Blob Storage System

System Description

Abstract
This document describes the Sensor Data Transfer System, which provides sensor measurement collection, storage, and retrieval capabilities within the Arrowhead Framework. The system enables decoupled data exchange between sensor producers and data consumers through the transferData service interface, supporting industrial IoT automation scenarios with measurement lifecycle management.

Contents

1 Overview **3**
1.1 Significant Prior Art . . . 4
1.2 How This System Is Meant to Be Used . . . 4
1.3 System Functionalities and Properties . . . 5
1.4 Important Delimitations . . . 7

2 Services **8**
2.1 Produced Services . . . 8
2.2 Consumed Services . . . 8

3 Security **10**
3.1 Operational Modes . . . 10
3.2 Certificate Handling . . . 10
3.3 Security Model . . . 10
3.4 Arrowhead Certificate Profile . . . 11

4 Revision History **12**
4.1 Amendments . . . 12
4.2 Quality Assurance . . . 12

1 Overview

This document describes the Blob Storage System, which provides persistent storage for binary large objects within Arrowhead local clouds. The system acts as a storage provider enabling systems to store binary data with associated metadata and retrieve stored content via unique identifiers or streaming interfaces. This was done to ease development of AI models.

The system exposes the blobStorage service, enabling producer systems to create blobs with binary content and metadata, consumer systems to read complete blobs or stream large object, update existing blob content, and delete blobs when no longer needed. This architecture supports file storage, image repositories, ai models in form of dill, pickle or safe tensors, and other binary data management scenarios .

The rest of this document is organized as follows. In Section 1.1, we reference major prior art capabilities of the system. In Section 1.2, we describe the intended usage of the system. In Section 1.3, we describe fundamental properties provided by the system. In Section 1.4, we describe delimitations of capabilities of the system. In Section 2, we describe the abstract service functions consumed or produced by the system. In Section 3, we describe the security capabilities of the system.

1.1 Significant Prior Art

The Blob Storage System builds upon established object storage and content management patterns:

Amazon S3 API: The Create/Read/Update/Delete operation model follows S3-style object storage semantics with identifier-based access and metadata attachment.

HTTP Range Requests: The streamRead operation implements chunked content delivery enabling efficient transfer of large blobs without full in-memory buffering.

Distributed file systems: Storage backend patterns reflect distributed filesystem capabilities including replication, erasure coding, and fault tolerance mechanisms.

Arrowhead Framework service patterns: Integration with Service Registry, Authorization, and Orchestration follows standard Arrowhead service provider architecture for industrial systems.

1.2 How This System Is Meant to Be Used

The Blob Storage System operates in industrial scenarios requiring persistent binary data storage with networked access:

1.2.1 Deployment Architecture

The system deploys as a storage service provider within an Arrowhead local cloud. It registers the blobStorage service with the Service Registry, making storage capabilities discoverable to authorized systems. The system requires persistent storage infrastructure capable of handling large binary objects.

Typical deployment includes:

- One or more instances with shared storage backend for scalability
- Storage backend: filesystem, object storage (S3-compatible), or blob database
- Network access to Arrowhead core systems (Service Registry, Authorization, Orchestration)
- Certificate infrastructure for secure communication
- Sufficient storage capacity for blob retention requirements

1.2.2 Producer Interaction Pattern

Systems storing data invoke the Create operation:

1. Producer system discovers blobStorage service via Service Registry
2. Authorization system validates producer credentials
3. Producer invokes Create with binary content and metadata
4. System validates content, assigns unique identifier, persists blob
5. System returns blob identifier and storage confirmation
6. Producer records identifier for future reference or sharing with consumers

For existing blobs requiring modification, producers invoke Update with blob identifier and new content.

1.2.3 Consumer Interaction Pattern

Systems retrieving stored data use Read or streamRead:

Complete blob retrieval:

1. Consumer discovers blobStorage service
2. Consumer obtains blob identifier through external coordination
3. Consumer invokes Read with identifier
4. System retrieves complete blob content and metadata
5. Consumer processes binary data

Streaming large blobs:

1. Consumer invokes streamRead with identifier
2. System returns initial chunk with continuation metadata
3. Consumer processes chunk
4. Consumer invokes streamRead with continuation token
5. Process repeats until blob fully transferred

Streaming enables memory-efficient handling of multi-gigabyte objects without requiring full content buffering.

