Reference Architecture

Everpure

Redis

FlashBlade and Redis: Two-Tier Memory for Clinical AI

HIPAA-aligned agent memory and an auditable system of record

Contents

Executive summary

Hospitals adopting clinical AI copilots encounter a memory problem that does not exist in proofs of concept or demos. A single-turn chatbot needs no memory, but a rounding copilot that must remember the previous turn while the clinician is still in the patient room, survive the handoff to the night team four hours later, and produce a defensible record of every recommendation in case the compliance office requests it 12 months later needs two kinds of memory. This is usually solved with two different stacks: a submillisecond hot tier for live working state, and a durable, governable tier that the hospital IT organization already knows how to audit against the Health Insurance Portability and Accountability Act (HIPAA) Security Rule. Bolting these together with point integrations is what stalls clinical AI projects between grand rounds and a deployment that actually touches patient care.

This reference architecture solves that problem by pairing the Redis Agent Memory Server, an open source memory service built on Redis Stack, with Everpure™ FlashBlade® as the durable system of record. Redis handles working memory and vector-backed, long-term recall on the hot path. FlashBlade persists versioned encounter snapshots and Redis Database (RDB) backups through native S3, with Network File System (NFS) on the same array as the safety net when S3 is temporarily unreachable. The two tiers are connected by a thin storage abstraction in the agent service, which writes S3 first, falls back to NFS on failure, and reconciles to S3 in the background once connectivity returns. The result is one hot tier for the platform team to operate, one durable surface for the compliance team to audit, and one consolidated piece of infrastructure for the hospital IT organization to back up.

This reference architecture is intended for clinical informatics teams, hospital IT and platform engineers, and AI engineers building clinical copilots inside healthcare delivery organizations. It provides the architecture, deployment guide, and operational patterns required to run the stack on real FlashBlade hardware behind the institution's existing HIPAA Security Rule controls.

The problem: Clinical copilots stall at the memory layer

Most clinical AI prototypes ignore memory because they do not need to account for it yet. The agent runs in a conference room, the conversation lives in the process, and nobody is depending on it after the demo ends. A production clinical copilot does not have that luxury. It has to remember the previous turn while the clinician is still in the patient room, survive the handoff to the night team four hours later, persist a defensible record of every recommendation for possible quality and safety committee review 12 months later, and never leak the patient text outside the institution's data perimeter. Two of those four requirements are about speed and three are about governance, and they are not solvable in a single store.

The natural reflex is to put everything in one place. One engineer might reach for Redis and assume durability is a configuration knob, while another might choose object storage and assume latency can be tuned the same way. Neither works in a hospital.

Redis provides a submillisecond hot path, vector search via RediSearch, and a working memory model that maps cleanly to how clinicians actually reason: per-encounter state, structured updates as labs and imaging return, and semantic recall over historical facts about the patient, the clinician's documented preferences, and the institution's standard order sets. What it does not give you, by itself, is a system of record that a HIPAA auditor will accept. RDB and append-only file (AOF) persistence are durability features for cache state, not retention features for clinical documentation. They are not what a hospital compliance officer points at when the auditor asks where the patient conversation history is stored, who has accessed it, and how long it will be retained.

Object storage gives you durable, governable, retention-capable storage. What it does not give you is a hot path. Every turn of a live clinical conversation cannot afford an S3 round trip, much less the retrieval scan a large language model (LLM) needs to ground its response in patient context. A 10ms tier in a 50ms budget is a different design than a 500ms tier in a 50ms budget. At the bedside, the difference is whether the copilot keeps up with the clinician or makes the clinician wait.

The architectural answer is to use both, with each in its natural regime. Redis carries the working state and the vector recall on the hot path. The durable tier owns the encounter as the system of record. Every session that ends, every snapshot that fires, and every audit-relevant event flows from Redis to FlashBlade as a versioned, governable object. The trick is making that handoff cost-effective enough that the application developer does not feel it and reliable enough that the hospital infrastructure team does not have to babysit it.

