# Asset Graph - Read Access Pattern and Workloads

The Asset Graph data model utilizes an Entity Component System. This schema-backed system ensures that any data written into the Asset Graph is validated against the schema. The schemas are integral to the system's functionality.

One of the core principles of this design is the need for scalability and flexibility, which a graph representation provides. Most design and manufacturing domain models can be mapped to a graph, as graphs inherently model various relationships among entities as first-class objects. This aligns well with models that evolve over time, offering maximum flexibility for data categorized under multiple taxonomies, unlike the rigid structures of relational or simple key-value databases. Additionally, graphs support relationship-based dynamic traversals of nodes and edges, known as Closures, which effectively represent complex structures like mechanical assemblies or building systems over their lifespan.

The Asset Graph is utilized across various industries, including Manufacturing, AEC (Architecture, Engineering, and Construction), and Media & Entertainment, to model their industry-specific data into core Asset Graph entities.

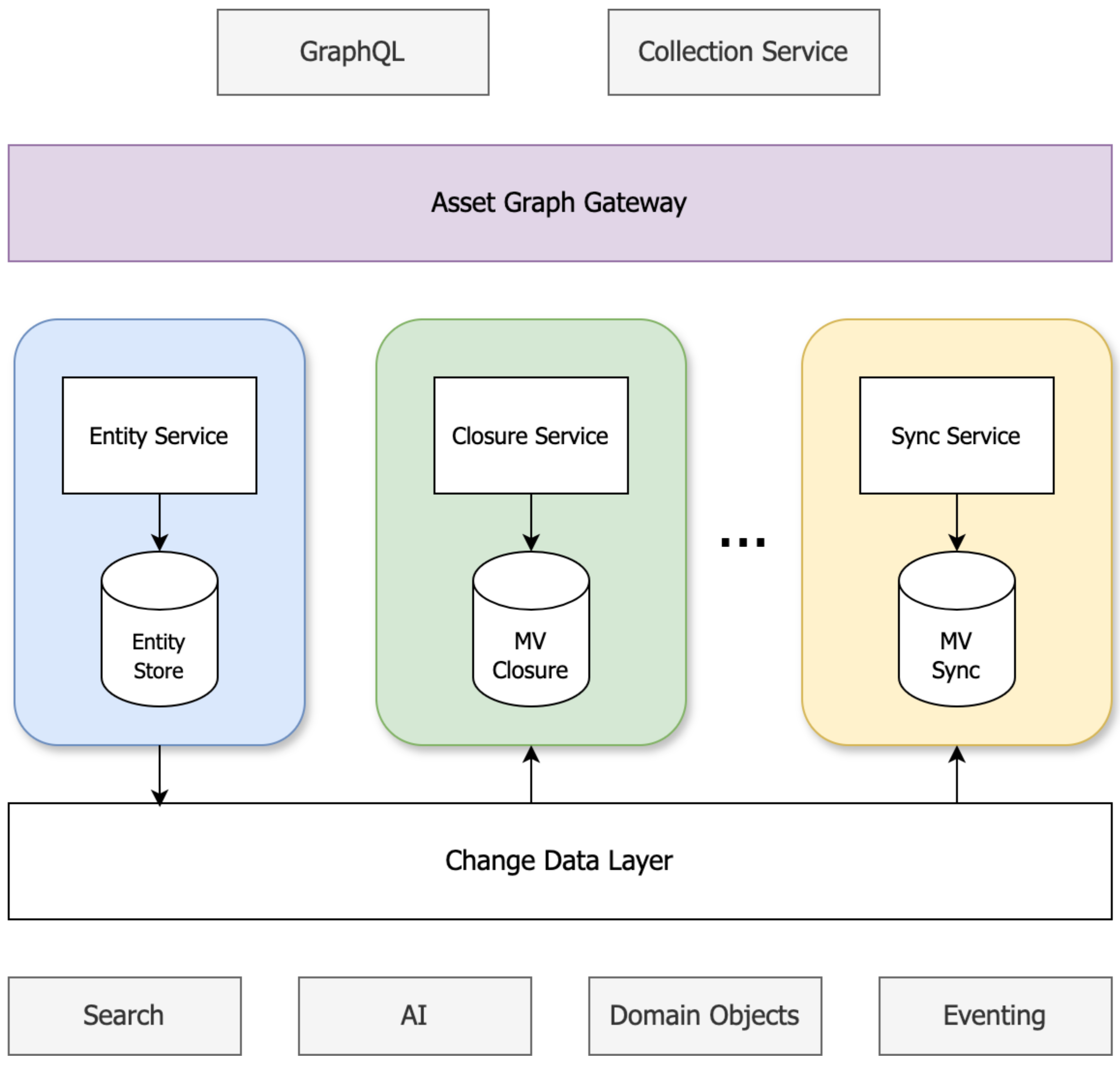
## Key Constructs of the Asset Graph:

- Collections: The top-level container.
- Spaces: The second-level container.
- Assets: The lowest-level entities, represented as nodes in the Asset Graph.
- Relationships: The edges between assets. In the future, relationships may extend to support connections between spaces.
- Snapshots: Point-in-time references to assets, relationships, and spaces.

