Independent Submission
Internet-Draft
Intended status: Informational
Expires: 28 August 2026

L. Muscariello
R. Polic
Cisco

24 February 2026

Agent Directory Service
draft-mp-agntcy-ads-01

Abstract

The Agent Directory Service (ADS) is a distributed directory service designed to store metadata for AI agent applications. This metadata, stored as directory records, enables the discovery of agent applications with specific skills for solving various problems. The implementation features distributed directories that interconnect through a content-routing protocol.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://spec.dir.agncty.org. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mp-agntcy-ads/.

Source for this draft and an issue tracker can be found at https://github.com/agntcy/dir.

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 28 August 2026.

Table of Contents

1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Introduction

Multi-Agent Systems (MAS) represent a new paradigm in distributed computing where software components leverage Large Language Models (LLMs) to perform specialized tasks and solve complex problems through collaborative intelligence. These systems combine LLMs with contextual knowledge and tool-calling capabilities, often abstracted through Model Context Protocol (MCP) servers, enabling dynamic workflows that adapt based on stored state and environmental conditions.

The diversity and complexity of MAS architectures present unique challenges for discovery and composition. As the ecosystem of AI agents expands, developers need efficient mechanisms to:

* *Discover compatible agents* with specific skills and capabilities

* *Evaluate performance characteristics* including cost, latency, and resource requirements

* *Compose multi-agent workflows* by linking agents with complementary capabilities

* *Verify claims* about agent performance and reliability

* *Track versioning and dependencies* between agent components

The Agent Directory Service (ADS) [AGNTCY-ADS] addresses these challenges by providing a distributed directory infrastructure specifically designed for the agentic AI ecosystem. Rather than attempting to formally define MAS architectures, which would constrain the creative composition patterns emerging in this rapidly evolving field— ADS focuses on providing flexible metadata storage and discovery mechanisms. A comparison among registries which can be centralized or distributed is reported here [AI-Registry-Evolution].

2.1. Core Capabilities

ADS enables several key capabilities for the agentic AI ecosystem:

Capability-Based Discovery: Agents publish structured metadata describing their functional abilities, costs, and performance characteristics. The system organizes this information using hierarchical skill taxonomies, enabling efficient matching of capabilities to requirements.

Verifiable Claims: While agent capabilities are often subjectively evaluated, ADS provides cryptographic mechanisms for data integrity and provenance tracking. This allows users to make informed decisions about agent selection while enabling reputation systems to emerge organically.

Semantic Linkage: Components can be securely linked to create various relationships like version histories for evolutionary development, collaborative partnerships where complementary skills solve complex problems, and dependency chains for composite agent workflows.

Distributed Architecture: Built on proven distributed systems principles, ADS uses content-addressing for global uniqueness and implements distributed hash tables [DHT] for scalable content discovery across decentralized networks.

2.2. Architectural Foundation

The system leverages the Open Agentic Schema Framework (OASF) to model agent information in a structured, extensible format. OASF enables rich queries such as "What agents can solve problem A?" or "What combination of skills and costs optimizes for task B?" This schema-driven approach supports both objective metrics (token consumption, GPU requirements) and subjective evaluations (user ratings, task completion quality).

Agent records are organized using modular extensions—reusable components like MCP server definitions, prompt-based agents, and evaluation metrics. This modular approach facilitates composition and reuse across different MAS architectures while maintaining flexibility for innovative use cases.

The underlying storage layer integrates with OCI (Open Container Initiative) [OCI.Image] standards, enabling interoperability with existing container ecosystems and leveraging mature tooling for content distribution and verification.

This document details the technical architecture of ADS, covering the record storage layer, security model, distributed data discovery mechanisms, and data distribution protocols between storage nodes.

3. Storage Architecture

ADS implements a decentralized storage architecture built on OCI (Open Container Initiative) registries as the foundational object storage layer. This design choice enables the system to leverage mature, standardized container registry infrastructure while achieving the speed, scalability, and security requirements of a distributed agent directory.

3.1. Content-Addressed Storage

The storage architecture centers on globally unique Content Identifiers (CID) that provide several critical properties for a distributed agent directory:

Immutability: Content identifiers are cryptographically derived from the data they represent, ensuring that any modification results in a different identifier. This property is essential for maintaining data integrity in agent records and enabling verifiable claims about agent capabilities.

Deduplication: Identical content automatically receives the same identifier across all nodes in the network, eliminating storage redundancy and reducing bandwidth requirements when the same agent components are referenced by multiple systems.

Verifiability: Any node can independently verify that received content matches its identifier, providing built-in protection against data corruption or tampering during transmission.

Location Independence: Content can be retrieved from any node that possesses it, as the identifier serves as a universal pointer that abstracts away physical storage locations.

3.2. OCI Integration

ADS leverages OCI (Open Container Initiative) specifications for storing and distributing agent records as OCI artifacts, offering several advantages:

3.2.1. Standards Compliance

By building on OCI specifications, ADS inherits compatibility with the extensive ecosystem of container registry tools, security scanners, and management platforms. This includes:

* *Authentication and authorization* mechanisms already deployed in enterprise environments
* *Content signing and verification* through tools like Notary and cosign
* *Vulnerability scanning* capabilities that can be extended to agent security assessments
* *Content delivery networks* optimized for OCI artifact distribution

3.3. Record Artifact Specification

ADS stores agent records as Record Artifacts following the OCI Image Manifest Specification [OCI.Manifest] and adhering to the OCI artifacts guidance [OCI.Artifact]. A Record Artifact is an AGNTCY OASF Record packaged as an OCI artifact.