That is the architecture this reference implements, with FlashBlade as the durable tier and a specific operational contract for moving data between the two.

Redis Agent Memory Server: The hot tier

The Redis Agent Memory Server is an open source memory service built on top of Redis Stack. It exposes two memory primitives that map directly to how production agents actually use memory.

Working memory

Working memory is the per-encounter state the agent carries through a clinical conversation. It holds the patient medical record number (MRN), the chief complaint, the differential diagnosis the agent is building, the labs and imaging reviewed in the current session, the reasoning trace the agent surfaces when asked to explain itself, and the pending orders that have not yet been entered into the electronic health record (EHR). Every turn reads that state at the start and writes the updated version at the end.

Working memory has two requirements. It has to be fast because every turn reads and writes it, and it has to be structured because the agent needs to update specific fields (for example, add a lab result or refine the differential) without retransmitting the entire encounter. Redis hashes and JSON satisfy both. The memory server wraps them in an HTTP API that the agent service calls without owning the Redis schema directly.

The result is a hot path of a single submillisecond read at the start of a turn and a single submillisecond write at the end of it. The reference implementation reports a memory retriever latency of 0.17ms p95 over a 100-item seeded set (k=5). That is in the noise next to the 14-second LLM inference call it sits beside. At the bedside, the clinician feels the model, not the framework.

Long-term memory

Long-term memory is the semantic store of facts an agent accumulates across sessions: the clinician's documented preference for early statin therapy, the patient's documented allergies, or the institution's standard order set for community-acquired pneumonia. Long-term memory is not a transcript; it is a vector-indexed pool of structured facts that the agent retrieves at the start of each turn to ground its response.

RediSearch provides hierarchical navigable small world (HNSW) vector indexing inside Redis, which means the same store that holds working memory also holds the long-term vectors, with no second system to operate. The memory server handles fact extraction (LLM-based summarization of session content into durable facts) and the retrieval surface (top-k semantic search by clinician or topic).

Long-term memory is the answer to the rehydration question: when a clinician walks back in at 3 a.m., the durable tier returns the previous session's transcript, and the long-term store returns the semantic facts that should ground the next turn. Both are warm by the second message.

LangGraph checkpointing

When the agent is built on a graph framework (LangGraph in this reference), Redis also provides a native checkpointer for graph state. That gives every intermediate node in the graph (for example, route, retrieve, reason, write) a durable checkpoint without the application managing serialization. A crashed turn can be resumed from the last checkpoint rather than restarted from scratch.

Operationally, this means the same Redis instance handles three jobs at once: working memory for the live conversation, vector index for long-term recall, and checkpoint storage for the graph itself. That is one service, one operational footprint, and three responsibilities the agent would otherwise have to coordinate across separate stores.

Everpure FlashBlade: The durable tier

Everpure FlashBlade is a unified, fast file and object platform. In this architecture, it serves a specific role as the durable system of record for everything the hot tier produces that the business is required to keep, retrieve, or audit.

Three properties of FlashBlade matter for this role.

1. Native S3, no gateway

The Agent Memory Server's hot path stays in Redis, but the persistence path of the application writes JSON snapshots and Redis RDB backups to S3-compatible object storage. FlashBlade speaks S3 natively as a first-class protocol. There is no gateway process proxying requests, no translation layer, and no separate service to deploy, monitor, and scale. The agent's boto3 client writes directly to FlashBlade via standard S3 API calls.

This matters because the alternative is a single-node, S3-compatible gateway sitting in front of a file system, which becomes the ceiling on your persistence throughput the moment session volume scales. FlashBlade removes that layer entirely, reducing operational complexity and tail latency on every write.

2. S3 and NFS on the same array

The agent service's storage layer writes S3 first, with NFS as the fallback path. When the S3 endpoint is reachable, every session ends with a PUT to a versioned key (for example, **s3://<bucket>/clinicians/<clinician_id>/sessions/<session_id>/memory.json**). When the S3 endpoint is temporarily unavailable (during a transient network partition or authentication blip, for example), the same payload is written to the NFS mirror under the same key structure, and the key is appended to a pending ledger (**_pending.jsonl**). A background reconciler drains the ledger as soon as S3 is reachable again.