1.2.4 Administrative Operations

System administrators manage storage lifecycle:

- Delete removes blobs for retention policy compliance or space reclamation
- instruction provides operational commands (implementation-specific capabilities)

1.2.5 Integration Scenarios

Machine vision systems: Camera systems store captured images. Defect detection algorithms retrieve images for analysis. Quality control systems access historical images for trend analysis.

Configuration management: Automation controllers store configuration backups. Recovery systems retrieve configurations after hardware failures.

Log aggregation: Industrial systems store diagnostic logs. Maintenance systems retrieve logs for troubleshooting.

Firmware distribution: Update servers store firmware images. Edge devices retrieve firmware for over-the-air updates using streaming for large files.

Document archives: HMI systems store operator manuals and maintenance procedures. Worker terminals retrieve documents on demand.

1.3 System Functionalities and Properties

1.3.1 Functional Properties of the System

Blob creation: The system accepts binary content with optional metadata, validates structure, assigns unique identifier, and persists to storage backend. Content type, size limits, and validation rules are implementation-configurable.

Blob retrieval: The system retrieves complete blob content and metadata by identifier. Retrieval fails if identifier invalid or requester lacks authorization.

Streaming retrieval: The system delivers large blobs in chunks, maintaining server-side cursor state between invocations. Chunk size configurable based on network and memory constraints.

Blob updates: The system replaces existing blob content while preserving identifier and selected metadata. Update semantics may follow full replacement or partial modification based on implementation.

Blob deletion: The system removes blob content and metadata, freeing storage resources. Deletion may be immediate or deferred based on retention policies.

Custom instructions: The system processes implementation-specific administrative commands through the instruction operation.

1.3.2 Configuration of System Properties

Storage backend: Configuration of filesystem paths, S3 endpoints, database connections, authentication credentials, and replication settings.

Blob size limits: Maximum blob size enforced during Create and Update operations. Typical limits range from megabytes to gigabytes based on storage infrastructure.

Streaming chunk size: Configurable chunk size for streamRead operations, balancing network efficiency against memory consumption. Typical values 64KB to 10MB.

Retention policies: Automatic deletion rules based on blob age, access patterns, or storage quota. Implementation-specific policy engines.

Identifier generation: Choice of identifier strategy: sequential integers, UUIDs, content hashes, or time-based identifiers.

Metadata validation: Configurable schema enforcement for blob metadata fields, including required keys, value types, and size constraints.

Service registration: Configuration of service metadata for Service Registry including protocol bindings, endpoint URLs, and capability advertisements.

1.3.3 Data Stored by the System

Blob content: Binary data of arbitrary type and size up to configured limits. Content stored in storage backend with integrity protection.

Blob metadata: Key-value pairs associated with each blob including content type, creation timestamp, owner system identifier, custom application metadata, and access control attributes.

Blob identifiers: Unique identifiers for each stored blob, persisted with bidirectional mapping to storage backend locations.

Streaming cursors: Temporary state tracking partial blob delivery for active streamRead sessions. Cursor expiration policies prevent resource leaks.

Audit logs: Operation history including creation, access, modification, and deletion events with timestamps and requesting system identifiers.

Storage requirements depend on blob size distribution, creation frequency, and retention duration. Systems storing high-resolution images or video require substantial capacity.

1.3.4 Non-Functional Properties

Security: Certificate-based authentication for all operations. Authorization checks validate Create/Update/Delete permissions and Read access rights. Transport-layer encryption protects blob content during transfer.