## Different Workloads:

- A space can contain millions of assets, each with up to 1,000 dynamic properties, varying between assets within a space. For example, a 99th percentile use-case might involve 10,000 property definitions per space. The system must handle high ingestion rates and support queries on these properties both within and across spaces.
- A collection might have a graph structure 10,000 levels deep and 10,000 nodes wide, requiring efficient traversal, ordering, and filtering capabilities.
- A space can contain millions of assets, each with distinct permissions that must be applied during traversal, whether through closure or synchronization.

## High-level architecture



## Read Access Patterns:

Asset Graph supports diverse read access patterns optimized for different use cases across Manufacturing, AEC, and Media & Entertainment industries. These patterns are designed to handle large-scale data retrieval with high performance and flexibility.

### Direct Access Patterns

#### Get by ID

- **Pattern**: Direct asset/relationship retrieval using unique identifiers
- **Scale**: Single asset retrieval from spaces containing millions of assets
- **Performance**: Sub-100ms response time for direct ID lookups
- **Use Cases**:
- Loading specific CAD components in Fusion 360
- Retrieving building elements in Revit workflows
- Accessing media assets in animation pipelines

#### Batch Get Operations

- **Pattern**: Retrieving multiple assets/relationships by ID in single request
- **Scale**: Batches of 100-10,000 entities per request
- **Optimization**: Parallel DynamoDB batch operations with intelligent batching
- **Use Cases**:
- Loading assembly components and their relationships
- Bulk asset synchronization for offline workflows
- Prefetching related entities for UI rendering

## Filter and Search Patterns

### Property-Based Filtering

- **Pattern**: Query assets based on component properties using various operators
- **Supported Operations**:
- Equality (`==`): `material.type == "steel"`
- Existence (`exists`): `metadata.createdBy exists`
- Contains (`contains`): `tags contains "structural"`
- Range queries (`>=`, `<=`): `geometry.volume >= 100.0`
- Logical operators (`AND`, `OR`): `material.type == "steel" AND geometry.volume >= 100.0`
- **Scale**: Filtering across millions of assets with 1,000+ dynamic properties per asset
- **Performance**: Index-backed queries with sub-second response times
- **Use Cases**:
- Finding all structural elements in a building model
- Locating components by manufacturing specifications
- Filtering assets by creation date or modification time

### Cross-Space Search

- **Pattern**: Search and filter operations spanning multiple spaces within a collection
- **Scale**: Querying across spaces containing 10,000+ property definitions
- **Complexity**: Handling heterogeneous schemas across different spaces
- **Use Cases**:
- Multi-discipline BIM coordination (architectural + structural + MEP)
- Cross-assembly part searches in manufacturing
- Global asset searches across project phases

### Full-Text Search

- **Pattern**: Text-based search across asset properties and metadata
- **Implementation**: OpenSearch-backed indexing for text content
- **Scale**: Searching millions of assets with rich text content
- **Features**:
- Fuzzy matching and stemming
- Multi-language support
- Relevance scoring and ranking
- **Use Cases**:
- Finding assets by description or comments
- Searching for components by manufacturer names
- Locating elements by specification text

## Graph Traversal Patterns

### Transitive Closure (Relationships Traversal)

- **Pattern**: Deep, multi-level graph traversal following relationship edges
- **Scale**: Traversing graphs 10,000 levels deep and 10,000 nodes wide
- **Traversal Types**:
- **Breadth-First**: Exploring neighbors at current depth before going deeper
- **Depth-First**: Deep exploration of single paths before backtracking
- **Bi-directional**: Starting from multiple nodes and meeting in the middle
- **Filtering**: Apply property filters during traversal to prune irrelevant paths
- **Ordering**: Sort results by properties, relationship types, or graph distance
- **Performance**: Optimized for complex mechanical assemblies and building systems
- **Use Cases**:
- Assembly explosion/implosion in CAD workflows
- Building system tracing (HVAC, electrical, plumbing)
- Dependency analysis in manufacturing processes
- Impact analysis for design changes