3.3.1. Manifest Structure

The manifest structure MUST include the following properties:

* mediaType string

  This REQUIRED property MUST be application/ vnd.oci.image.manifest.v1+json.

* artifactType string

  This REQUIRED property MUST be application/ vnd.agntcy.dir.record.v1+json. Future versions of the Record Artifact Specification MAY define additional artifact types.

* config descriptor

  This REQUIRED property MUST reference an empty configuration with media type application/vnd.oci.empty.v1+json, indicating that no additional configuration data is needed for Record Artifacts.

* layers array of objects

  This REQUIRED property MUST contain one or more layer descriptors representing OASF Record components.

* subject descriptor

  This OPTIONAL property MAY reference another Record Artifact to create linkage between records for version histories.

3.3.2. Layer Structure

Each layer descriptor MUST include the following properties:

* mediaType string

  This REQUIRED property specifies the layer content type. Implementations MUST support the following media types:

  - application/vnd.agntcy.oasf.types.{version}.Record+json: The first layer (index 0) MUST use this media type and contains base record data. This layer MUST use the data field for inline storage (base64-encoded) since the content is small, frequently accessed, and unique to each record. Object annotations, schema_version and created_at are stored as part of layer annotations.

  - application/vnd.agntcy.oasf.types.{version}.Skill+json: Contains skill definition data. Object annotations, id and name are stored as part of layer annotations.

- application/vnd.agntcy.oasf.types.{version}.Domain+json: Contains domain definition data. Object annotations, id and name are stored as part of layer annotations.

- application/vnd.agntcy.oasf.types.{version}.Locator+json: Contains locator definition data referencing the location where the agent can be accessed or deployed. Object annotations, type and url are stored as part of layer annotations and URL fields.

- application/vnd.agntcy.oasf.types.{version}.Module+json: Contains module definition data. Object annotations, id and name are stored as part of layer annotations.

Media types MUST map to application/ vnd.{schema_uri}.{version}.{type}+{encoding} format, where - {schema_uri} corresponds to the OASF protobuf schema namespace - {version} corresponds to a specific OASF type version (e.g., v1alpha2) - {type} is the OASF type name (e.g. Record) - {encoding} MUST be json

* urls array of strings

  This OPTIONAL property MAY contain URLs pointing to external resources, particularly useful for Locator layers to specify deployment targets.

* annotations string-string map

  This OPTIONAL property contains arbitrary attributes. Implementations SHOULD use the predefined annotation keys:

  - agntcy.oasf.record/schema_version: Schema version (for Record layers)

  - agntcy.oasf.record/created_at: Creation timestamp (for Record layers)

  - agntcy.oasf.locator/type: Locator type (for Locator layers)

  - agntcy.oasf.skill/{id,name}: Skill id/name (for Skill layers)

  - agntcy.oasf.domain/{id,name}: Domain id/name (for Domain layers)

  - agntcy.oasf.module/{id,name}: Module id/name (for Module layers)

3.3.3. Example Record Artifacts


```
   {
     "schemaVersion": 2,
     "mediaType": "application/vnd.oci.image.manifest.v1+json",
     "artifactType": "application/vnd.agntcy.dir.record.v1+json",
     "config": {
       "mediaType": "application/vnd.oci.empty.v1+json",
       "digest": "sha256:44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff8a
",
       "size": 2
     },
     "layers": [
       {
         "mediaType": "application/vnd.agntcy.oasf.types.v1alpha2.Record+json",
         "data": "<BASE64_ENCODED_RECORD_DATA>",
         "digest": "sha256:d5815835051dd97d800a03f641ed8162877920e734d3d705b698912602b8c7
63",
         "size": 216,
         "annotations": {
           "agntcy.oasf.record/schema_version": "0.7.0",
           "agntcy.oasf.record/created_at": "2026-01-06T00:00:00Z"
         }
       },
       {
         "mediaType": "application/vnd.agntcy.oasf.types.v1alpha2.Skill+json",
         "digest": "sha256:44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff
8a",
         "size": 2,
         "annotations": {
           "agntcy.oasf.skill/id": "10201",
           "agntcy.oasf.skill/name": "Natural Language Processing/Sentiment Analysis"
         }
       },
       {
         "mediaType": "application/vnd.agntcy.oasf.types.v1alpha2.Locator+json",
         "digest": "sha256:44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff
8a",
         "size": 2,
         "urls": [
           "https://ghcr.io/agntcy/agent:latest",
           "ipfs://bafybeibwzif37xyzabcdefg"
         ],
         "annotations": {
           "agntcy.oasf.locator/type": "docker-image"
         }
       }
     ],
     "subject": {
       "mediaType": "application/vnd.oci.image.manifest.v1+json",
       "artifactType": "application/vnd.agntcy.dir.record.v1+json",
       "digest": "sha256:7e346bc58473bc1d8a98776fa2a89a3e2a446b27f0e8a33ad49c3d4f28b6471d
",
```

```
      "size": 702
    }
  }
```

