Trusted Digital Web
Trusted Content Storage Architecture (TCS Stack): Architecture Reference Models (TCS-ARMs)

A Comprehensive Analysis with Assessments and Recommendations

Version 0.95 SNAPSHOT March 3, 2021

Michael Herman
Self-Sovereign Blockchain
Futurist, Architect, and Developer

Trusted Digital Web
Hyperonomy Digital Identity Lab
Parallelspace Corporation

Alberta, Canada

mwherman@parallelspace.net

The publication of this document coincides with recent discussions (January-February 2021) about secure data storage solutions in the Decentralized Identity Foundation (DIF) Secure Data Storage working group (sds-wg) that were taking place during the development of the Confidential Storage specification. This is not a DIF publication, unofficial, official, or otherwise.

Table of Contents

Table of Figures

ABSTRACT

The purpose of this document is to describe a series of architecture reference models (ARMs) to help support decision-making with respect to the Confidential Storage (CS) specification being developed by the Decentralized Identity Foundation (DIF) Secure Data Storage working group (sds-wg).

This is not a DIF publication, unofficial, official, or otherwise.

The current scope of this document includes describing and documenting ARMs for the following:

- TCS Core
    - Layer A Trusted Content Storage Kernel
    - Layer B Trusted Content Storage Service
    - Layer C Authorization Service (omitted in this version of this document)
    - Red Line of Confidentiality/Self-Sovereignty
- TCS Services
    - Layer D Services, which, in turn, is partitioned into:
        - Layer D1 Direct Access Services
        - Layer D2 Indirect Access Services

The intended audience for this whitepaper is a broad range of professionals interested in furthering the application and use of a highly secure, modular, encrypted data storage solution for use in software apps, agents, and services. This includes software architects, application developers, and user experience (UX) specialists; as well as people involved in a broad range of standards efforts related to decentralized identity, verified credentials, and secure storage.

The work documented here was performed under the auspices of the Trusted Digital Web project in the Hyperonomy Digital Identity Lab of Parallelspace Corporation.

CONTEXT

"Sometimes called "reasoning from first principles," the idea is to break down complicated problems into basic elements and then reassemble them from the ground up. It's one of the best ways to learn to think for yourself, unlock your creative potential, and move from linear to non-linear results."

[First Principles: The Building Blocks of True Knowledge(https://fs.blog/2018/04/first-principles/)]

"I think it is most important to reason from first principles rather than by analogy. One of the ways we conduct our lives is we reason by analogy. We do this because something was like something else that was done or it was like what other people were doing. It's mentally easier to reason by analogy rather than from first principles."

[First Principles Method Explained by Elon Musk (https://www.youtube.com/watch?v=NV3sBlRgzTI)]

Preface

This is a *first-principles* software architecture reference model (ARM) whitepaper that documents the comprehensive analysis of the potential architecture variations for the Trusted Content Storage Stack (TCS Stack) including detailed interim as well as final assessments and recommendations.

The intent and purpose of this document has taken several strategic turns during the course of its development. Originally entitled *Secure Data Storage Working Group (sds-wg) Confidential Storage (CS): Functional Architecture Reference Models (CS-FARMs)*, the initial goal was to help evolve the *SDS Layers* diagram in the Confidential Storage 1.0 Specification (https://identity.foundation/confidential-storage/#ecosystem-overview) to make it more useful for discussing the technology and architectural options for supported replication between encrypted data vaults.

This whitepaper evolved quickly over the subsequent weeks from being a few pages in length to more than 200 pages. The whitepaper had forked from its original mission. The purpose of the current version of the whitepaper to fully describe the architecture variations for a new project: Trusted Content Storage Architecture (TCS Stack).

The work documented here was performed under the auspices of the Trusted Digital Web project in the Hyperonomy Digital Identity Lab of Parallelspace Corporation.

NOTE: The publication of this document coincides with recent discussions (January-February 2021) about secure data storage solutions in the Decentralized Identity Foundation (DIF) Secure Data Storage working group (sds-wg) that were taking place during the development of the Confidential Storage specification. This is not a DIF publication, unofficial, official, or otherwise.

Purpose

The purpose of this document is to provide the first complete description of the motivations, key concepts, problem statement, and solution approach for a range of architecture reference models (ARMs) for the highly secure, modular, encrypted data storage problems.

The scope of the document includes the components illustrated in the following sds-wg Secure Data Storage (SDS) Layers block diagram.

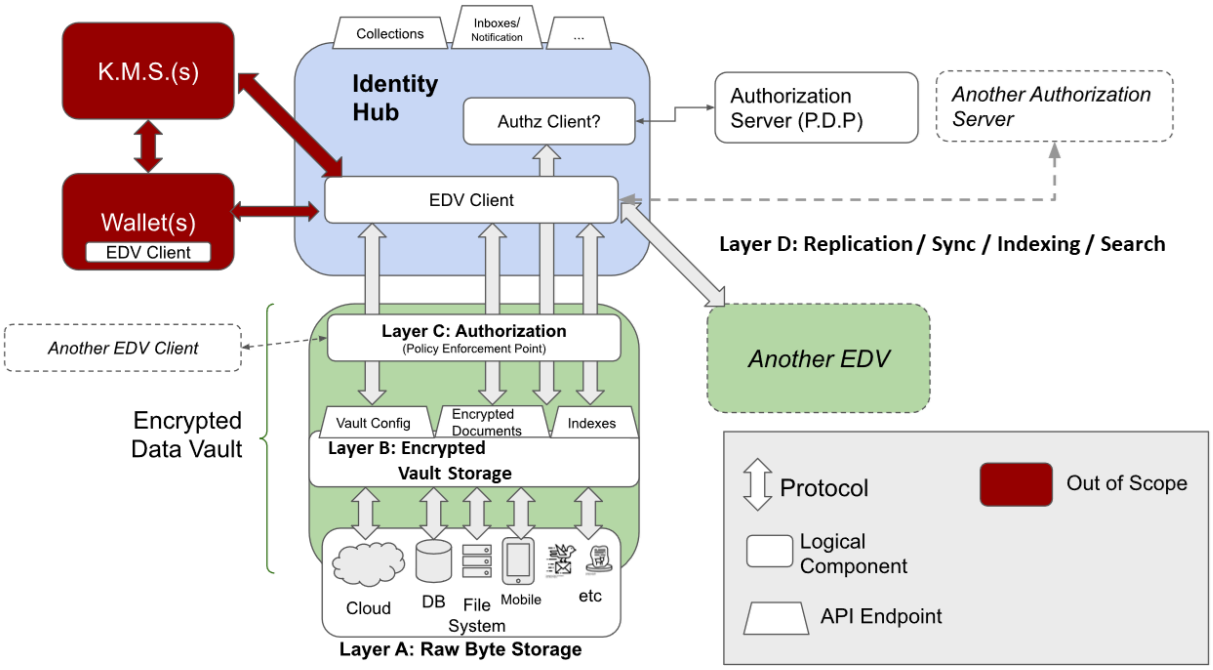


Figure 1. Secure Data Storage (SDS) Layers: Block Diagram

The ARMs described and documented here correspond as closely as possible with the layered component model illustrated in this diagram. These include:

- TCS Core
  - Layer A Trusted Content Storage Kernel
  - Layer B Trusted Content Storage Service
  - Layer C Authorization Service[1]
- TCS Services
  - Layer D TCS Services, which, in turn, is partitioned into:
    - Layer D1 Direct Access Services
    - Layer D2 Indirect Access Services

Intended Audience

The intended audience for this whitepaper is a broad range of professionals interested in furthering the application and use of a highly secure, modular, encrypted data storage solution for use in software apps, agents, and services. This includes software architects, application developers, and user experience

[1] NOTE: TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper.

(UX) specialists; as well as people involved in a broad range of standards efforts related to decentralized identity, verified credentials, and secure storage.

Motivation

The motivation for this document is to bring an architectural approach to develop a range of models (more specifically, architecture reference models) to facilitate current discussions (February 2021) related to the selection, architecture, and design of:

a. Secure content storage model, and
b. Secure content services such as indexing, search, and replication

Organization

The following sections of this whitepaper describe the dimensions for a set of architecture variations to be considered for the Confidential Storage specification including the development of a series of layer-by-layer assessments and recommendations – followed by a final recommendation.

The current section describes the Context including the purpose, intended audience, motivation, and key concepts.

The Problem Statement section more carefully describes the scope of the whitepaper through a series of successive evolutions of the SDS Layers diagram (https://identity.foundation/confidential-storage/#ecosystem-overview). This section also includes an important discussion of the key concepts used throughout this whitepaper.

The Architecture Variations section enumerates several alternative architecture reference models (ARMs) for the following key service layers:

1. TCS Core
    a. Layer A Trusted Content Storage Kernel
        i. Layer A Content Change Detection Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    b. Layer B Trusted Content Storage Service
        i. Layer B Content Change Tracking and Notification Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    c. Layer C Authorization Service[2]
2. TCS Services Layer D Access Models
    a. Layer D1 Direct Access Model
    b. Layer D2 Indirect Access Model
3. TCS Services

[2] NOTE: TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper.

a. Layer D Replication Services
    i. Layer D Replication Outbound Processing Service
    ii. Layer D Replication Package Transfer Service
    iii. Layer D Replication Inbound Processing Service
b. Layer D Indexing & Search Services
    i. Layer D Content Indexing Service
    ii. Layer D Search Query Processing Service

NOTE: As mentioned earlier, TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper. Several sections contain the note: NOTE: This section is a placeholder for future discussion and elicitation of the architecture variations for Layer C Authorization Service.

The Architecture Assessments section discusses the pros-and-cons of each service layer variation against a set of considerations, principles, and requirements based on common expectations of a Trusted Content Storage solution as well as prior art for each service layer.

The Architecture Recommendations section makes specific recommendations for each service layer based on the pros-and-cons assessments from the previous section. Finally, this section makes a final system-level recommendation for all 6 of the key service layers.

The reader can choose to stop reading whenever they feel they have grasped the amount of detail that best fits their goals.

Problem Statement

> *"We store a significant amount of sensitive data online, such as personally identifying information (PII), trade secrets, family pictures, and customer information. The data that we store is often not protected in an appropriate manner.*
>
> *This specification describes a privacy-respecting mechanism for storing, indexing, and retrieving encrypted data at a storage provider. It is often useful when an individual or organization wants to protect data in a way that the storage provider cannot view, analyze, aggregate, or resell the data. This approach also ensures that application data is portable and protected from storage provider data breaches."*
> [Abstract, Confidential Storage 0.1 specification (https://identity.foundation/confidential-storage/)]

This section describes the scope of the whitepaper through a series of successive evolutions of the SDS Layers diagram (https://identity.foundation/confidential-storage/#ecosystem-overview). This section also includes an important discussion of the key concepts used throughout this whitepaper.