The critical property is that the NFS fallback is ***the same array***. FlashBlade exposes S3 and NFS over the same physical storage, which means the fallback path is not a second-class downgrade to a worse store; it is the same durable substrate accessed through a different protocol. A session written to NFS is just as durable as one written to S3, and the reconciler is moving data within FlashBlade rather than copying it across systems.

Without this property, "S3 fallback" implies a second storage system, typically a local disk on the agent host or a separate filer somewhere in the hospital data center. That second system has different durability, different backup posture, and a different compliance review. For a hospital IT organization already managing HIPAA Security Rule controls on its existing storage estate, adding a second store under a clinical AI workload means a second business associate agreement (BAA) scope to track, a second access audit trail to operate, and a second retention policy to keep aligned with the institution's medical record policy. Collapsing both protocols onto one array eliminates that risk and that review.

3. Ability to scale with session volume

Clinical AI workloads do not have a predictable footprint. A pilot in one department might run a dozen encounters per day. A hospital-wide rollout across rounding, ambulatory documentation, and emergency department triage might reach thousands per hour. A multi-hospital health system standardizing on one copilot stack might reach hundreds of thousands per day. Encounter snapshots, RDB backups, and long-term fact extractions accumulate. The durable tier has to absorb that growth without becoming the bottleneck.

The scale-out architecture of FlashBlade, designed for unstructured data at petabyte scale, handles that growth natively. Write throughput scales linearly with blade count, and read performance remains consistent as data volumes grow. There is no reconfiguration of the agent stack required when storage is added; the S3 endpoint and bucket stay the same.

FlashBlade sits within a broader Everpure AI infrastructure stack. For organizations also running on-premises GPU inference, the same stack offers the Pure Key-Value Accelerator (KVA) for inference KV-cache offload and Portworx® for Kubernetes-native persistent volumes. This reference architecture focuses on FlashBlade as the durable memory tier; in practice, the same array often supports the model serving and observability pipelines alongside it.

Architecture overview

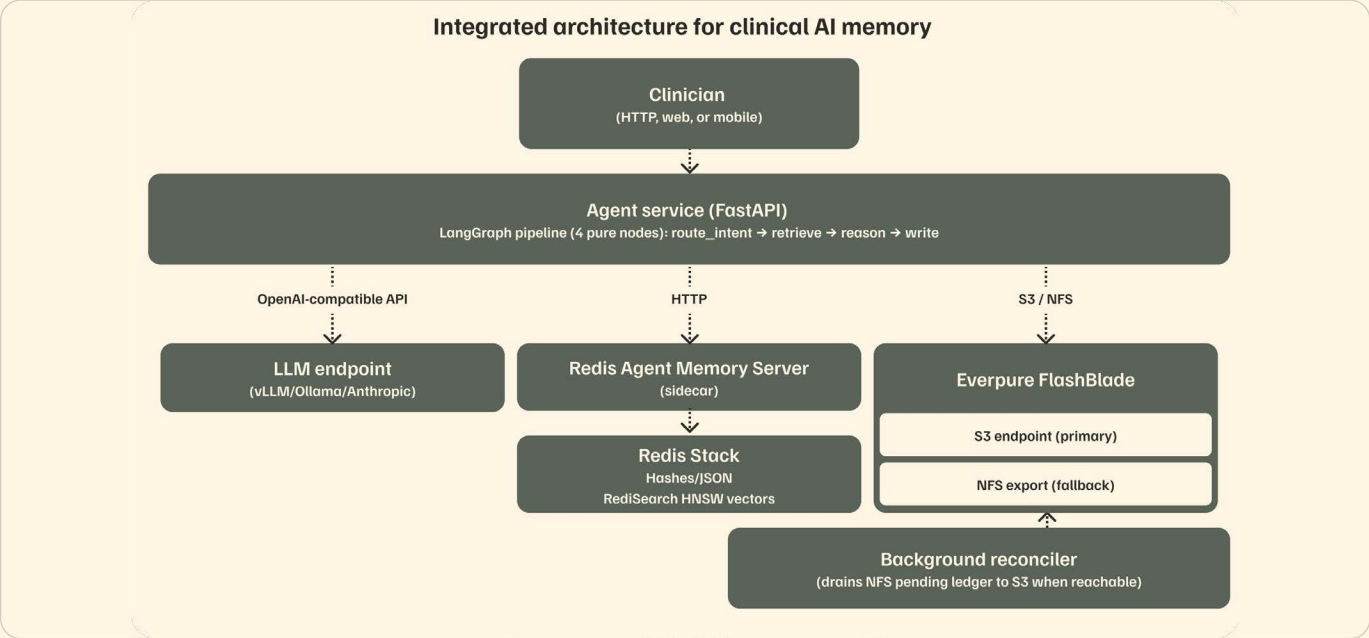
The integrated architecture consists of the following components:

- **Application layer**: any clinical AI workload that needs durable encounter state, including rounding copilots, ambulatory documentation assistants, emergency department triage and disposition copilots, ICU sign-out summarization agents, and discharge planning copilots. The application is built against a small **MemoryClient** interface for working memory and a **FlashBladeMemoryStore** for durable persistence. The interfaces hide the storage transport, so the application code never branches on "S3 up" vs. "S3 down."
- **Agent service**: a FastAPI service that exposes the conversation surface (create session, send message, end session, list sessions, resume session). The agent itself is built on LangGraph as a four-node pipeline (route intent, retrieve memory, reason, write memory). Each node is a pure function over an immutable state object, which keeps the graph testable in isolation from the model and the stores.
- **Redis Agent Memory Server**: a sidecar process that owns working memory create, read, update, delete (CRUD) and long-term semantic memory over a Redis Stack instance. The agent service calls it over HTTP. Working memory is a per-session structured state; long-term memory is RediSearch-indexed vectors over per-clinician facts.
- **Redis Stack**: the backing store for both working memory (hashes/JSON) and long-term memory (HNSW vector index via RediSearch). It also serves as the LangGraph checkpoint store when the optional checkpointer is enabled.
- **LLM endpoint**: any OpenAI-compatible inference endpoint. For HIPAA-shaped, no-egress deployments, a self-hosted endpoint on local GPU hardware keeps patient text inside the institution's network; vLLM is the preferred choice for scalable inference, with Ollama suited to local development and pilot work. The agent is agnostic to which model is behind the endpoint.
- **FlashBlade S3**: the S3-native endpoint that receives versioned session snapshots, long-term fact exports, and Redis RDB backups. It is written through **boto3** from the agent service.
- **FlashBlade NFS**: the same physical storage exposed via NFS, mounted into the agent service as the fallback write path when S3 is temporarily unreachable. The reconciler drains the pending ledger to S3 when connectivity returns.

A representative request-to-persistence path flows as follows:

> User sends a message → Agent service invokes graph → Graph reads working memory and retrieves long-term context from Redis via memory server → LLM call → Graph writes updated working memory back to Redis via memory server → End of session triggers serialization of working memory and message history into a versioned JSON schema → Durable store writes payload S3 first and falls back to NFS on failure → When session resumes (for example, for a shift change or follow-up encounter), resume call reads payload back from FlashBlade and rehydrates Redis

The following diagram shows the integrated architecture, with the hot path (Redis Agent Memory Server) and the durable path (FlashBlade S3 primary, NFS fallback) connected through the agent service.



The hot path stays in Redis at submillisecond latency. The durable path writes to FlashBlade S3 as the system of record for the encounter, with NFS on the same array as the fallback when S3 is briefly unreachable. The agent service is the only egress surface, which keeps the protected health information (PHI) perimeter narrow and auditable against the HIPAA Security Rule. A background reconciler replays pending writes to S3 once connectivity is restored.

