent checkpointing and short- to medium-term memory. It favors small random I/O, moderate capacity, and relatively tight tail latency so that frequent state saves do not break conversational responsiveness. The 8 KiB recommended I/O size and 3000 IOPS target are representative of a busy checkpointing workload where many agents periodically flush small state blobs or logs.

```
   PersistentStateLineOfService {
     ApsVersion: "1.0"
     Id: "aps-standard-1"
     Name: "APS-Standard-Checkpoint"
     Purpose: "AgentPersistentState"

     IOProfile: SmallRandom

     CapacityRangeGiB {
       MinGiB: 5
       MaxGiB: 100
       RecommendedStepGiB: 5
     }

     Performance {
       MinIOPS: 3000
       MaxLatencyMsP95: 20
       MaxLatencyMsP99: 50
       RecommendedIOSizeKiB: 8
     }

     Durability {
       DurabilityClass: High
       ReplicationScope: MultiAZ
     }

     CrashConsistency {
       RequiresWriteOrdering: true
     }

     Retention {
       RetentionDays: 30
       ForgettingPolicySupported: true
     }

     Privacy {
       ContainsUserFacingDataExpected: true
       MinDeletionSemantics: CryptographicErase
       ErasureScope: Volume
       MaxEraseLatencyHours: 24
     }

     ConcurrencyHints {
       AdvisoryLockingSupported: true
     }
   }
```

A.2. APS-Vector (Embedding-Heavy)

APS-Vector is intended for embedding-heavy and index-heavy agent workloads. It assumes larger capacity, higher IOPS, and a RecommendedIOSizeKiB of 32 KiB to reflect typical vector segment sizes during scan-oriented operations. It disables advisory locking hints because many vector index implementations use append-only or copy-on-write designs and do not rely on traditional filesystem locking semantics for concurrency control. The VectorPhase field is set to indicate that this profile is tuned primarily for read-heavy query traffic.

```
   PersistentStateLineOfService {
     ApsVersion: "1.0"
     Id: "aps-vector-1"
     Name: "APS-Vector-Index"
     Purpose: "AgentPersistentState"

     IOProfile: VectorIndex
     VectorPhase: Query

     CapacityRangeGiB {
       MinGiB: 50
       MaxGiB: 2000
       RecommendedStepGiB: 50
     }

     Performance {
       MinIOPS: 8000
       MaxLatencyMsP95: 25
       MaxLatencyMsP99: 80
       RecommendedIOSizeKiB: 32
     }

     Durability {
       DurabilityClass: VeryHigh
       ReplicationScope: MultiRegion
     }

     CrashConsistency {
       RequiresWriteOrdering: true
     }

     Retention {
       RetentionDays: 90
       ForgettingPolicySupported: true
     }

     Privacy {
       ContainsUserFacingDataExpected: true
       MinDeletionSemantics: CryptographicErase
       ErasureScope: FilesystemTree
       MaxEraseLatencyHours: 48
     }

     ConcurrencyHints {
       AdvisoryLockingSupported: false
     }
   }
```

A.3. APS-Light (BestEffort / ApplicationKey)

APS-Light illustrates deployments that rely on application-managed keys or do not require strict retention or erase guarantees. Suitable for experimentation, low-risk workloads, or sidecar-based encryption designs where the storage layer itself does not enforce cryptographic erase.

```
   PersistentStateLineOfService {
     ApsVersion: "1.0"
     Id: "aps-light-1"
     Name: "APS-Light-Experiment"
     Purpose: "AgentPersistentState"

     IOProfile: MixedRandom

     CapacityRangeGiB {
       MinGiB: 1
       MaxGiB: 200
       RecommendedStepGiB: 1
     }

     Performance {
       MinIOPS: 500
       MaxLatencyMsP95: 40
       MaxLatencyMsP99: 120
       RecommendedIOSizeKiB: 4
     }

     Durability {
       DurabilityClass: Standard
       ReplicationScope: LocalAZ
     }

     CrashConsistency {
       RequiresWriteOrdering: false
     }

     Retention {
       RetentionDays: 7
       ForgettingPolicySupported: false
     }

     Privacy {
       ContainsUserFacingDataExpected: false
       MinDeletionSemantics: BestEffort
       ErasureScope: ApplicationKey
       MaxEraseLatencyHours: 24
     }

     ConcurrencyHints {
       AdvisoryLockingSupported: false
     }
   }
```

Author’s Address

Madhava Gaikwad
Email: gaikwad.madhav@gmail.com