Overview

The problem to be solved by this whitepaper is how to best analyze, assess, and recommend a series of architecture reference models (ARMs) for architecting, designing, and implementing a Secure Storage or Confidential Storage solution.

Background

The original impetus for this whitepaper includes:

- The Confidential Storage specification being developed by the Secure Data Storage working group (sds-wg) within the Decentralized Identity Foundation (DIF)
- Several sds-wg conference calls in January and February 2021 and related side conversations

A key input to this whitepaper is the Confidential Storage (CS) Layers Model diagram described below.

Confidential Storage (CS) Layers Model

The original Confidential Storage (CS) Layers Model diagram (aka "SDS Layers" diagram) can be found here: https://identity.foundation/confidential-storage/#ecosystem-overview – a version of which is depicted below.

The first evolution of the SDS Layers diagram is simple: grouping of the service layers into 2 groups:

- CS Core
  - Layer A Raw Byte Storage
  - Layer B Encrypted Vault Storage
  - Layer C Authorization
- CS Services
  - Layer D Replication, Sync, Indexing, and Search Services

This version is depicted below.

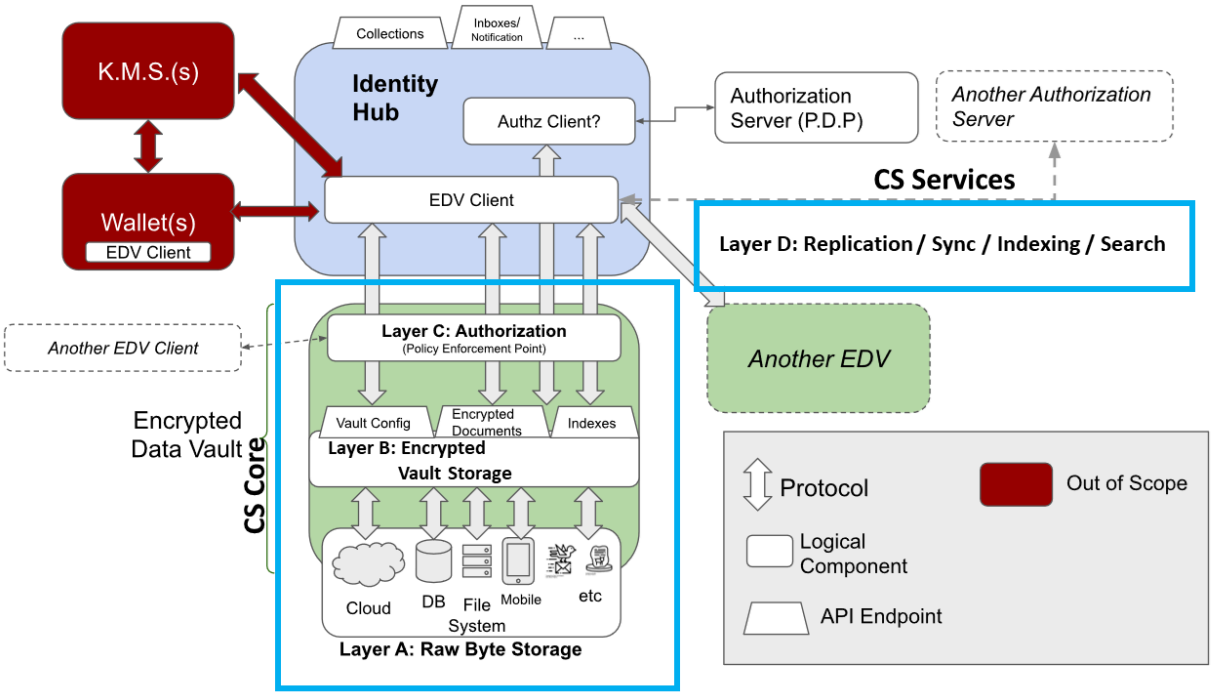


Figure 2. Confidential Storage (CS) Layers Model: CS Core and CS Services

The complete Confidential Storage (CS) Layers Model is depicted below.

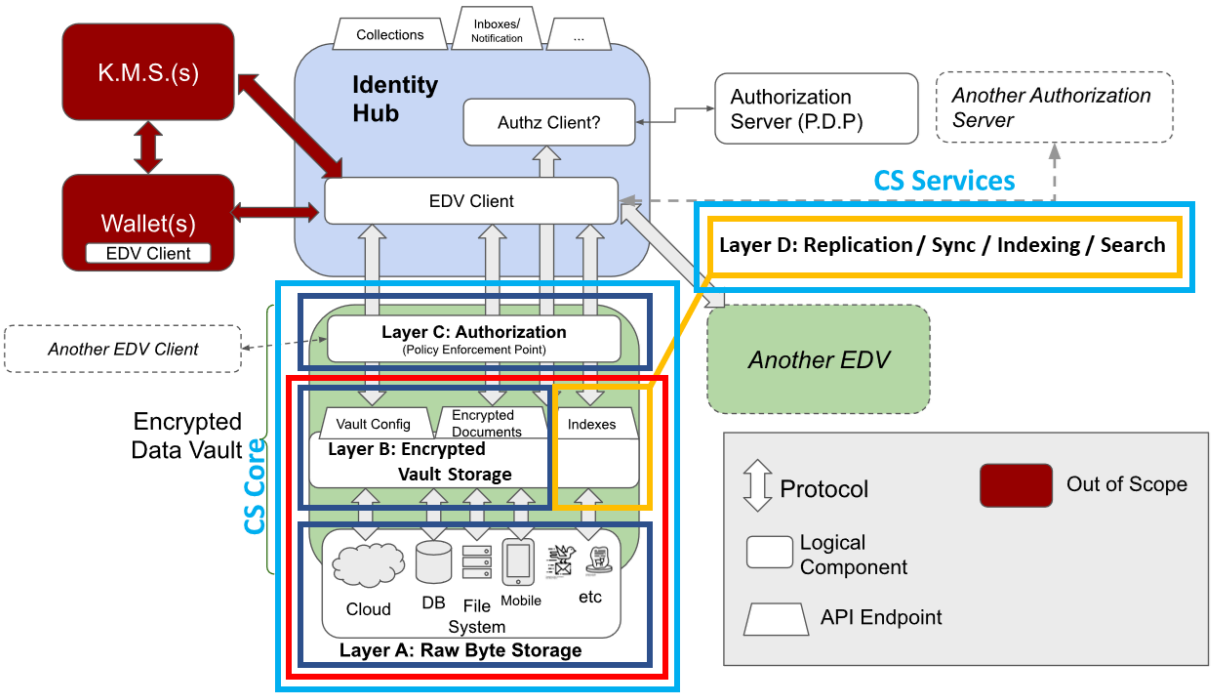


Figure 3. Confidential Storage (CS) Layers Model (CS Core and CS Services)

CS Core: Layers A and B: Trusted Content Storage

Layers A and B, together, are responsible for managing and providing access to Trusted Content Storage. Layer A represents the logical containers where secure content is stored. Layer B represents the service or API that provides access to the content in the secure content storage containers.

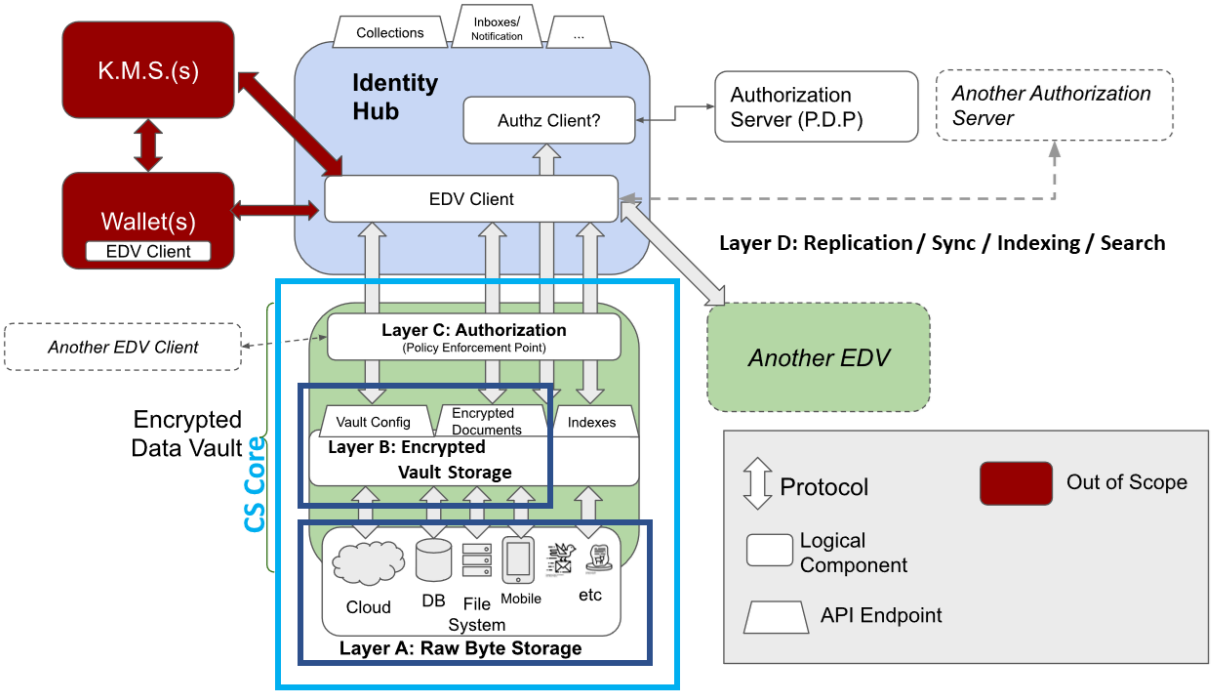


Figure 4. CS Core: Layers A and B

CS Core: Layer C Authorization Service

While Layer C Authorization Service is depicted in this section as well as in the evolution of the SDS Layers diagram, further discussion of Layer C and the Authorization Service is deemed to be out-of-scope for this whitepaper.