The same architecture also handles the shutdown case. The FastAPI lifespan handler runs a graceful flush on SIGTERM: every active (not-yet-ended) session is serialized through the durable store, and the Redis RDB is copied through the same path as a point-in-time snapshot. The session-flush half runs in every deployment; the RDB-copy half requires Redis's RDB directory to be mounted into the agent service container (typically a shared NFS path on FlashBlade).

Deployment guide: Connecting Redis and FlashBlade

Prerequisites

FlashBlade configuration

- A FlashBlade array with an S3 endpoint accessible from your application network (for example, **https://flashblade.example.internal:443**)
- An S3 bucket created on FlashBlade (for example, **agent-memory-prod**)
- An access key and secret key with read/write permissions on the bucket
- An NFS export from the same FlashBlade, mountable from every agent service host or pod (for example, **flashblade:/exports/agent-memory**)

Agent service host

- A container runtime: Docker 24+ with the Compose plug-in for single-host deployments or a Kubernetes 1.24+ cluster for production and enterprise deployments
- For Kubernetes: Helm 3.x to deploy the stack and a Container Storage Interface (CSI) driver for FlashBlade-backed persistent volumes (Portworx or the Everpure CSI driver for direct NFS/S3 access); the NFS export is mounted into pods as a PersistentVolume, and the S3 endpoint is reached over the cluster network
- Network connectivity from every agent service host or pod to both the FlashBlade S3 endpoint and the NFS export
- Python 3.11+ if running the agent service as a local process rather than a container

Redis hot tier

- A Redis Stack instance (v7.4+ recommended) with RediSearch enabled, reachable from both the agent service and the Redis Agent Memory Server. For enterprise fleets, Redis Cluster or Redis Enterprise provides horizontal scale and high availability across nodes.
- The Redis Agent Memory Server (`ghcr.io/redis/agent-memory-server:latest`) deployed as a sidecar (Docker) or colocated container (Kubernetes pod) with `REDIS_URL` pointing at the Redis Stack instance.
- For graceful-shutdown RDB snapshots: Redis's `dir` configured to a path readable from both the Redis container and the agent service container (typically the FlashBlade NFS mount, exposed as a shared PersistentVolume in Kubernetes).

LLM endpoint

- An OpenAI-compatible inference endpoint reachable from the agent service (for example, vLLM on on-premises GPU hardware for scalable HIPAA-shaped deployments, Anthropic Claude for cloud deployments, or Ollama for local development). For enterprise inference, vLLM behind a load balancer across multiple GPU nodes provides the throughput and availability a production fleet requires.

Configuring FlashBlade as the durable tier

The agent service reads its storage configuration from environment variables. To point it at a production FlashBlade, set the following:

```
FB_S3_ENDPOINT=https://flashblade.example.internal:443 FB_S3_BUCKET=clinician-copilot-prod FB_S3_ACCESS_KEY=<access-key>
FB_S3_SECRET_KEY=<secret-key> FB_NFS_PATH=/mnt/flashblade-nfs
```

Mount the FlashBlade NFS export inside the agent service container at the path specified by `FB_NFS_PATH`:

```
volumes:  - type: bind   source: /mnt/flashblade-nfs   target: /mnt/flashblade-nfs
```

No code change is required. The durable memory store picks up the new endpoint at boot, writes new sessions to S3, and falls through to the NFS mount if the S3 endpoint is unreachable.

Configuring the Redis hot tier

The Redis Agent Memory Server runs as a sidecar, with one environment variable pointing at Redis Stack:

```
redis: image: redis/redis-stack:7.4.0-v1 ports: - "6379:6379" command: ["redis-server", "--dir", "/mnt/flashblade-nfs/redis", "--dbfilename", "dump.rdb"] volumes: - type: bind source: /mnt/flashblade-nfs target: /mnt/flashblade-nfs memory-server: image: ghcr.io/redis/agent-memory-server:latest environment: REDIS_URL: redis://redis:6379/0 ports: - "8088:8088"
```

Two things to note: first, the Redis **--dir** is pointed at the FlashBlade NFS mount. This places the RDB snapshot on durable storage by default, independent of the agent's lifespan-flush path. Second, the same NFS mount is shared with the agent service, which is what allows the lifespan handler to copy the RDB through **FlashBladeMemoryStore** on graceful shutdown.