3.3.4. Record Reconstruction

Applications consuming Record Artifacts MUST implement the following reconstruction algorithm:

1. *Pull Manifest*: Retrieve the OCI image manifest from the registry using standard OCI Distribution API operations

2. *Extract Base Record*: Parse the first layer (index 0) with media type application/vnd.agntcy.oasf.types.{version}.Record+json and base64-decode the data field to obtain the base record JSON data

3. *Retrieve Component Layers*: For each subsequent layer (Skill, Domain, Locator, Module):

   * Fetch the blob content from the registry using the layer’s digest

   * Verify the blob’s SHA-256 digest matches the descriptor’s digest field

   * Parse the JSON content according to the layer’s mediaType

   * Merge the component data into the appropriate field of the base record

   * Merge descriptor annotations into the corresponding component metadata

4. *Validate Schema*: Validate the reconstructed record against the OASF schema version specified in the agntcy.oasf.record/schema_version annotation

3.4. Application Integration

3.4.1. Open Agent Schema Framework (OASF)

The Open Agent Schema Framework (OASF) [AGNTCY-OASF] complements Record Artifact Specification by defining the actual data models for agent-related information referenced in the OCI artifacts.

3.4.2. Third-Party Applications

Third-party applications can build on top of the Record Artifact Specification to create tools and services for managing, distributing, and utilizing agent records. This can be accomplished by leveraging existing OCI ecosystem components such as registries, clients, and tooling, or by developing custom solutions that link to Record Artifacts.

3.4.3. Runtime Environments

Runtime environments that deploy and manage agents can leverage Record Artifacts to obtain the necessary information for launching and managing agent instances based on the definitions contained within the artifacts. Implementations can provide management of process lifecycles, monitoring, and other capabilities based on the metadata defined in the records, or by linking records with runtime-specific artifacts.

3.5. Artifact Organization

Agent records are stored as OCI artifacts with a structured organization. Records MUST be addressed by digest for immutable retrieval. Tags are mutable aliases and can be used as human-readable pointers, but implementations MUST NOT assume a tag resolves to more than one manifest at a time. The examples below show digest-pinned references:

```
null_repo/records/
├── skills/
│   ├── natural_language_processing/
│   │   ├── sentiment-analysis:bert-v1@sha256:abc123...
│   │   ├── sentiment-analysis:roberta-v1@sha256:def456...
│   │   ├── sentiment-analysis:distilbert-v1@sha256:ghi789...
│   │   ├── text-classification:bert-v2@sha256:abc123...
│   │   └── emotion-detection:bert-v1@sha256:abc123...
│   ├── images_computer_vision/
│   │   ├── object-detection:yolo-v2.1@sha256:jkl012...
│   │   ├── object-detection:rcnn-v2.1@sha256:mno345...
│   │   └── scene-understanding:yolo-v1@sha256:jkl012...
│   └── analytical_skills/
│       └── mathematical:v1.5.0@sha256:pqr678...
├── evaluations/
│   ├── performance-metrics:latest@sha256:stu901...
│   └── benchmark-results:v1.0.0@sha256:vwx234...
└── compositions/
    ├── security-analyst:v3.0.0@sha256:yza567...
    └── research-assistant:v2.2.0@sha256:bcd890...
```

This naming scheme demonstrates that the same content identifier can belong to multiple skills, reflecting the reality that many AI agents are multi-capable. For example, the BERT-based agent (sha256:abc123...) appears under multiple skill categories: natural_language_processing/sentiment-analysis, natural_language_processing/text-classification, and natural_language_processing/emotion-detection, representing different capabilities of the same underlying agent implementation. Similarly, the YOLO vision model (sha256:jkl012...) provides both object-detection and scene-understanding capabilities under images_computer_vision.

This cross-referencing approach allows agents to be discovered through any of their supported capabilities while maintaining unique addressability through content identifiers. Each skill category can have its own versioning and metadata, enabling fine-grained capability management even when multiple skills share the same underlying implementation.

Each artifact contains structured metadata following OASF schemas, enabling rich queries and capability matching across all variants within a given category.

3.5.1. Multi-Registry Federation

The architecture supports federation across multiple registry instances, enabling:

* *Organizational boundaries*: Different organizations can maintain their own registries while participating in the global directory

* *Geographic distribution*: Content can be replicated to registries closer to consumers, reducing latency

* *Specialization*: Registries can focus on specific domains (e.g., medical AI agents, financial analysis tools)

* *Redundancy*: Critical agent records can be replicated across multiple registries for availability

4. MAS Data Discovery

ADS implements a two-level mapping system that enables efficient discovery of Multi-Agent System components through a distributed hash table [DHT] architecture. This approach separates capability-based discovery from content location, providing both scalability and flexibility in agent retrieval.

4.1. Skill Taxonomy

Effective agent discovery in multi-agent systems requires sophisticated organization of capabilities and skills. ADS employs a hierarchical skill taxonomy that serves as the foundation for efficient search and discovery operations across the distributed network.