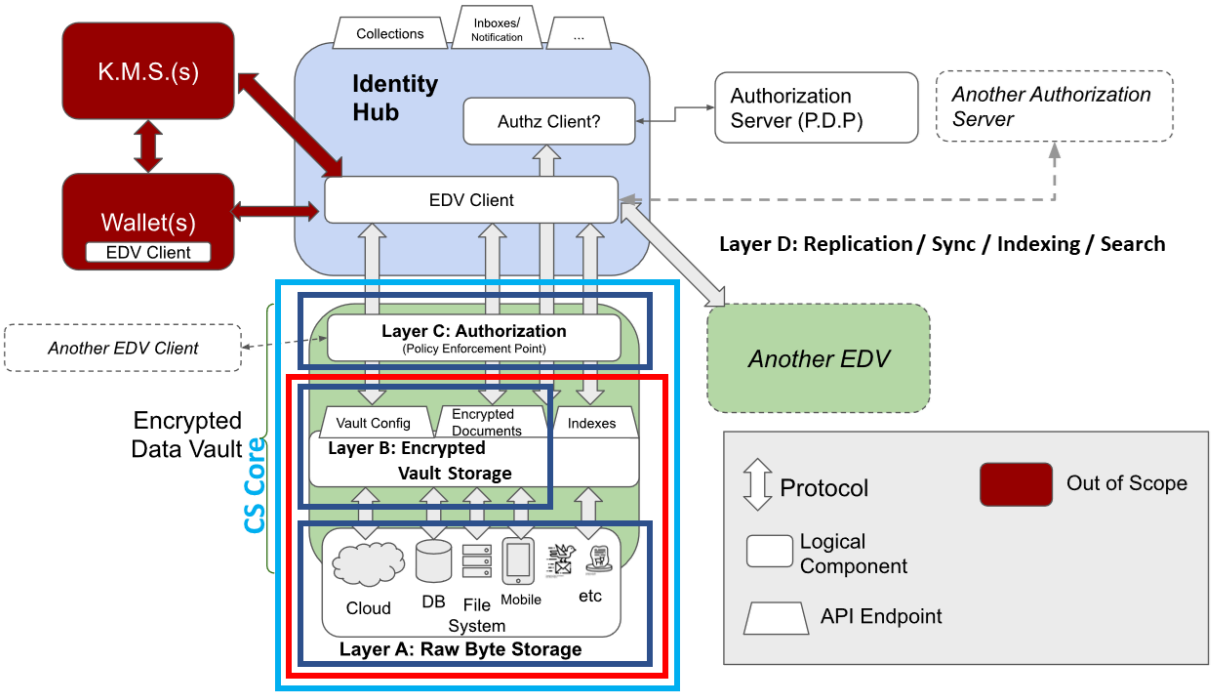


Figure 5. CS Core: Layer C Authorization Service

CS Core: Red Line of Confidentiality/Self-Sovereignty

The Red Line of Confidentiality/Self-Sovereignty (RLOCSS - pronounced r-locks) is a key concept used through-out this whitepaper. During the development of these architecture variation diagrams, the author found it all too easy to simply connect different services directly to the resources in secure content storage that a particular service requires access to (e.g. secure storage configuration information, the live stored content, various indices, and service configuration information, etc.).

A visual aid was needed to ensure no service outside CS Core attempted to access a secure content storage resource “from the side”. Hence, the Red Line of Confidentiality/Self-Sovereignty. The term was first used when creating the conceptual architecture for the “HPEC” Trusted Digital Assistant app in January 2021. For more information, see APPENDIX C – Trusted Digital Assistant: A Conceptual Design on page .

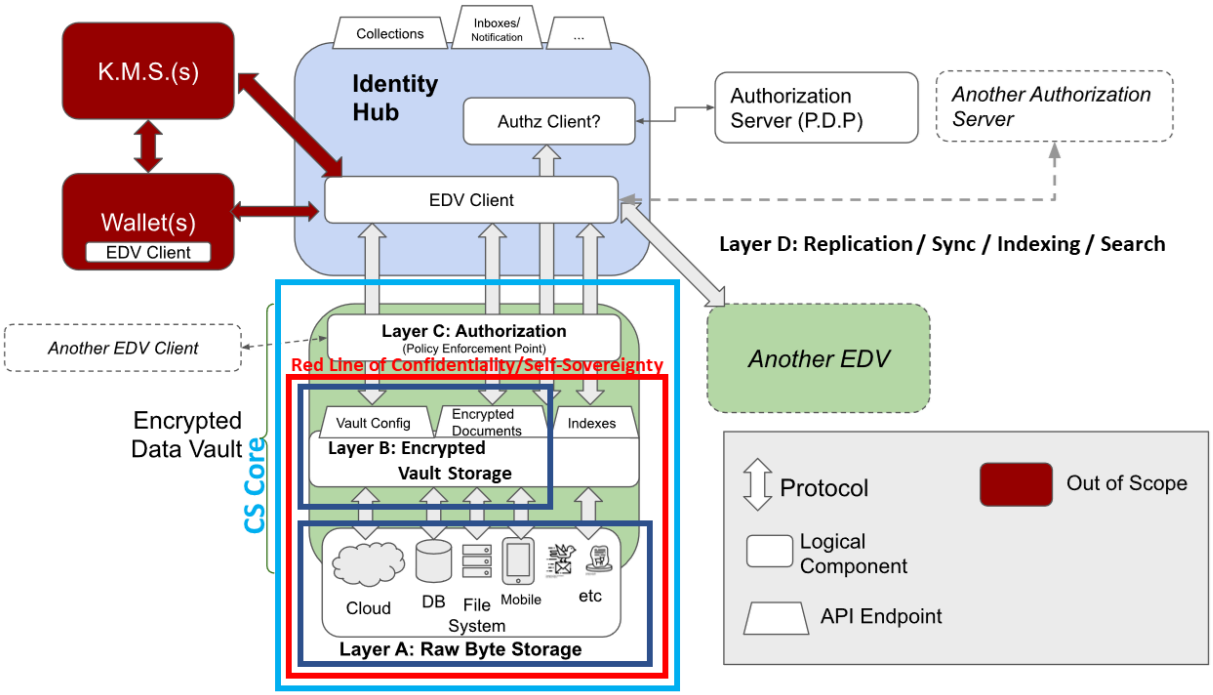


Figure 6. CS Core: Red Line of Confidentiality/Self-Sovereignty

NOTE: It can be debated whether Layer C Authorization Service should be inside the Red Line of Confidentiality/Self-Sovereignty. Currently, Layer C lies outside the Red Line because:

- *Layer C Authorization Service is deemed out of scope for this version of the whitepaper*
- *It is envisioned that Layers A and B can be designed and implemented as a standalone secure content storage technology; that is, developed independently or to the exclusion of a specific Authorization Service.*

CS Services: Layer D Services Model

Layer D CS Services is where "all the action is" …quite literally. Layer D CS Services is used to group and logically separate the valued added services like replication and indexing & search from Layers A, B, & C in CS Core. Layer D CS Services is depicted in the next evolution of the SDS Layers diagram (see below).

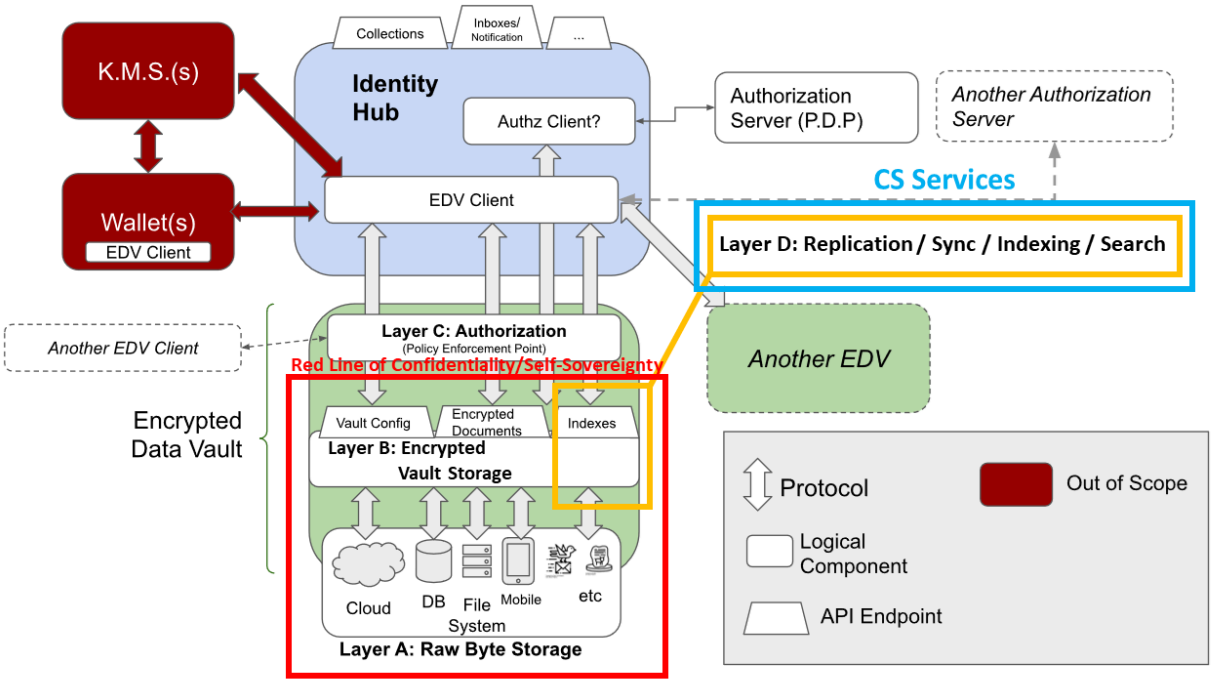


Figure 7. CS Services: Layer D Replication, Sync, Indexing & Search

Confidential Storage (CS) Abstract Model

The Confidential Storage (CS) Abstract Model (illustrated below) is a visual simplification of the Confidential Storage (CS) Layers Model.

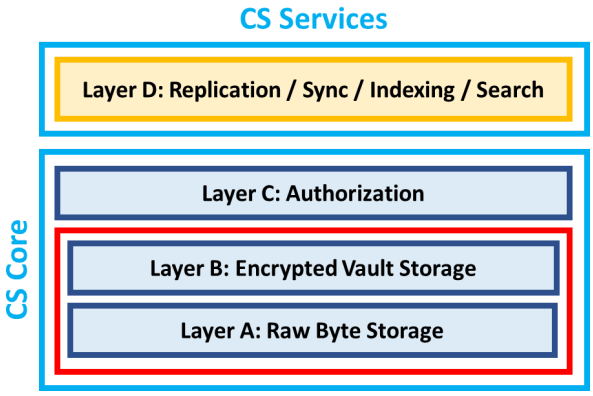


Figure 8. Confidential Storage (CS) Abstract Model

The layers in the Confidential Storage (CS) Abstract Model correspond one-to-one with the layers in the Confidential Storage (CS) Layers Model.