The agent service then points at the memory server and at Redis directly (for the optional LangGraph checkpointer):

```
REDIS_URL=redis://redis:6379/0 MEMORY_SERVER_URL=http://memory-server:8088
```

Configuring the agent service

The agent service is a single FastAPI process with the following environment variables:

```
# Hot tier REDIS_URL=redis://redis:6379/0 MEMORY_SERVER_URL=http://memory-server:8088 # Durable tier (FlashBlade) FB_S3_ENDPOINT=https://flashblade.example.internal:443 FB_S3_BUCKET=clinician-copilot-prod FB_S3_ACCESS_KEY=<access-key> FB_S3_SECRET_KEY=<secret-key> FB_NFS_PATH=/mnt/flashblade-nfs # LLM endpoint (one of) LLM_MODEL=gpt-oss:20b OPENAI_BASE_URL=https://inference.example.internal:8000/v1 OPENAI_API_KEY=<service-token> # Operational tuning RECONCILE_INTERVAL_S=30 # how often to drain pending ledger SHUTDOWN_PERSIST_TIMEOUT_S=10 # per-session SIGTERM flush deadline REDIS_BGSAVE_TIMEOUT_S=30 # max wait for BGSAVE on shutdown
```

Verifying the integration

After deployment, verify each layer of the stack.

1. Confirm that the services are running:

```
docker compose ps # expects: redis, memory-server, agent all healthy curl -fsS http://<AGENT_URL>:8080/healthz # expects: {"ok": true}
```

2. Verify FlashBlade S3 connectivity from the agent service:

```
docker compose exec agent python -c "import boto3, os; s3 = boto3.client('s3', endpoint_url=os.environ['FB_S3_ENDPOINT'], aws_access_key_id=os.environ['FB_S3_ACCESS_KEY'], aws_secret_access_key=os.environ['FB_S3_SECRET_KEY']); print(s3.list_buckets())"
```

3. Verify that the NFS fallback path is mounted:

```
docker compose exec agent ls -la /mnt/flashblade-nfs # expects: a writable directory on FlashBlade NFS
```

4. Run an end-to-end session and confirm it persists to S3:

```
# Create a session and post one turn SID=$(curl -fsS -X POST http://<AGENT_URL>:8080/sessions \ -H 'content-type: application/json' \ -d '{"clinician_id":"C-DEMO","patient_id":"P-1234"}' \ | python3 -c 'import json,sys; print(json.load(sys.stdin)["session_id"])') curl -fsS -X POST "http://<AGENT_URL>:8080/sessions/$SID/messages" \ -H 'content-type: application/json' -d '{"text":"initial rounding note"}' # End the session: expects {"backend":"s3", ...} curl -fsS -X POST "http://<AGENT_URL>:8080/sessions/$SID/end" \ -H 'content-type: application/json'
```

5. Confirm the durable object on FlashBlade:

```
aws --endpoint-url=$FB_S3_ENDPOINT s3 ls \ s3://$FB_S3_BUCKET/clinicians/C-DEMO/sessions/$SID/
```

6. Exercise the graceful-shutdown path:

```
# Start a session, do not end it, then SIGTERM the agent container docker compose stop agent docker compose logs agent --tail=20 | grep shutdown_session_persisted # expects: event=shutdown_session_persisted sid=... backend=s3 key=... docker compose start agent curl -fsS "http://<AGENT_URL>:8080/sessions?clinician_id=C-DEMO" # expects: the in-flight session reappears with ended_at populated | | :---- |
```

Production hardening

S3-to-NFS fallback and reconciliation. The `FlashBladeMemoryStore` retries S3 three times with exponential backoff (0.2s, 0.6s, and 1.8s) before falling through to the NFS path. Whenever a fallback occurs, the store emits a structured log entry (`event=fb_fallback`) and appends the failed key to `_pending.jsonl` on the NFS mount. The background reconciler reads that ledger on every interval (`RECONCILE_INTERVAL_S`, default 30s) and replays pending writes to S3 when connectivity returns. Operationally, this means a transient FlashBlade S3 outage drains itself without operator intervention as long as the NFS mount is healthy.