Latency:

- Create/Update: Milliseconds to seconds depending on blob size and storage backend write performance
- Read: Milliseconds for small blobs, seconds for large objects, dominated by network transfer time
- streamRead: Milliseconds per chunk, enabling constant-memory large blob retrieval
- Delete: Milliseconds for metadata removal, background garbage collection for storage reclamation

Throughput:

- Concurrent operations: Hundreds of simultaneous Read operations supported
- Write throughput: Limited by storage backend write bandwidth and concurrency capabilities
- Network saturation: Streaming capable of saturating gigabit links for large blob transfers

Availability: High-availability deployment via multiple instances sharing storage backend. Storage backend replication provides resilience against hardware failures. Load balancing distributes requests across instances.

Data integrity:

- Content checksums validate blob integrity after storage and before retrieval
- Atomic write operations ensure complete blob storage or rollback
- Replication maintains multiple copies for durability

Scalability:

- Horizontal scaling: Add instances to increase concurrent request capacity
- Storage scaling: Backend capacity expands independently of service instances
- Identifier space: UUID-based identifiers support virtually unlimited blob count

1.3.5 Stateful or Stateless

The Blob Storage System is hybrid stateful/stateless:

Persistent state: Blob content and metadata persist indefinitely subject to retention policies. Storage survives system restarts and failures.

Session state: streamRead operations maintain temporary cursor state for partial transfer continuity. Cursor state expires after inactivity timeout.

Stateless operations: Create, Read, Update, Delete are stateless—each invocation is independent without session requirements beyond authentication.

State recovery: After restart, all stored blobs remain accessible. Active streaming sessions may require restart with fresh streamRead invocation.

1.4 Important Delimitations

Not a filesystem: The system provides flat blob namespace without hierarchical directory structures. Organization via metadata or naming conventions is application responsibility.

No versioning: Update operations replace blob content completely. Historical versions are not maintained unless implemented externally.

No access control lists: Authorization operates at operation level (Create/Read/Update/Delete), not per-blob granularity. Fine-grained access control requires external policy enforcement.

No search or query: Read and streamRead require exact blob identifiers. The system provides no search by metadata, content type, creation time, or other attributes. Discovery requires external indexing.

No content transformation: The system stores and retrieves binary data without transcoding, compression, or format conversion. Applications handle data transformation.

Implementation-defined limits: Maximum blob size, concurrent streaming sessions, retention duration, and metadata constraints vary by implementation.

No distributed transactions: Create/Update/Delete operations are atomic per-blob but do not support multi-blob transactions. Applications requiring atomicity across multiple blobs implement external coordination.

Identifier coordination external: The system generates blob identifiers but provides no discovery mechanism. Applications coordinate identifier sharing through external channels (databases, message queues, orchestration metadata).

No deduplication guarantee: Identical content submitted multiple times creates separate blobs unless implementation chooses content-addressable storage.

2 Services

2.1 Produced Services

The Blob Storage System produces one service:

2.1.1 blobStorage Service

Service Definition: See blobStorage Service Description (SD) document for complete interface specification.
Service ID: blob-storage
Interface operations:

- Create: Stores new blob with binary content and metadata
- Read: Retrieves complete blob content and metadata by identifier
- Update: Replaces existing blob content
- Delete: Removes blob from storage
- streamRead: Retrieves blob content in chunks for memory-efficient large object transfer
- instruction: Executes implementation-specific administrative commands

Protocol implementations: Concrete implementations require Interface Design Description (IDD) documents specifying protocol bindings (HTTP/JSON with multipart upload, CoAP with blockwise transfer, gRPC streaming, etc.), content encoding, and error handling.

2.2 Consumed Services

The Sensor Data Transfer System consumes Arrowhead core services:

2.2.1 Service Registry

Purpose: Service registration and discovery.
Operations used:

- register: System registers transferData service during startup
- unregister: System removes registration during shutdown
- query: System discovers other required services

Dependency criticality: Mandatory. Without Service Registry, the system cannot advertise availability to potential consumers.

2.2.2 Authorization System

Purpose: Access control and credential validation.
Operations used:

- authorize: System validates incoming service invocations against authorization rules
- checkAuthorizationIntraCloud: System verifies requestor credentials

Dependency criticality: Mandatory for secure deployments. Optional for development environments with relaxed security.

2.2.3 Orchestration System (optional)

Purpose: Dynamic service binding and routing.