Key Concepts

ArchiMate Modeling Notation

> *The ArchiMate® modeling language is an open and independent Enterprise Architecture standard that supports the description, analysis, and visualization of architecture within and across business domains. ArchiMate is one of the open standards hosted by The Open Group® and is fully aligned with TOGAF®. ArchiMate aids stakeholders in assessing the impact of design choices and changes.*
>
> [Archi open source modeler website (https://www.archimatetool.com/)]

The ArchiMate modeling notation is defined by the Open Group's ArchiMate Specification (current version 3.1). Quoting from https://www.opengroup.org/archimate-home,

> *The ArchiMate Specification, a standard of The Open Group, is an open and independent modeling language for Enterprise Architecture that is supported by different tool vendors and consulting firms. The ArchiMate Specification provides a common language and elements to enable enterprise, business, solution, and technology architects, business analysts and modelers, and software engineers to describe, analyze, and visualize the relationships among business domains in an unambiguous way.*

A partial set of ArchiMate symbols for both entities and relationships appear in the figure below. In this architecture variations presented in this whitepaper, only about 6 of these symbols (mostly from the Application Architecture domain) are used.

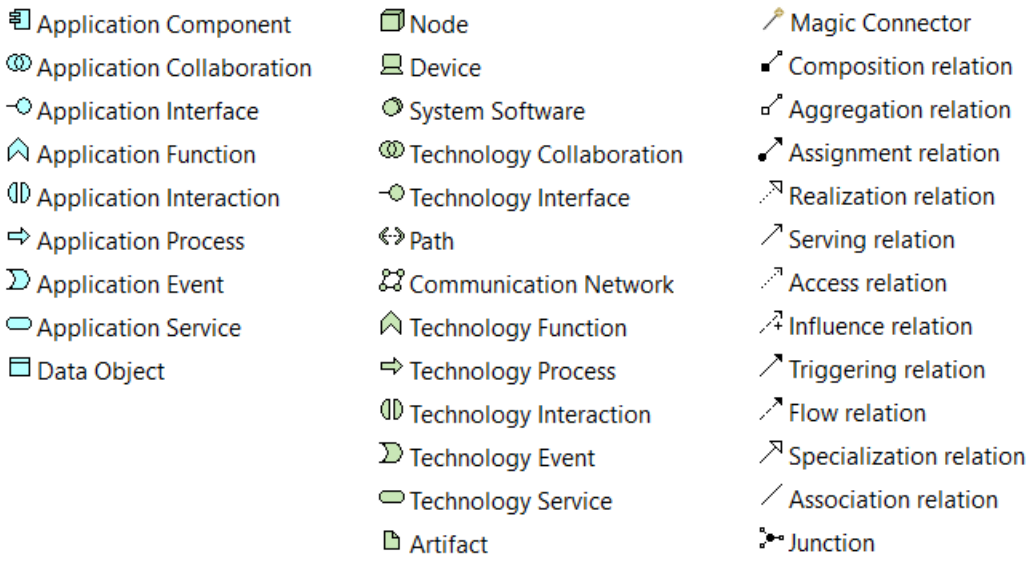


Figure 9. ArchiMate Symbols: (a) Application, (b) Technology, & (c) Relationships

The list of symbols used in this whitepaper includes:

- Technology domain
  - Artifact (used in Layer A Trusted Content Storage Kernel)
- Application domain
  - Application Component (software implementation)

- Application Service (public capabilities defined by an API definition)
- Application Interface (public API definition)
- Application Function (component capabilities)
- Application Event (content change detection trigger)

The following figure is an illustration of a repeating pattern of 5 of these ArchiMate symbols being used to define, implement, and expose a software service.

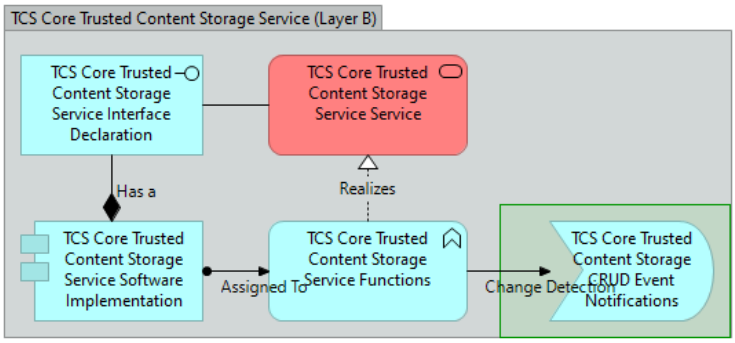


Figure 10. TCS-ARMs ArchiMate Sample

The creation of architecture diagrams using the ArchiMate notation is greatly simplified using Archi, the open-source modeling tool that is freely downloadable from https://www.archimatetool.com/.

Archi Open Source Modeler for ArchiMate

The Archi® modeling toolkit is targeted toward all levels of Enterprise Architects and Modelers. It provides a low cost to entry solution to users who may be making their first steps in the ArchiMate modeling language, or who are looking for an open-source, cross-platform ArchiMate modeling tool for their company or institution and wish to engage with the language within a TOGAF® or other Enterprise Architecture framework.

[Archi open source modeler website (https://www.archimatetool.com/)]

Architecture Reference Model (ARM)

Architectural Pattern

An architectural pattern is a description of element and relation types together with a set of constraints on how they may be used. A pattern can be thought of as a set of constraints on an architecture-on the element types and their patterns of interaction-and these constraints define a set or family of architectures that satisfy them. For example, client-server is a common architectural pattern. Client and server are two element types, and their coordination is described in terms of the protocol that the server uses to communicate with each of its clients. Use of the term client-server implies only that multiple clients exist; the clients themselves are not identified, and there is no discussion of what functionality, other than the implementation of the protocols, has been assigned to any of the clients or to the server. Countless architectures are of the client-server pattern under this (informal) definition, but they are different from each other. An architectural pattern is not an

architecture, then, but it still conveys a useful image of the system-it imposes useful constraints on the architecture and, in turn, on the system.

One of the most useful aspects of patterns is that they exhibit known quality attributes. This is why the architect chooses a particular pattern and not one at random. Some patterns represent known solutions to performance problems, others lend themselves well to high-security systems, still, others have been used successfully in high-availability systems. Choosing an architectural pattern is often the architect's first major design choice.

Reference Model

A reference model is a division of functionality together with data flow between the pieces. A reference model is a standard decomposition of a known problem into parts that cooperatively solve the problem. Arising from experience, reference models are a characteristic of mature domains. Can you name the standard parts of a compiler or a database management system? Can you explain in broad terms how the parts work together to accomplish their collective purpose? If so, it is because you have been taught a reference model of these applications.

Reference Architecture

A reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flow between them. Whereas a reference model divides the functionality, a reference architecture is the mapping of that functionality onto a system decomposition. The mapping may be, but by no means necessarily is, one-to-one. A software element may implement a part of a function or several functions.

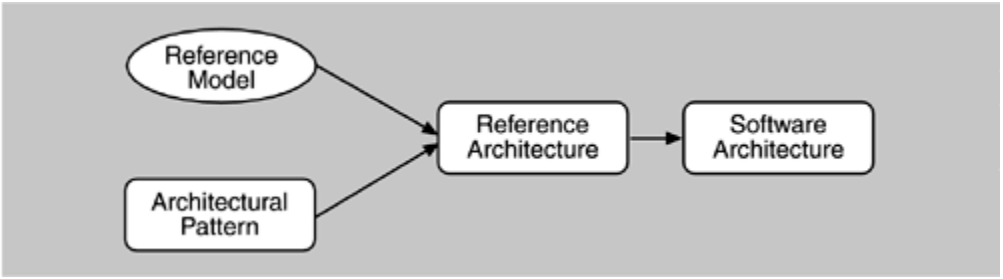


Figure 11. Reference models, architectural patterns, reference architectures, and software architectures.

[Software Architecture in Practice, Second Edition (http://www.ece.ubc.ca/~matei/EECE417/BASS/ch02lev1sec3.html)]

The arrows indicate that subsequent concepts contain more design elements.

Architecture Reference Model (ARM)

In this whitepaper, following the above 3 definitions, an architecture reference model (ARM) is akin to the first definition, an *architectural pattern*.

An architecture reference model (ARM) is an ARM that is focused on functionality; that is, the services and capabilities delivered by a solution that conforms to the ARM (and underlying ARM or architecture pattern).

Trusted Content Storage (TCS): Architecture Reference Models (TCS-ARMs)

This section documents the evolution of the Confidential Storage (CS) Layers model into the Trusted Content Storage (TCS) Stack model.

TCS Stack

For the purposes of this whitepaper, more specific terminology was chosen for naming each of the 4 layers in the TCS-ARMs models used throughout the rest of this document. The mappings from the Confidential Storage (CS) Layers Model to the TCS Stack are illustrated below.

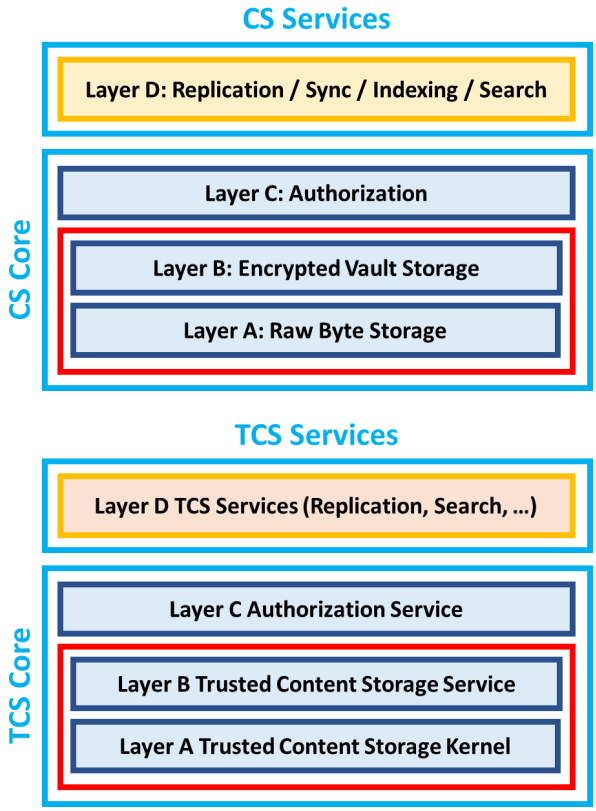


Figure 12. (a) Confidential Storage (CS) Abstract Model mapped to (b) TCS Stack

The TCS Stack appears below (standalone) – the result of the above direct mapping from the Confidential Storage (CS) Abstract Model.