Schema versioning. Every persisted payload includes a `schema_version` field. The store rejects unknown versions with a `SchemaVersionError` at read time rather than silently returning a malformed state. Production deployments should keep at least one read-only deserializer per legacy schema version and migrate forward through write-new-on-read.

Retention and life cycle. FlashBlade S3 supports standard S3 lifecycle policies. Session snapshots, long-term facts, and RDB backups are written to distinct prefixes (`clinicians/<id>/sessions/`, `clinicians/<id>/longterm/`, `snapshots/redis/`) so retention can be tuned per category. Healthcare deployments typically retain session JSON for the duration required by the institution's medical record policy and RDB snapshots for a shorter operational window (for example, 30 days).

Secrets management. The previous examples use inline credentials for clarity. In production, manage FlashBlade access keys through Kubernetes Secrets or a secrets manager (for example, HashiCorp Vault or AWS Secrets Manager) and reference them via `envFrom` in the deployment manifest.

Scaling considerations. The agent service is stateless and scales horizontally behind any standard load balancer; all session state lives in Redis or FlashBlade. Redis scales vertically up to the working-set size of the deployment and horizontally via Redis Cluster for very large fleets. FlashBlade scales linearly with blade count and requires no reconfiguration of the agent service when storage is added.

Network posture. For HIPAA-shaped deployments, the most important property is that no patient text leaves the host running inference. The architecture supports this directly: the LLM endpoint can be a vLLM service running on on-premises GPU hardware, the Redis and FlashBlade tiers are both on the institution's network, and the agent service is the only egress surface. Egress can then be restricted to telemetry and operational endpoints rather than including the patient conversation surface.

Business value

The architecture delivers value across five dimensions that healthcare delivery organizations care about for different reasons.

1. Continuity of care across shifts and encounters

The most clinically meaningful property of the architecture is that the encounter survives the shift. A clinician who starts an admission at 7 p.m. and hands off to the night team at 11 p.m. leaves a durable, structured record of the conversation on FlashBlade; the incoming team's copilot rehydrates that record into Redis and continues from the same context. The same property holds for a patient who is readmitted, an outpatient visit that follows up on a prior encounter, and a case that crosses service lines. Continuity of context is what turns a per-shift novelty into a system the clinical team actually depends on.

2. HIPAA defensibility on one durable surface

Every closed encounter, extracted fact, and Redis RDB snapshot lives on FlashBlade, with the same governance posture as the institution's other PHI. There is one storage system to back up, one storage system to audit, one access log to monitor, and one retention policy to enforce against the institution's medical record schedule. The alternative (Redis with AOF enabled or Redis plus a developer-managed object store) is not a defensible answer to a HIPAA auditor asking where patient conversation history is stored, who can access it, and how long it will be retained. FlashBlade with documented retention policies is.

3. No-egress inference as a procurement accelerator

The architecture supports an LLM endpoint running entirely inside the institution's network: a vLLM service on hospital-owned GPU hardware or a local Ollama instance for development and pilot work. When inference is local, no patient text leaves the host running the model, which removes an entire category of compliance conversations: there is no cloud LLM vendor BAA to negotiate for the inference step, no inference egress to review with the privacy office, and no token-billing surface that changes with conversation volume. For institutions whose procurement timeline is dominated by BAA negotiation rather than infrastructure decisions, this is often the single largest acceleration the architecture provides.

4. Auditable, AI-assisted clinical decision trail

Every session is persisted as a versioned JSON object containing the messages, structured working memory, retrieved long-term facts, and timing of each step. That makes the agent's reasoning trail reviewable by the quality and safety committee months after the encounter and traceable when a case is selected for retrospective review. As U.S. Food and Drug Administration (FDA) and U.S. Department of Health and Human Services (HHS) guidance on clinical AI matures, the institutions positioned to comply are the ones that already have the AI decision trail captured in a structured format on durable, governable storage. This architecture captures it as a side effect of how the agent runs, not as an after-the-fact instrumentation pass.