4.1.1. The Challenge of Capability Search

Traditional keyword-based search approaches face significant limitations when applied to agent discovery:

Vocabulary Fragmentation: Different publishers may describe similar capabilities using varying terminology. For example, "sentiment analysis," "opinion mining," and "emotional classification" may all refer to similar agent capabilities, leading to search results that miss relevant agents due to terminology mismatches.

Scale Complexity: As the number of agents in the ecosystem grows, exhaustive search across all records becomes computationally prohibitive. Without structured organization, every query potentially requires examining every agent record, leading to poor performance characteristics.

Semantic Relationships: Many agent capabilities have natural hierarchical relationships that flat keyword systems cannot capture. An agent capable of "named entity recognition" is inherently relevant to searches for broader "text analysis" capabilities, but keyword matching alone cannot establish these connections.

4.1.2. Taxonomy-Driven Search Optimization

ADS addresses these challenges through a structured hierarchical taxonomy that provides several critical optimization benefits:

Search Space Partitioning: The taxonomy enables efficient partitioning of the search space. When processing a query for "computer vision" capabilities, the system can immediately focus on the relevant taxonomy branch, eliminating the need to examine agents in unrelated categories like natural language processing or mathematical reasoning.

Index Structure Optimization: The hierarchical organization allows the distributed hash table to create specialized indices for different taxonomy branches. Rather than maintaining a single massive index, the DHT can distribute indexing responsibility across nodes, with each node specializing in specific capability domains.

Query Semantic Expansion: The taxonomy enables intelligent query expansion where searches automatically include semantically related subcategories. A search for "text analysis" can transparently include results from "sentiment analysis," "entity extraction," and "text classification" without requiring users to explicitly enumerate all relevant subcategories.

Standardized Vocabulary: By providing a canonical taxonomy, ADS reduces terminology fragmentation. Publishers are encouraged to tag their agents using standardized skill categories, improving search precision and recall across the ecosystem.

4.2. Two-Level Mapping Architecture

The discovery system operates through two distinct mapping layers:

4.3. Skill Taxonomy for Search Optimization

ADS employs a hierarchical skill taxonomy to optimize search performance and enable efficient capability-based discovery. Taxonomies provide several critical advantages for agent discovery systems:

Search Space Reduction: Rather than performing exhaustive searches across all agent records, taxonomies allow the system to quickly narrow the search space to relevant categories. When a user queries for "natural language processing" capabilities, the system can immediately identify the subset of agents tagged with NLP skills without examining agents focused on computer vision or mathematical reasoning.

Hierarchical Organization: Skills are organized in a tree-like structure *that reflects natural relationships between capabilities. For example:

```
Natural Language Processing
├── Text Analysis
│   ├── Sentiment Analysis
│   ├── Named Entity Recognition
│   └── Text Classification
├── Language Generation
│   ├── Text Summarization
│   ├── Content Creation
│   └── Translation
└── Conversational AI
    ├── Dialogue Management
    ├── Intent Recognition
    └── Response Generation
```

This hierarchy enables both specific queries ("sentiment analysis agents") and broader capability searches ("all natural language processing agents") while maintaining efficient indexing structures.

Query Expansion and Refinement: Taxonomies support automatic query expansion *where searches for parent categories can include relevant child categories. A *query for "text analysis" can automatically include agents tagged with *"sentiment analysis," "named entity recognition," and "text classification" *without requiring users to know all specific subcategories.

Semantic Consistency: Standardized taxonomies reduce ambiguity and improve *search precision by providing consistent terminology across the ecosystem. This *prevents fragmentation where similar capabilities are described using different *terms by different publishers.

Scalable Indexing: The hierarchical structure enables efficient distributed *indexing where different DHT nodes can specialize in specific taxonomy *branches, distributing both storage load and query processing across the *network.

4.4. Additional Taxonomies

While skills form the primary taxonomy for capability-based discovery, ADS supports multiple parallel taxonomies to enable rich, multi-dimensional agent classification and search.

4.4.1. Domain Taxonomy

The domain taxonomy organizes agents by their application domains, representing the broader problem spaces or industries where agents are designed to operate:

```
Application Domains
├── Networking
│   ├── Network Configuration
│   ├── Traffic Analysis
│   └── Protocol Implementation
├── Security
│   ├── Threat Detection
│   ├── Vulnerability Assessment
│   └── Access Control
├── Software Development
│   ├── Code Generation
│   ├── Testing Automation
│   └── Documentation
├── Finance and Business
│   ├── Risk Analysis
│   ├── Market Research
│   └── Process Automation
└── Healthcare
    ├── Medical Imaging
    ├── Clinical Decision Support
    └── Drug Discovery
```

Domain classification enables users to discover agents that are specifically tuned for their operational context, even if those agents share similar underlying skills with agents from other domains.

4.4.2. Module Taxonomy

The module taxonomy categorizes agents by the OASF modules they implement, facilitating the discovery of agents compatible with specific system architectures. Modules are organized into the OASF module categories (Core and Integration), and each module captures a structured integration surface or capability extension:

The Agent Spec module aligns with the Open Agent Spec specification [Oracle-AgentSpec].

```
Modules
├── Core
│   ├── Language Model (‘language_model‘)
│   ├── Evaluation (‘evaluation‘)
│   └── Observability (‘observability‘)
└── Integration
    ├── MCP (Model Context Protocol, ‘mcp‘)
    ├── A2A (Agent-to-Agent Communication, ‘a2a‘)
    ├── Agent Spec (‘agentspec‘)
    └── ACP (‘acp‘, deprecated)
```

Module identifiers use the OASF module name values. The acp module is deprecated in OASF; use a2a when possible.