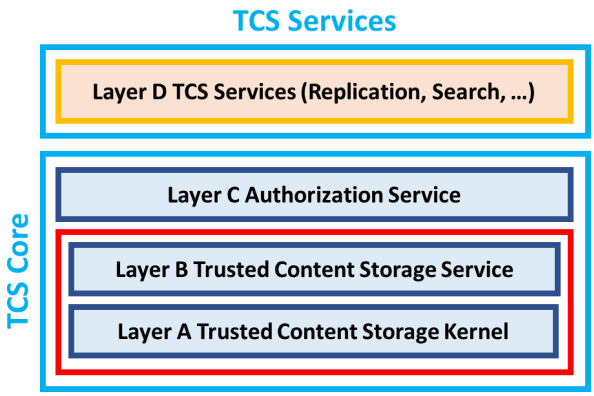


Figure 13. TCS Stack

Detailed TCS Stack

The Detailed TCS Stack Diagram expands Layer A Trusted Content Storage Kernel and Layer B Trusted Content Storage Service layers to expose more of each layer's subcomponents (sub-elements).

In Layer A Trusted Content Storage Kernel, the primary subcomponent is the EDV Microkernel.

In Layer B Trusted Content Storage Service, the primary subcomponents are: (a) the protocol handlers (endpoints), and (b) the underlying services needed to integrate them with the EDV Microkernel.

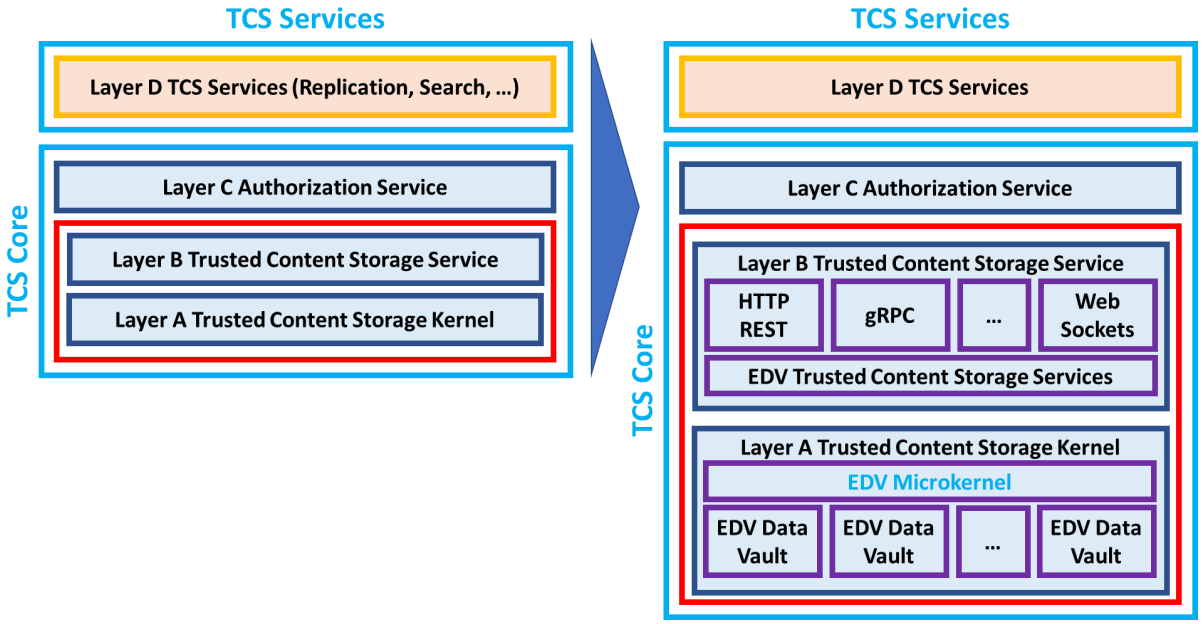


Figure 14. (a) TCS Stack mapped to (b) Detailed TCS Stack

TCS Extended Stack

For reasons that will become apparent in the rest of this document the above TCS Stack is not sufficient; for example, when a process such as Content Indexing Service which reads the Live Content from Trusted Content Storage and updates the Content Index (also in Trusted Content Storage), it might make sense for Content Indexing Service to run inside the Red Line of Confidentiality/Self-Sovereignty for performance as well as security reasons.

The TCS Extended Stack addresses these types of scenarios by allowing for the partitioning of TCS Services Layer D Services into 2 sub-layers:

- Layer D1 Direct Access Model Services
- Layer D2 Indirect Access Model Services

This further evolution of the TCS Stack into the TCS Extended Stack is illustrated below.

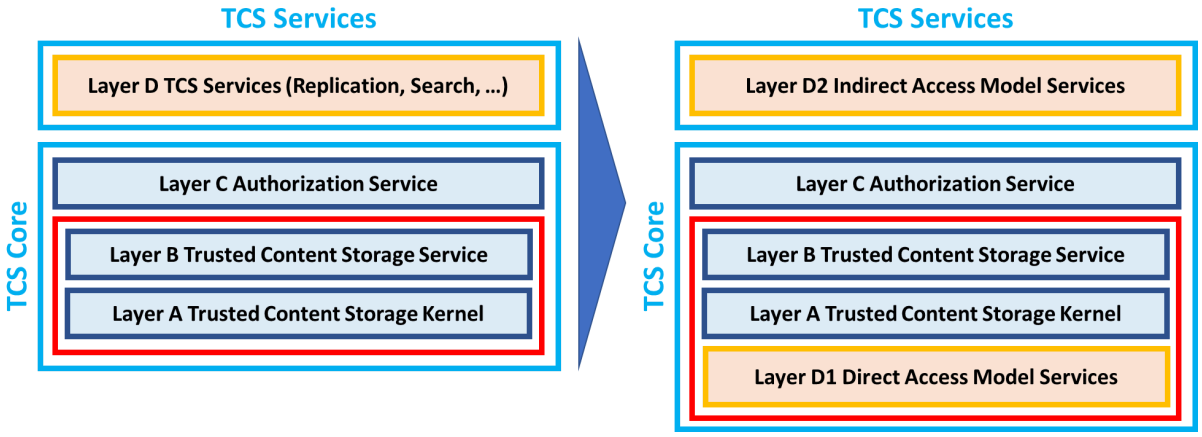


Figure 15. (a) TCS Stack mapped to (b) TCS Extended Stack

The TCS Extended Stack appears below (standalone) – the result of the above direct mapping from the TCS Stack.

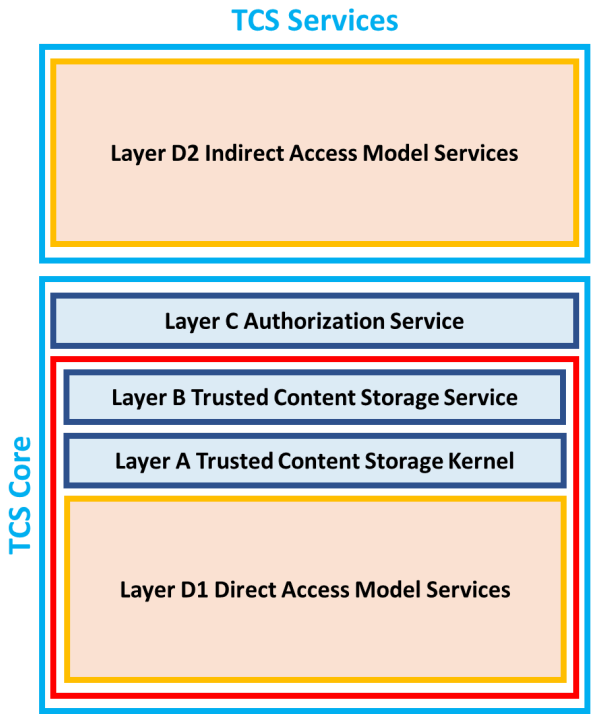


Figure 16. TCS Extended Stack[3,4]

Layer D CS Services is further partitioned into 2 sublayers:

- Layer D1 Direct Access Services – A subset of TCS Services Layer D Services that, effectively, run inside the Red Line of Confidentiality/Self-Sovereign, or otherwise, have access to the Layer A storage via the Layer B API (indirect access). This, potentially, includes optimized direct access to Layer A storage (direct access).
- Layer D2 Indirect Access Services – A subset of TCS Services Layer D Services that run outside the Red Line of Confidentiality/Self-Sovereign, or otherwise, never have direct access to the Layer A storage other than through the Layer B API (indirect access).

From an architecture variations perspective, Layer D1 Direct Access Services are those services that might run in the Red Line of Confidentiality/Self-Sovereign. For example, a variation of an Indexing service that processes Live Content in Trusted Content Storage and writes to a Content Index also in

[3] In Layer D1 Direct Access Model Services, for example, a TCS Service can, effectively, run inside the Red Line of Confidentiality/Self-Sovereign, or otherwise, have access to the Layer A Trusted Content Storage via the Layer B Trusted Content Storage Service (API) (indirect access). This, potentially, includes optimized direct access to Layer A Trusted Content Storage (direct access). This architecture variation is discussed in the remainder of the document.

[4] In Layer D2 Indirect Access Model Services, for example, the Index Manager (Indexer), can run outside the Red Line of Confidentiality/Self-Sovereign, or otherwise, never have direct access to the Layer A Trusted Content Storage other than through the Layer B Trusted Content Storage Service (API) (indirect access). This architecture variation is discussed in the remainder of the document.

Trusted Content Storage might best be implemented inside the Red Line of Confidentiality/Self-Sovereign (as part of TCS Core) for both security as well as performance reasons.

This “D1” variation of an Indexing service processes a sensitive input resource (the Live Content in Trusted Content Storage) and updates a sensitive output resource (the Content Index also in Trusted Content Storage). This type of service is categorized as a “D1” service because runs closest to the CS Core (in fact, the D1 Direct Access Service Model always runs inside CS Core).

Layer D2 Indirect Access Services run outside the Red Line of Confidentiality/Self-Sovereign and outside CS Core. “D2” services can only access resources (content) in Trusted Content Storage via the Layer B Trusted Content Storage Service API.

The key thing to note is the Layer D1 Direct Access Model Services live inside the Red Line of Confidentiality/Self-Sovereignty and hence, from a security and deployment perspective are packaged as part of TCS Core.

Detailed TCS Extended Stack

Combining the above concepts yields the Detailed TCS Extended Stack illustrated below.