5. Latency budget that survives the bedside

The hot path of the agent (working memory read, long-term retrieval, working memory write) stays in Redis at submillisecond latency. The benchmark numbers in the reference implementation report a memory retriever p95 of 0.17ms and a full graph turn (without the LLM call) of 0.0072ms p95. The framework adds essentially nothing to the LLM cost; the latency the clinician feels is the model, which is also the only place latency work pays off. This is the property that lets a multi-turn rounding conversation feel like a conversation rather than a slideshow.

Other clinical workloads

The rounding copilot is one instantiation of the architecture. The same two-tier memory pattern applies to several other clinical AI workloads commonly built inside healthcare delivery organizations.

Ambulatory documentation copilots that listen during a visit build the structured note in working memory as the encounter progresses and persist the closed note to FlashBlade as the system of record for downstream coding, billing, and clinical review. The hot tier holds the in-progress note and the patient's relevant history, and the durable tier persists every closed encounter for the medical record retention window.

Emergency department triage and disposition copilots that build a structured triage assessment over multiple turns with the patient and the nurse retrieve the institution's protocol library and the patient's prior visit history from long-term memory. They then persist the disposition decision and its supporting evidence to FlashBlade for chart review and quality and safety oversight.

ICU sign-out and handoff summarization agents that maintain a structured per-patient summary across the shift retrieve longitudinal context (for example, ventilator trends, vasopressor decisions, and neurologic exam history) and persist the handoff summary at shift change. The next team's copilot rehydrates the same structured summary, which is exactly the cross-shift continuity property the rounding copilot uses, applied to a higher-acuity setting.

Discharge planning copilots that track the discharge readiness criteria across a multi-day admission in working memory retrieve the patient's outpatient resources and prior discharge plans from long-term memory and persist the plan to FlashBlade so the case manager who picks it up the next morning sees the same context the prior shift built.

In all four cases, the architecture is identical. Only the schema of working memory (for example, note vs. triage vs. handoff vs. discharge plan) and the categories of long-term facts change.

Conclusion

Clinical AI copilots are not single-store workloads. They need a hot path that operates at the speed of the bedside conversation and a durable path that operates at the cadence of compliance and clinical review. Redis and FlashBlade fit those two roles natively, and combining them removes the architectural decision that most often stalls clinical AI projects between grand rounds and a deployment that touches patient care.

The Redis Agent Memory Server provides:

- **Working memory** at submillisecond latency for live clinical turns
- **Long-term semantic memory** via RediSearch HNSW for cross-encounter recall (patient history, clinician preferences, and institutional order sets)
- **Checkpoint storage** for the agent graph when built on LangGraph

Everpure FlashBlade provides:

- **A native S3 endpoint** as the system of record for encounter snapshots, long-term facts, and Redis RDB backups
- **NFS on the same array** as a durability-equivalent fallback when S3 is temporarily unreachable, with one BAA scope rather than two
- **Scale-out capacity** that grows with encounter volume across a multi-hospital health system without reconfiguration of the agent stack

The clinician copilot reference implementation demonstrates the full architecture end to end against a HIPAA-shaped workload, with documented benchmark numbers, integration tests covering each failure mode, and a verified graceful-shutdown path that flushes in-flight encounters before the process exits. Source, deployment, and reproduction details are in the project repository.

For institutions evaluating this architecture, the deployment guide in this reference architecture is sufficient to stand up against a production FlashBlade. Point the agent service at the institution's S3 endpoint and NFS export, configure the Redis Agent Memory Server, and the same code that runs the demo runs the production deployment. No application changes are required between the two.

Build the copilot the way your hospital builds any other clinical system: on infrastructure your IT organization already trusts, against a system of record your compliance team already audits, with the patient data perimeter you already enforce. Memory is not the exception.

Learn How Everpure Drives Healthcare Success

ID50997-01-en-07/26