4.4.3. Multi-Dimensional Search

The parallel taxonomy system enables sophisticated multi-dimensional queries that combine criteria across different classification axes. *All searches must include at least one skill criterion as the mandatory foundation*, with domain and module taxonomies providing additional filtering dimensions:

* *Skill + Domain*: "Find natural language processing agents specialized for healthcare applications"

* *Skill + Module*: "Discover computer vision agents that support MCP integration"

* *Skill + Module + Domain*: "Locate natural language processing agents with observability modules for manufacturing applications"

Skills as Search Foundation: The skills taxonomy serves as the primary index structure in the DHT, making skill-based criteria mandatory for efficient query resolution. This design ensures that:

* *Query Performance*: All searches leverage the optimized skills-to-CID mapping as the starting point, providing consistent performance characteristics

* *Result Relevance*: Domain and module filters are applied to skill-based result sets, ensuring functional capability remains the core selection criterion

* *Index Efficiency*: The DHT can optimize storage and lookup patterns around the skills taxonomy while supporting supplementary filtering through domains and modules

Domain-only or module-only queries are not supported, as they would bypass the primary indexing structure and provide results that may not have the functional capabilities required by the requesting system.

This multi-taxonomic approach provides the flexibility to support diverse use cases while maintaining efficient indexing and search performance across all dimensions.

4.4.4. Skills-to-CID Mapping

The first level maps agent capabilities and skills to their corresponding Content Identifiers (CID):

```
Skills Index:
"natural_language_processing" → ["sha256:abc123...", "sha256:def456...", "sha256:ghi789.
..."]
"images_computer_vision"      → ["sha256:jkl012...", "sha256:mno345..."]
"analytical_skills"           → ["sha256:pqr678...", "sha256:abc123..."]
"multi_modal"                 → ["sha256:stu901...", "sha256:vwx234..."]
```

This mapping enables queries such as "find all agents capable of natural language processing" to quickly resolve to a set of content identifiers without needing to search through individual agent records.

4.4.5. CID-to-PeerID Mapping

The second level maps Content Identifiers to the Peer IDs of nodes that store the corresponding agent records:

```
Content Location Index:
"sha256:abc123..." → ["12D3KooWBhvxmvKvTYGJvXjnGBp7Ybr9WyoXkZvFnRtC4aBcDeFg",
                      "12D3KooWXyZrUvHzPqKvTYGJvXjnGBp7Ybr9WyoXkZvFnRtC5gHi"]
"sha256:jkl012..." → ["12D3KooWZaBcDeFgXyZrUvHzPqKvTYGJvXjnGBp7Ybr9WyoXkZvF",
                      "12D3KooWGHiJkLmNoPqRsTuVwXyZ123456789AbCdEfGhIjKlMnO"]
```

This separation allows the system to: - *Optimize for capability queries* without requiring knowledge of data locations

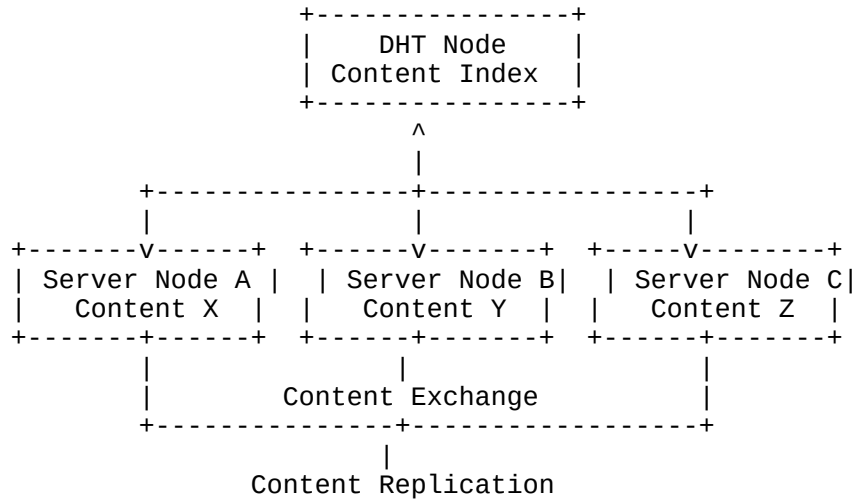
* *Enable dynamic content replication* as peer availability changes

* *Support multiple storage strategies* for the same content across different peers

4.5. DHT-Based Discovery Process

The Distributed Hash Table stores and maintains both mapping layers across the network:

ADS uses [Kad-DHT] [DHT] for server and content discovery by using the libp2p implementation that constitutes the IPFS core DHT [libp2p-kad-dht].