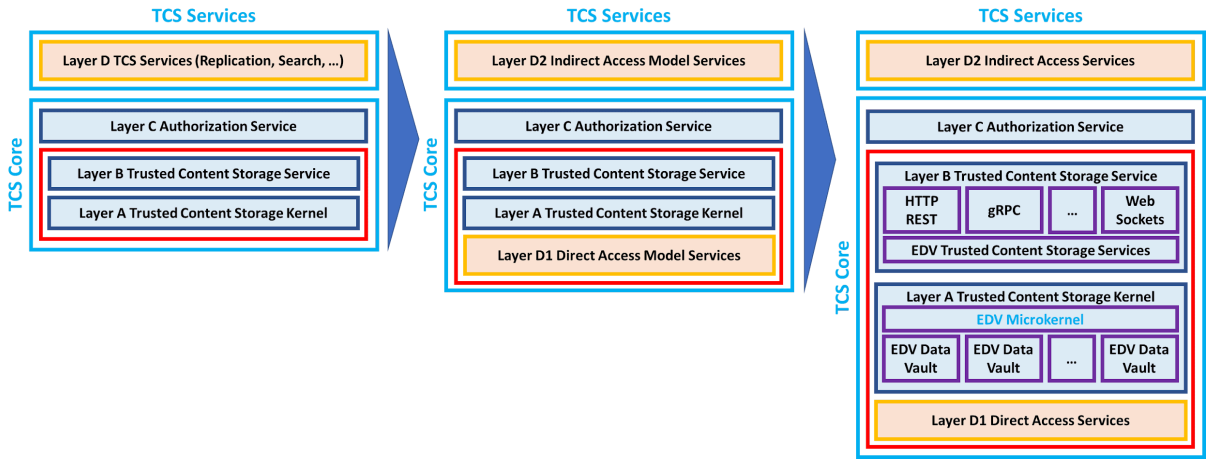


Figure 17. (a) TCS Stack mapped to (b) TCS Extended Stack mapped to (c) Detailed TCS Extended Stack

Content Change Detection, Tracking, and Notification

Within TCS Core Layer B Trusted Content Storage Service – Content Change Detection/Tracking/Notification Models, there are 3 Content Change Detection/Tracking/Notification Models:

- CRUD Events
- Update Sequence Numbers (USNs)
- Crawler Events

Content Change Detection

Content Change Detection’s responsibility is to detect or otherwise sense changes being made to any content being made in an EDV Data Vault. This includes changes to configuration data, indices, etc. in addition to everyday content resources stored in a data vault.

In the specific context of the TCS Stack, this also includes Read operations as well as both successful operation attempts and unsuccessful operation attempts.

In the TCS Stack, the responsibility for Content Change Detection is assigned to the EDV Microkernel in the Layer A Trusted Content Storage Kernel.

Content Change Tracking

Once a change has been detected, Content Change Tracking is responsible for tracking the change by either: (a) raising or triggering an event, (b) using an update sequence number (USN) to track changes on a resource-by-resource basis, or (c) recording the event in a secondary container in the data vault.

In the TCS Stack, the responsibility for Content Change Tracking is assigned to the Layer B Trusted Content Storage Service.

Content Change Notification

Once a change has been detected and tracked, Content Notification is responsible for notifying TCS Services Layer D Services so that the change can be acted upon. These may be Layer D1 Services with direct access to the data vault via the EDV Microkernel or Layer D2 Services with indirect access to the data vault via the Layer B Trusted Content Storage Service Service.

In the TCS Stack, the responsibility for Content Change Notification is assigned to the Layer B Trusted Content Storage Service.

TCS Stack Architecture Variations

With this section, the document is getting into the primary purpose of this whitepaper: the illimitation and illustration of the TCS-ARMs architecture variations. The primary dimensions of the architecture variations are:

1. TCS Core
    a. Layer A Trusted Content Storage Kernel
        i. Layer A Content Change Detection Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    b. Layer B Trusted Content Storage Service
        i. Layer B Content Change Tracking and Notification Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    c. Layer C Authorization Service[5]

[5] NOTE: TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper.

2. TCS Services Layer D Access Models
    a. Layer D1 Direct Access Model
    b. Layer D2 Indirect Access Model
3. TCS Services
    a. Layer D Replication Services
        i. Layer D Replication Outbound Processing Service
        ii. Layer D Replication Package Transfer Service
        iii. Layer D Replication Inbound Processing Service
    b. Layer D Indexing & Search Services
        i. Layer D Content Indexing Service
        ii. Layer D Search Query Processing Service

Within TCS Core Layer B Trusted Content Storage Service – Content Change Detection/Tracking/Notification Models, there are 3 Content Change Detection/Tracking/Notification Models:

1. CRUD Events
2. Update Sequence Numbers (USNs)
3. Crawler Events

These are detailed in the next section Architecture Variations.

In addition, for each of the primary services in TCS Services Layer D Services (Replication Outbound Processing Service, Replication Inbound Processing Service, and Content Indexing Service), there are 2 service models:

- Layer D1 Direct Access Service Model Services
- Layer D2 Indirect Access Service Model Services

These are illustrated below and are similarly detailed in the next section Architecture Variations.

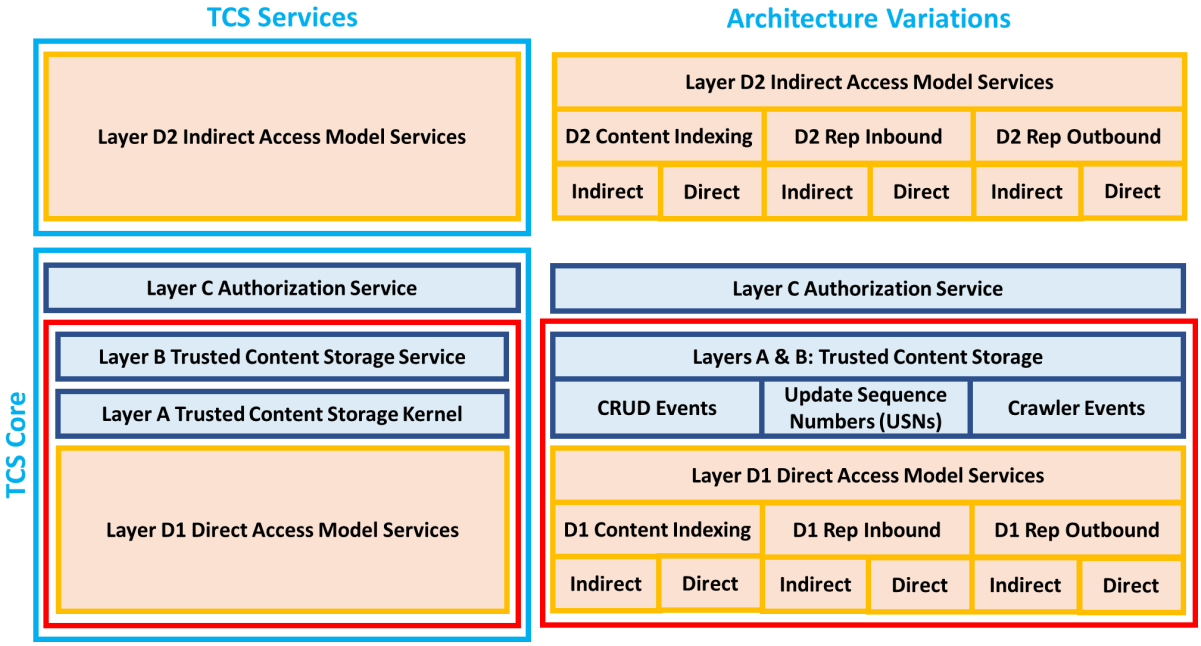


Figure 18. TCS Stack: Architecture Variations

The TCS Stack Architecture Variations Model appears below (standalone) – an evolution of the TCS Extended Stack.

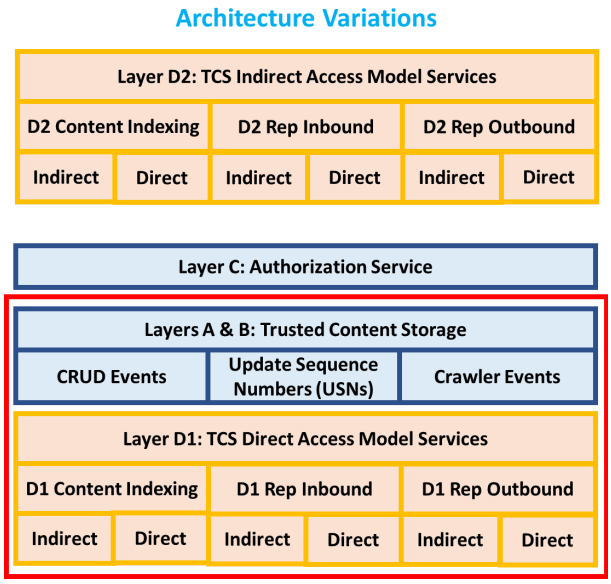


Figure 19. TCS Stack: Architecture Variations Model

Generic Replication Pipeline

A generic Replication Pipeline is illustrated below.

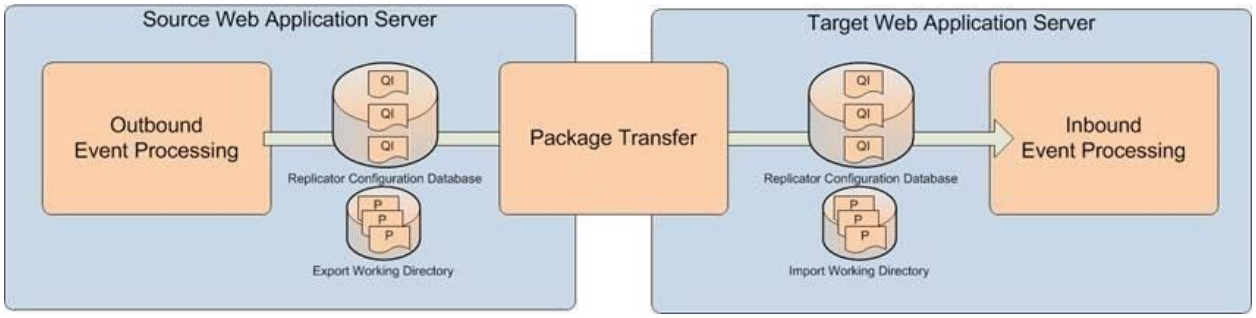


Figure 20. Generic Replication Pipeline (Syntergy Replicator for SharePoint)

Outbound Processing Service

Outbound Processing Service is responsible for capturing and recording Replication Events that occur in the Source Repository. Outbound Processing Service is controlled by Replication Maps which determine what Events need to be captured, packaged, and transferred to the Target Repository. Groups of Replication Events are packaged into two types of messages or objects: Queued Items and Replication Packages.

Queued Item

A Queued Item is a unit of work to be transferred to a Target Repository for remote execution. The Replicator Web Service on a Target Repository is called to push a Queued Item from the Source Repository to a Target Repository.

Replication Package

A Replication Package is a collection of one or more Replication Events plus data about the changed information that is packaged in a format specific to the Replication Transport being used. When an Event is being processed, Outbound Processing Service calls the Source repository object model to extract the changed information from the Source Repository.

Package Transfer Service

The Package Transfer Service activity is responsible for the transfer of Queued Items and Replication Packages (Packages) from the Source Repository to the Target Repository. Package Transfer is the process that sits between Outbound Processing Service (on the Source Repository) and Inbound Processing Service (on the Target Repositories).