Operations used:

- requestOrchestration: Consumer systems use Orchestration to obtain transferData service endpoints

Dependency criticality: Recommended. Enables dynamic consumer-provider binding. Static configuration is alternative approach.

3 Security

The Sensor Data Transfer System supports both secure and insecure operational modes for development flexibility and production hardening.

3.1 Operational Modes

Insecure mode: The system accepts plaintext connections without authentication. Suitable only for isolated development environments. No certificate validation occurs. Authorization checks are bypassed.

Secure mode: The system enforces certificate-based authentication and transport encryption. All service invocations require valid X.509 certificates. Authorization rules control access to operations.

Mode selection occurs via configuration at system startup. Production deployments must use secure mode.

3.2 Certificate Handling

The system accepts both Arrowhead-compliant and non-compliant X.509 certificates depending on configuration:

Arrowhead-compliant certificates: Certificates follow Arrowhead Framework certificate profile with system name encoding in Common Name field and cloud identifier in Organization field. The system validates certificate structure and extracts identity information for authorization decisions.

Non-compliant certificates: Generic X.509 certificates with standard fields. The system accepts these certificates but extracts system identity through alternative mechanisms (Subject Alternative Name fields, custom extensions, or external identity mapping).

Certificate validation includes:

- Trust chain verification against configured Certificate Authority
- Expiration checking
- Revocation status validation if CRL or OCSP configured
- Certificate purpose verification (client authentication for consumers, server authentication for system)

3.3 Security Model

3.3.1 Protocol Security

Transport encryption: TLS 1.2 or higher for all network communication in secure mode. Cipher suite selection follows industry best practices excluding weak ciphers (RC4, DES, export-grade).

Protocol bindings: HTTPS for HTTP-based interfaces, DTLS for CoAP, TLS for MQTT. Plaintext protocols (HTTP, CoAP without DTLS) available only in insecure mode.

3.3.2 Data Protection

In-transit: Measurement data encrypted via TLS during transmission. No plaintext sensor values traverse network in secure mode.

At-rest: Database-level encryption optional based on implementation choices. The system does not perform application-layer encryption of stored measurements—database security mechanisms provide at-rest protection.

Metadata privacy: Sensor identifiers, timestamps, and metadata receive same protection as measurement values.

3.3.3 System Authentication

Mutual TLS: Both client and server present certificates. The system validates client certificates for producer authentication. Clients validate system certificate to prevent impersonation.

Identity extraction: System identity derived from certificate Common Name (Arrowhead profile) or configurable field mapping (non-compliant certificates).

3.3.4 Service Authorization

Producer authorization: Transfer operation checks whether requesting system has permission to submit measurements. Authorization rules may restrict specific sensors to specific measurement types.

Consumer authorization: GetTransfer operation validates whether requesting system can retrieve transfer records. Rules may limit consumers to subsets of measurements based on sensor identity or measurement type.

Administrative authorization: DeleteTransfer requires elevated privileges. Only authorized administrative systems or measurement owners can remove records.

Authorization rule storage: Rules maintained by Arrowhead Authorization System. The Sensor Data Transfer System queries authorization status for each operation invocation.

3.3.5 Audit and Logging

Operation logging: All service invocations logged with timestamp, requesting system identity, operation type, and success status.

Failure logging: Authentication failures, authorization denials, and validation errors logged with sufficient detail for security incident investigation.

Sensitive data: Measurement values excluded from logs to prevent information leakage. Identifiers and metadata included for correlation.

3.4 Arrowhead Certificate Profile

For complete certificate structure and field requirements, consult the Arrowhead Framework documentation at github.com/eclipse-arrowhead/documentation.

Key profile requirements:

- Common Name format: system-name.cloud-name.operator-name.arrowhead.eu
- Organization field: Cloud identifier
- Organizational Unit: System role classification
- Extended Key Usage: Client authentication for consumer systems, server authentication for provider systems

4 Revision History

4.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2025-10-22	4.4.1	Initial draft	Rasmus Tengstedt

4.2 Quality Assurance

No.	Date	Version	Approved by
1	-	4.4.1	Pending