Flow:
1. Servers register content with DHT
2. DHT maintains content-to-server mappings
3. Servers query DHT to locate content
4. DHT returns list of servers hosting content
5. Servers download content from peers

4.5.1. Skill Registration

When agents are published to the network:

1. *Capability Extraction*: The system parses OASF records to extract skills, domains, and modules

2. *DHT Updates*: Skills-to-CID mappings are distributed across DHT nodes using consistent hashing

3. *Location Registration*: Peer nodes register themselves as providers for specific CIDs

4.5.2. Discovery Query Resolution

Agent discovery follows a three-phase process:

1. *Capability Resolution*: Query "agents with skill X" resolves to a list of relevant CIDs via DHT lookup

2. *Location Resolution*: For each discovered CID, query DHT to find peer nodes storing the content

3. *Result Aggregation*: Combine capability matches with location information to produce actionable discovery results

```
Discovery Flow:
Query: "natural_language_processing" AND "finance_and_banking"
   ↓
Phase 1: Skills → CIDs
   DHT["natural_language_processing"] → ["sha256:abc123...", "sha256:def456..."]
   DHT["finance_and_banking"] → ["sha256:abc123...", "sha256:yza567..."]
   Intersection → ["sha256:abc123..."]
   ↓
Phase 2: CIDs → Peer IDs
   DHT["sha256:abc123..."] → ["12D3KooW...", "12D3KooX..."]
   ↓
Result: Agent sha256:abc123... available from peers 12D3KooW... and 12D3KooX...
```

4.5.3. Additional Tanomoxies

4.6. Content Distribution via OCI Protocol

Once discovery identifies the relevant CIDs and their hosting peers, the actual agent records are retrieved using the OCI distribution protocol:

4.6.1. Peer-to-Peer Synchronization

The discovered list of CIDs enables efficient content synchronization [OCI.Distribution] between peers:

1. *Content Negotiation*: Requesting peer queries hosting peers for available agent records

2. *OCI Pull Operations*: Standard OCI registry pull commands retrieve agent artifacts and metadata

3. *Incremental Sync*: Only missing or updated content is transferred, reducing bandwidth requirements

4. *Verification*: Content integrity is verified through cryptographic hash validation during transfer

4.6.2. Distribution Strategies

The system supports multiple distribution patterns, each with distinct trade-offs and operational considerations:

On-Demand Retrieval: Records are pulled from remote peers only when specifically requested, minimizing local storage requirements.

Trade-offs: While this approach minimizes storage costs and ensures fresh content, it introduces several challenges:

* *Query Latency*: Each request requires network round-trips to locate and retrieve content, increasing response times

* *Network Cost*: Spurious or exploratory requests generate unnecessary network traffic and computational overhead

* *DoS Vulnerability*: The system becomes susceptible to denial-of-service attacks where malicious actors can trigger expensive content retrieval operations by flooding the network with requests for non-existent or rarely-accessed agents

* *Scalability Limits*: Performance degrades as the network grows due to increased query coordination overhead

Proactive Caching: Popular or frequently accessed agents are automatically replicated to improve query response times.

Trade-offs: This strategy offers significant scalability benefits but requires sophisticated management:

* *Performance Gains*: Dramatic reduction in query latency for popular content, enabling sub-second response times

* *Scalability*: Can handle high query volumes efficiently once popular content is cached locally

* *Popularity Measurement*: Requires implementing metrics collection and analysis to identify which agents warrant caching. This includes tracking query frequencies, download patterns, and usage statistics across the network

* *Storage Requirements*: Needs sufficient local storage capacity to maintain cached copies of popular records

* *Cache Management*: Must implement cache eviction policies, freshness validation, and synchronization mechanisms

* *Administrator Oversight*: Proactive caching policies must be configured and monitored by agent directory node administrators to balance storage costs with performance benefits

Strategic Replication: Critical agents can be replicated across multiple peers to ensure high availability and reduce single points of failure.

Trade-offs: This approach addresses availability concerns but introduces subjective complexity:

* *High Availability*: Ensures critical agents remain accessible even during peer failures or network partitions

* *Reduced Single Points of Failure*: Distributes risk across multiple storage locations

* *Subjective Criticality*: The definition of "critical" or "useful" agents varies significantly between users, organizations, and use cases. What constitutes strategic value for financial services may be irrelevant for manufacturing applications

* *Administrative Burden*: Requires agent directory node administrators to make strategic decisions about which agents warrant replication, considering factors like:

  - Organizational priorities and business requirements

  - Compliance and regulatory considerations

  - Cost-benefit analysis of storage versus availability

  - Community consensus on agent importance

* *Resource Allocation*: Strategic replication consumes storage resources that could otherwise be used for proactive caching of popular content

Administrative Management: Both Proactive Caching and Strategic Replication require active management by agent directory node administrators. Administrators must:

* Configure caching policies based on local network characteristics and storage capacity

* Monitor popularity metrics and adjust caching strategies accordingly

* Define strategic replication criteria aligned with organizational objectives

* Balance resource allocation between different distribution strategies

* Implement governance policies for content lifecycle management

This architecture provides a scalable foundation for MAS data discovery that can efficiently handle large networks of distributed agents while maintaining low latency for capability-based queries.