Inbound Processing Service

Inbound Processing Service is responsible for processing the Queued Items and Packages received and accepted by the Target Repository and applying them to the repository. The Queued Items and Packages are applied to the Target Repository content base by calling the Target Repository object model.

Generic Content Change Detection, Tracking, and Notification Model

A generic Content Change Detection, Tracking, and Notification Model is illustrated below.

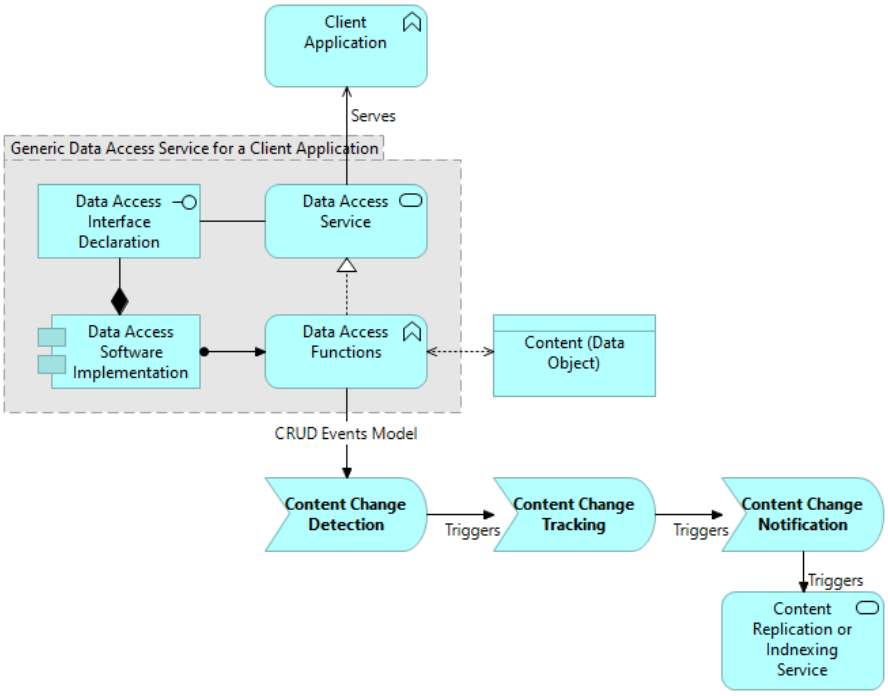


Figure 21. Generic Content Change Detection, Tracking, and Notification Model

Architecture Variations

With the above background information and key concepts under our belt, the TCS Stack architecture variations can be illustrated and described. The following 3 versions of the TCS Stack architecture diagrams will provide the context for describing the architecture variations for each layer in the TCS (Extended) Stack.

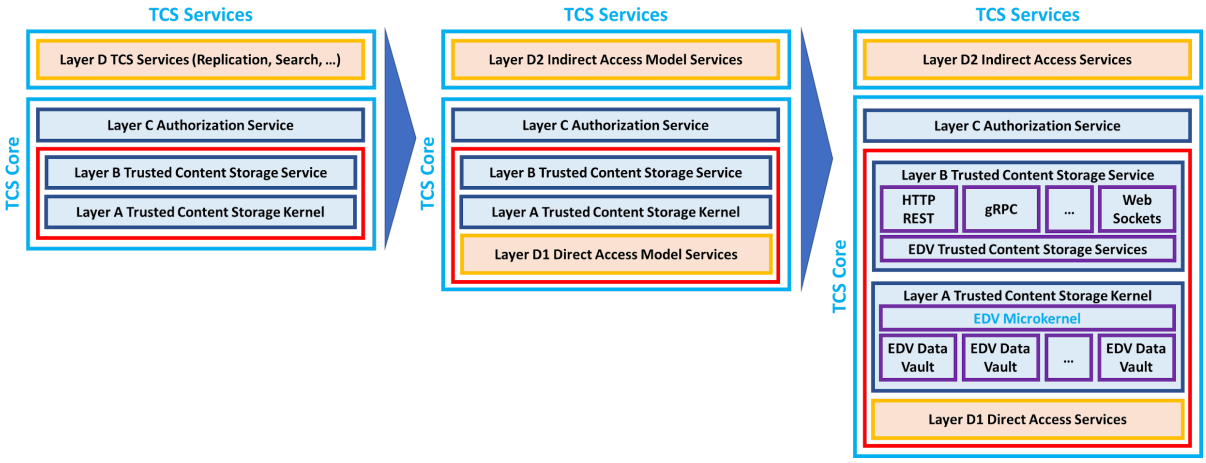


Figure 22. (a) TCS Stack, (b) TCS Extended Stack, (c) Detailed TCS Extended Stack

The same organization will also be used in the Architecture Assessments and Architecture Recommendations sections of this document.

The rest of this section enumerates several alternative architecture reference models (ARMs) using the following dimensions:

1. TCS Core
    a. Layer A Trusted Content Storage Kernel
        i. Layer A Content Change Detection Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    b. Layer B Trusted Content Storage Service
        i. Layer B Content Change Tracking and Notification Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    c. Layer C Authorization Service[6]

[6] NOTE: TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper.

2. TCS Services Layer D Access Models
    a. Layer D1 Direct Access Model
    b. Layer D2 Indirect Access Model
3. TCS Services
    a. Layer D Replication Services
        i. Layer D Replication Outbound Processing Service
        ii. Layer D Replication Package Transfer Service
        iii. Layer D Replication Inbound Processing Service
    b. Layer D Indexing & Search Services
        i. Layer D Content Indexing Service
        ii. Layer D Search Query Processing Service

The TCS Stack architecture variations landscape is depicted in the following diagram.

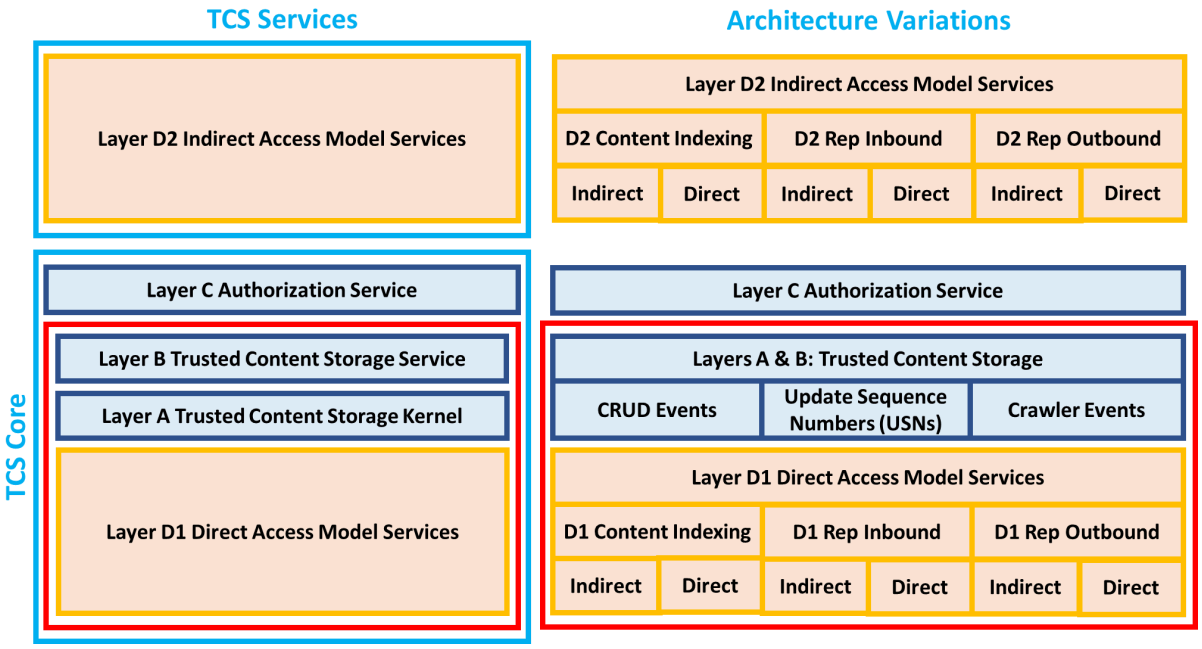


Figure 23. TCS Stack Architecture Variations Landscape

The high-order determinant for classifying the variations includes the following:

- Content Change Detection/Tracking/Detection Models
    - Layer A Content Change Detection Models
    - Layer B Content Change Tracking and Notification Models

The Content Change Detection Model for Layer A Trusted Content Storage Kernel and the Content Change Tracking and Notification Model chosen for Layer B Trusted Content Storage Service are the primary determinates of the architecture variations for the other dimensions.

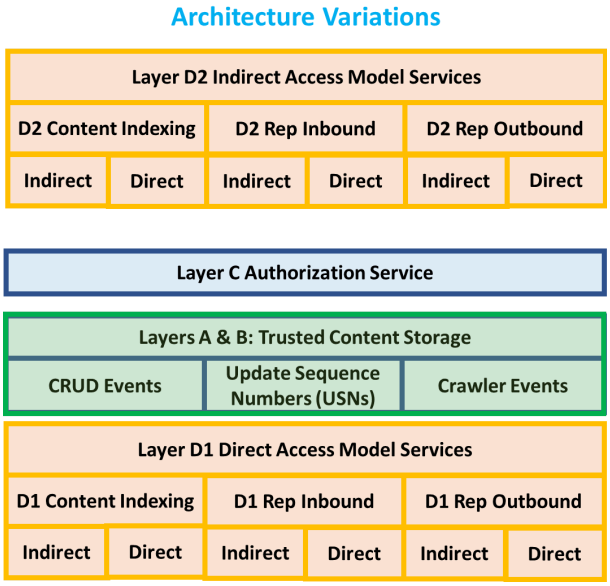


Figure 24. Layer B Content Change Detection/Tracking/Notification

In particular, these primary TCS Services Layer D Services:

- Layer D Replication Outbound Processing Service
- Layer D Replication Package Transfer Service
- Layer D Replication Inbound Processing Service
- Layer D Content Indexing Service

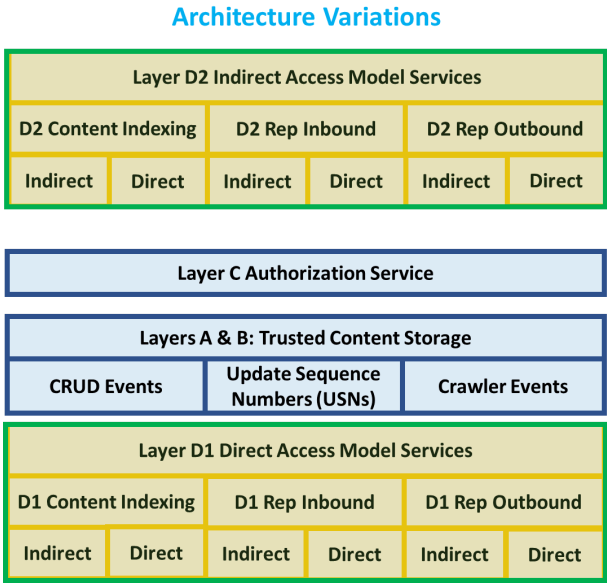


Figure 25. Primary Layer D Services

When the cross-product of the primary TCS Services Layer D Services and the Layer B Content Change Detection/Tracking/Notification variations is performed, a very large total number of possible architecture variations is the result (as depicted below).