### Closure with Permissions

- **Pattern**: Traversal with fine-grained access control applied per asset
- **Scale**: Millions of assets with distinct permission sets per user/role
- **Security**: Permission evaluation during traversal without exposing unauthorized data
- **Performance**: Optimized permission checking to avoid N+1 query problems
- **Use Cases**:
- Multi-tenant environments with isolated data access
- Role-based visibility in collaborative design
- Client-specific data exposure in managed services

### Shortest Path Queries

- **Pattern**: Finding optimal paths between two or more assets

- **Algorithms**: Dijkstra's algorithm with weighted relationships
- **Scale**: Path finding in large, complex graph structures
- **Use Cases**:
- Route optimization in building navigation
- Process flow analysis in manufacturing
- Dependency path analysis in system design

## Temporal and Versioning Patterns

### Time-Based Queries

- **Pattern**: Retrieving assets and relationships based on temporal criteria
- **Time Dimensions**:
- Creation time: `createdAt >= "2024-01-01"`
- Modification time: `lastModified <= "2024-12-31"`
- Custom timestamps: Project milestones, manufacturing dates
- **Scale**: Temporal queries across millions of time-stamped entities
- **Use Cases**:
- Project timeline reconstruction
- Change history analysis
- Incremental synchronization workflows

### Snapshot-Based Access

- **Pattern**: Point-in-time retrieval using snapshot references
- **Consistency**: Consistent view of assets and relationships at specific moments
- **Scale**: Managing snapshots across collections with millions of entities
- **Use Cases**:
- Design milestone comparisons
- Rollback to previous versions
- Historical analysis and auditing
- Concurrent access to stable data views

### Version Comparison (Diff/Delta)

- **Pattern**: Computing differences between two versions of entities or containers
- **Scale**: Comparing containers with 100K-1M elements
- **Granularity**:
- Asset-level changes (properties, relationships)
- Space-level changes (asset additions/deletions)
- Collection-level changes (structural modifications)
- **Performance**: Optimized algorithms for large-scale difference computation
- **Use Cases**:
- Change detection in design iterations
- Conflict resolution in collaborative workflows
- Impact analysis for design modifications
- Synchronization between different tools

## Synchronization Patterns

### Incremental Sync

- **Pattern**: Retrieving only changed entities since last synchronization
- **Mechanisms**:
- Timestamp-based: Using `lastModified` timestamps
- Snapshot-based: Comparing against known snapshot references
- Change log-based: Using event streams for incremental updates
- **Scale**: Syncing subsets from millions of entities efficiently
- **Use Cases**:
- Desktop application synchronization
- Mobile offline/online synchronization
- Cross-system data integration

### Bulk Synchronization

- **Pattern**: Large-scale data synchronization for desktop and mobile clients
- **Scale**: Millions of concurrent users with varying sync requirements
- **Optimization**: Intelligent batching and compression for bandwidth efficiency
- **Use Cases**:
- Initial application setup and data loading
- Periodic full synchronization for data consistency
- Migration between different system versions

## Performance Characteristics

### Scalability Metrics

- **Asset Scale**: Spaces with millions of assets, each having up to 1,000 dynamic properties
- **Property Definitions**: Up to 10,000 property definitions per space (99th percentile)
- **Graph Complexity**: Graphs up to 10,000 levels deep and 10,000 nodes wide
- **Component Size**: 10KB-100KB average, 1MB at 99th percentile

### Performance Targets

- **Direct Access**: Sub-100ms for single asset retrieval
- **Filtered Queries**: Sub-second response for complex property-based filters
- **Graph Traversal**: Optimized for deep traversals with early termination
- **Search Operations**: Near real-time results for text-based searches
- **Bulk Operations**: High throughput for batch retrieval operations

These read access patterns enable Asset Graph to support diverse industry workflows while maintaining high performance and scalability across different data access scenarios.

## Use-Cases Based on Different Industries:

- A write-heavy system capable of handling bulk writes of millions of entities in a batch. These writes need to be atomic and highly scalable, accommodating multiple large and small batch requests.
- Millions of desktop and mobile users who can sync/read entities based on time, snapshots, or versions.
- Efficient closure traversal starting from a specific entity, capable of deep, multi-level traversals with filtering and ordering capabilities.
- A scalable search function based on multiple key/value pairs with operations such as ==, exists, contains, >=, or <= with AND/OR logical operators.
- A diff / delta computation between 2 versions of a entity or a parent container that can contain 100K-1M elements
- Size of the schema-based components to be 10KB-100KB on avg and p99 is 1MB