4.7. Agent Directory Record Examples

The following examples illustrate the structure of OASF-compliant agent records stored in the directory:


```
{
  "content_id": "sha256:abc123...",
  "record": {
    "name": "BERT Sentiment Analyzer",
    "version": "1.0.0",
    "schema_version": "0.2.0",
    "description": "Multi-capability NLP agent providing sentiment analysis, text classif
ication, and emotion detection",
    "skills": ["natural_language_processing"],
    "domains": ["finance_and_banking", "legal_and_compliance"],
    "modules": ["mcp", "observability"],
    "capabilities": {
      "threads": true,
      "interrupt_support": false,
      "callbacks": true,
      "streaming": ["text", "json"]
    }
  },
  "performance_metrics": {
    "tokens_per_second": 1000,
    "gpu_memory_mb": 4096,
    "latency_p99_ms": 150,
    "accuracy_score": 0.94
  },
  "evaluation_data": {
    "overall_rating": 4.2,
    "cost_per_million_tokens": 2.50
  },
  "registries": [
    "registry.example.com",
    "hub.agents.org"
  ],
  "last_updated": "2025-08-07T10:30:00Z"
}
```

```
{
  "content_id": "sha256:jkl012...",
  "record": {
    "name": "YOLO Vision Agent",
    "version": "2.1.0",
    "schema_version": "0.2.0",
    "description": "Computer vision agent for object detection and scene understanding",
    "skills": ["images_computer_vision"],
    "domains": ["transportation_logistics_mobility", "manufacturing_and_industrial_operat
ions"],
    "modules": ["agentspec", "observability"],
    "capabilities": {
      "threads": false,
      "interrupt_support": true,
      "callbacks": false,
      "streaming": ["image", "json"]
    }
  },
  "performance_metrics": {
    "inference_fps": 30,
    "gpu_memory_mb": 8192,
    "detection_accuracy_map": 0.89,
    "processing_latency_ms": 33
  },
  "evaluation_data": {
    "overall_rating": 4.7,
    "cost_per_image": 0.05
  },
  "registries": [
    "vision.agents.com",
    "registry.example.com"
  ],
  "last_updated": "2025-08-07T14:20:00Z"
}
```

```
{
  "content_id": "sha256:pqr678...",
  "record": {
    "name": "Mathematical Reasoning Agent",
    "version": "1.5.0",
    "schema_version": "0.2.0",
    "description": "Agent specialized in mathematical problem solving and analytical reas
oning",
    "skills": ["analytical_skills", "natural_language_processing"],
    "domains": ["education", "finance_and_banking"],
    "modules": ["a2a", "evaluation"],
    "capabilities": {
      "threads": true,
      "interrupt_support": true,
      "callbacks": true,
      "streaming": ["text", "latex"]
    }
  },
  "performance_metrics": {
    "problems_per_minute": 12,
    "cpu_cores": 4,
    "memory_mb": 2048,
    "accuracy_on_gsm8k": 0.87
  },
  "evaluation_data": {
    "overall_rating": 4.5,
    "cost_per_problem": 0.10
  },
  "registries": [
    "math.agents.edu",
    "registry.example.com"
  ],
  "last_updated": "2025-08-07T16:45:00Z"
}
```


These examples demonstrate how the DHT indexing system extracts skills and domains from agent records to populate the Skills-to-CID mappings, enabling efficient capability-based discovery across the distributed network.

4.8. Security Model

The OCI-based architecture provides multiple layers of security that address the unique challenges of distributed agent directories:

4.8.1. Cryptographic Integrity

ADS leverages the OCI layer’s built-in cryptographic mechanisms to ensure data integrity:

Automatic Hash Computation: The OCI registry layer automatically computes SHA-256 hash digests for all stored artifacts. These content identifiers are generated transparently during the push operation, ensuring that every agent record has a cryptographically verifiable fingerprint without additional overhead.

Tamper Detection: Content addressing ensures immediate tamper detection through cryptographic hash verification. Any modification to an agent record—whether malicious or accidental—results in a different hash digest, making unauthorized changes immediately detectable during retrieval operations.

End-to-End Verification: Clients can independently verify that received content matches its advertised identifier, providing built-in protection against data corruption during transmission or storage without trusting intermediate network components.

4.8.2. Content Provenance and Digital Signatures

ADS integrates with Sigstore, a security framework for OCI storage, to provide comprehensive content provenance and authenticity guarantees:

Sigstore Integration: The system leverages Sigstore’s security framework to provide verifiable proof of when and by whom agent records were signed. This creates an immutable audit trail that cannot be retroactively modified, enabling forensic analysis of agent deployment history.

Keyless Signing: Sigstore’s keyless signing approach eliminates the complexity and security risks associated with long-lived cryptographic keys:

* *Identity-Based Authentication*: Uses OpenID Connect (OIDC) [OpenID.Auth] tokens from trusted identity providers (GitHub, Google, Microsoft) to authenticate publishers at signing time

* *Short-Lived Certificates*: Issues ephemeral signing certificates valid only for minutes, reducing the window of potential key compromise

* *Automatic Key Rotation*: Eliminates the need for manual key management, distribution, and rotation procedures

* *Scalable Trust*: Publishers don’t need to maintain or distribute public keys, making the system accessible to individual developers and large organizations alike

Transparency and Verification: All signatures are stored directly in OCI storage alongside the agent artifacts and public keys, providing:

* *Public Auditability*: Anyone can verify the signing history of agent records stored in accessible registries

* *Non-Repudiation*: Publishers cannot deny having signed records that are cryptographically linked to their identity

* *Supply Chain Security*: Enables detection of compromised or unauthorized agent publications

4.8.3. Trust Boundaries and Isolation

Organizational Isolation: Separate registries maintain security boundaries between different organizations, allowing each entity to control their own agent ecosystem while still participating in the broader federated network.

Content Verification: Nodes can validate artifact integrity and signature authenticity without trusting transport layers or intermediate storage systems. This zero-trust approach ensures security even when using untrusted storage infrastructure.

Reputation Systems: The cryptographic foundation enables the development of reputation systems based on verifiable evidence rather than subjective claims. Publishers with consistent signing practices and high-quality agents can build measurable trust over time.

4.8.4. Threat Mitigation

The security model addresses several key threats to distributed agent directories:

Supply Chain Attacks: Sigstore integration and transparency logs make it difficult for attackers to inject malicious agents without detection, as all publications are cryptographically signed and publicly auditable.

Data Integrity Attacks: Automatic hash verification prevents tampering with agent records during storage or transmission, ensuring users receive authentic content.

Identity Spoofing: OIDC-based keyless signing prevents attackers from impersonating legitimate publishers without compromising their identity provider credentials.

Availability Attacks: The distributed nature of the system, combined with content replication across multiple registries, provides resilience against denial-of-service attacks targeting individual nodes.

4.8.5. Access Control

* *Registry-level permissions* control who can publish and retrieve agent records
* *Fine-grained policies* can restrict access to specific agent categories or capability types
* *Audit trails* leverage existing registry logging capabilities to track access patterns

4.8.6. Trust Boundaries

* *Organizational isolation* through separate registries maintains security boundaries
* *Content verification* allows nodes to validate artifact integrity without trusting transport layers
* *Reputation systems* can build on cryptographic proofs of past agent performance

4.9. Performance Optimizations

The architecture incorporates several optimizations for the specific requirements of agent discovery:

4.9.1. Bandwidth Optimization

* *Incremental updates* use OCI layer semantics to transmit only changed portions of agent records
* *Content compression* reduces storage and transmission costs for large agent definitions
* *Selective replication* based on query patterns minimizes unnecessary data transfer

4.9.2. Scalability Architecture

The system scales horizontally through several mechanisms:

* *Registry sharding* distributes storage load across multiple OCI registry instances

* *Index partitioning* in the DHT allows query load to scale with the number of participating nodes

* *Lazy loading* defers retrieval of detailed agent specifications until actually needed

This architecture provides a robust foundation for a decentralized agent directory that can scale to support the growing ecosystem of AI agents while maintaining the security and reliability requirements of production systems.

5. IANA Considerations

This document has no IANA actions.

6. References

6.1. Normative References

[AGNTCY-OASF]
Community, A., "Open Agent Schema Framework (OASF)", n.d., <https://github.com/agntcy/oasf>.

[OCI.Artifact]
Initiative, O. C., "OCI Artifacts Guidance Specification", n.d., <https://github.com/opencontainers/image-spec/blob/v1.1.1/artifacts-guidance.md>.

[OCI.Distribution]
Initiative, O. C., "OCI Distribution Specification", n.d., <https://github.com/opencontainers/distribution-spec>.

[OCI.Image]
Initiative, O. C., "OCI Image Format Specification", n.d., <https://github.com/opencontainers/image-spec>.

[OCI.Manifest]
Initiative, O. C., "OCI Image Manifest Specification", n.d., <https://github.com/opencontainers/image-spec/blob/v1.1.1/manifest.md>.

[OpenID.Auth]
Foundation, O., "OpenID Authentication 2.0 - Final", n.d., <https://openid.net/specs/openid-authentication-2_0.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/rfc/rfc6920>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

6.2. Informative References

[AGNTCY-ADS] Muscariello, L., Pandey, V., and R. Polic, "The AGNTCY Agent Directory Service: Architecture and Implementation", 2025, <https://arxiv.org/abs/2509.18787>.

[AI-Registry-Evolution] Singh, A., Ehtesham, A., Lambe, M., Grogan, J. J., Singh, A., Kumar, S., Muscariello, L., Pandey, V., Sauvage De Saint Marc, G., Chari, P., and R. Raskar, "Evolution of AI Agent Registry Solutions: Centralized, Enterprise, and Distributed Approaches", 2025, <https://arxiv.org/abs/2508.03095>.

[DHT] Maymounkov, P. and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric", IPTPS ’01 , 2001.

[libp2p-kad-dht] Community, libp2p., "go-libp2p-kad-dht: A Kademlia DHT implementation on go-libp2p", n.d., <https://github.com/libp2p/go-libp2p-kad-dht>.

[Oracle-AgentSpec] Oracle, "Open Agent Spec Documentation", n.d., <https://oracle.github.io/agent-spec/index.html>.

[Sigstore] Foundation, C. N. C., "Sigstore: A New Standard for Signing, Verifying and Protecting Software", n.d., <https://www.sigstore.dev>.

Authors’ Addresses

Luca Muscariello
Cisco
Email: lumuscar@cisco.com

Ramiz Polic
Cisco
Email: rpolic@cisco.com