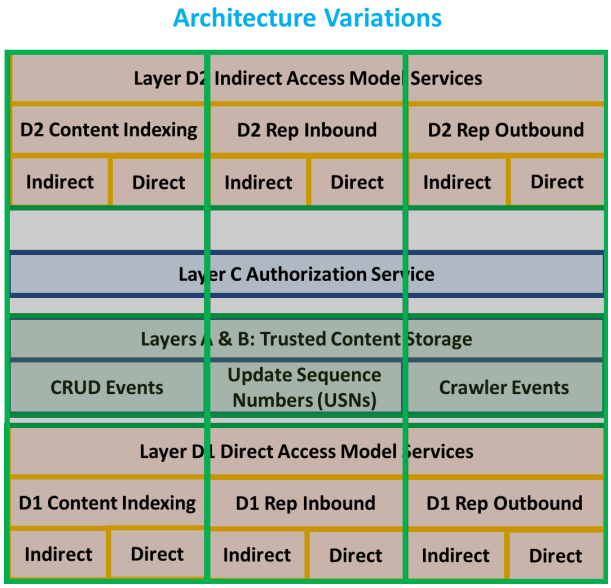


Figure 26. Layer B Content Change Detection/Tracking/Notification Variations X Layer D Services

The remainder of this section on Architecture Variations is framed by the above dimensions.

The next section following Architecture Variations, Architecture Assessments, will evaluate the pros-and-cons of each variation against a set of considerations, principles, and requirements based on common expectations of a Trusted Content Storage solution as well as prior art for each service layer.

TCS Core Layer A Architecture Variations

There are 2 variations for TCS Core Layer A (Trusted Content Storage Kernel):

- EDV Microkernel
- Heterogeneous Multi-vaults/Server Instance Support

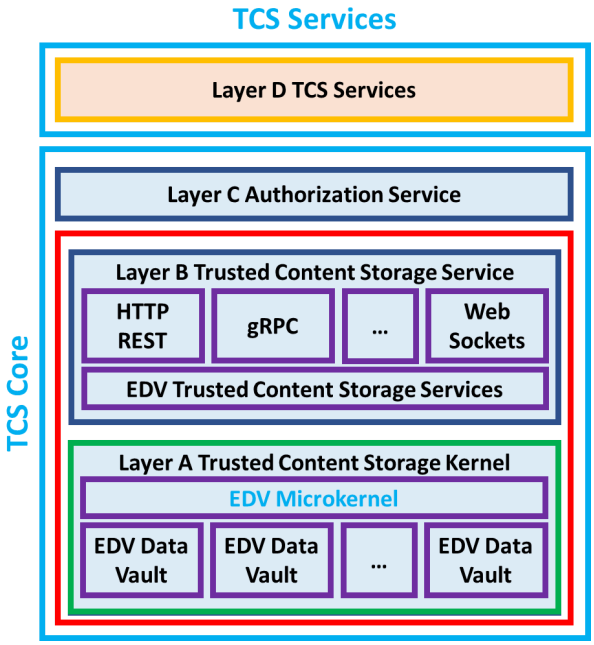


Figure 27. TCS Core Layer A Architecture Variations

Layer A Trusted Content Storage Kernel

The formal name for Layer A is TCS Core Layer A Trusted Content Storage Kernel. Its primary subcomponents are:

- EDV Microkernel
- EDV Data Vaults

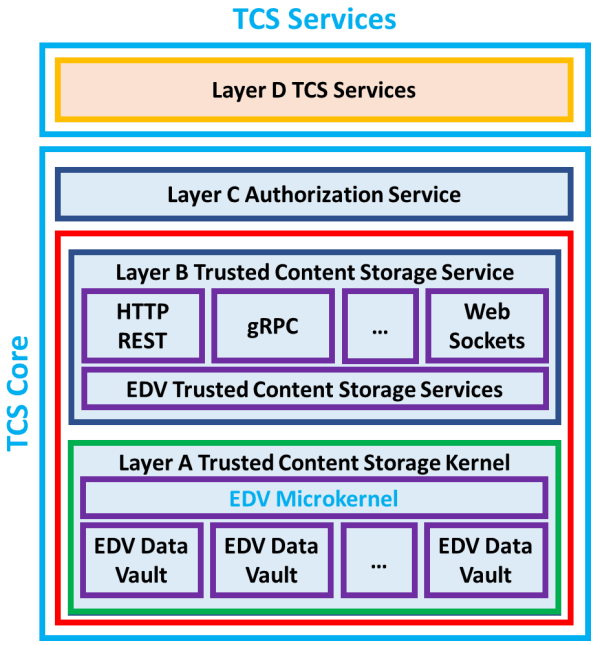


Figure 28. TCS Core Layer A Architecture Variations: EDV Microkernel and EDV Data Vaults

The following ARM for Layer A Trusted Content Storage Kernel is illustrated below. This figure depicts the most basic or minimal resources that are expected to be stored in Layer A containers.

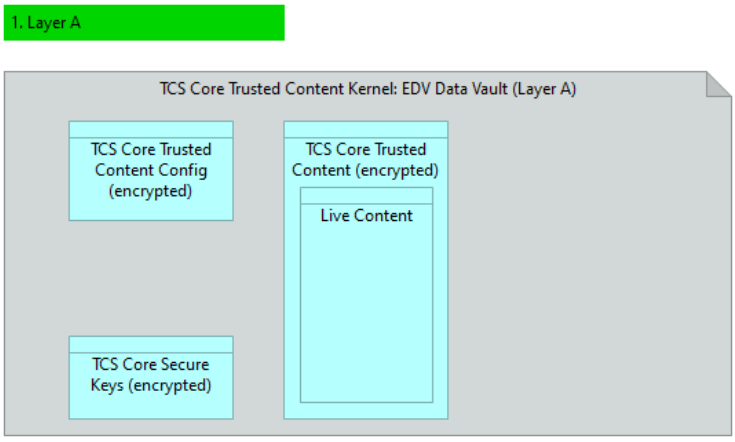


Figure 29. Layer A Trusted Content Storage Kernel [1]

However, concerning the trusted storage requirements of the various TCS Services Layer D Services, additional resources might also reside in Layer A Trusted Content Storage Kernel as illustrated below.

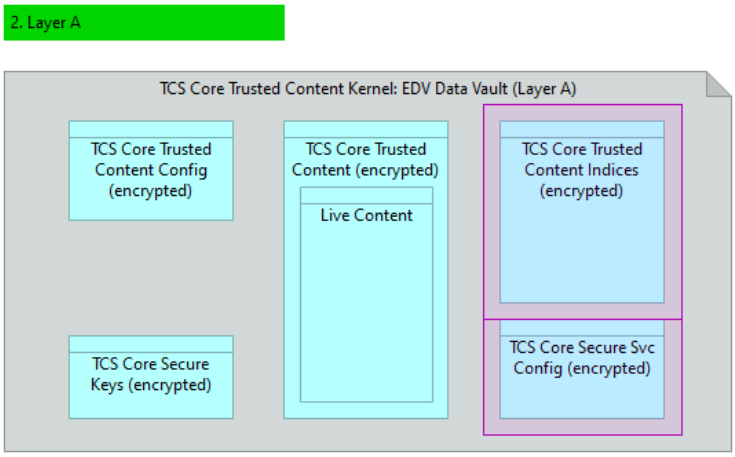


Figure 30. Layer A Trusted Content Storage Kernel: Many Containers [2]

EDV Microkernel Architecture

The EDV Microkernel consists of a low-level API interface and underlying implementation for managing EDV Data Vaults and EDV data content. For simplicity, imagine an EDV Data Vault being a single physical file or block of physical storage on a local device (including deployment into local storage on a server).

The EDV Microkernel interface is a flat "C-callable" interface that provides programmatic access to one or more mounted EDV Data Vaults. The interface is designed to be exposed through a range of different protocol endpoints in Layer B Trusted Content Storage Service.

The EDV Microkernel is capable of managing multiple EDV Data Vaults (and their content) mounted on a single EDV server instance.

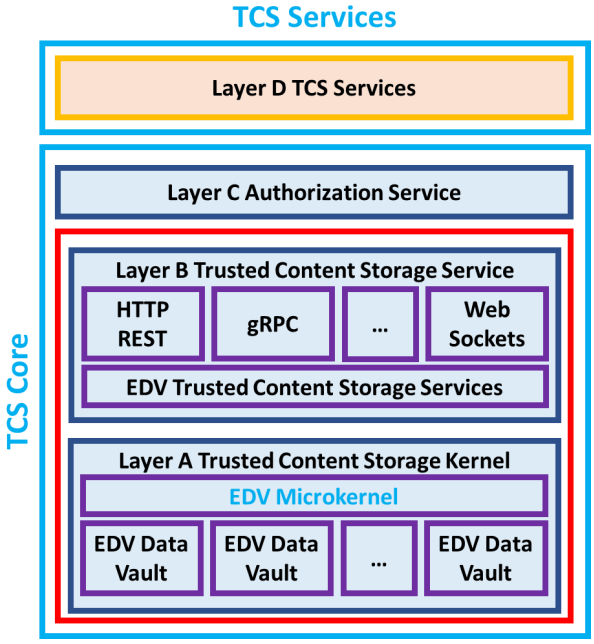


Figure 31. TCS Stack Layer A EDV Microkernel Architecture: Multiple EDV Data Vaults per EDV Server Instance

Layer A Content Change Detection

ISSUE: Content Change Detection, Tracking, and Notifications: Layer A or Layer B or both? At the time of writing, Content Change Detection, Tracking, and Notifications are assigned to Layer B Trusted Content Storage Service and the concept of the EDV Kernel is being newly introduced as a (fixed) variation in Layer A Trusted Content Storage Kernel.

As a result, all of the detailed (ArchiMate) architecture reference model diagrams show that the responsibility for Content Change Detection, Tracking, and Notifications is relegated to Layer B. With the introduction of the EDV Microkernel, at least one of these functions (Detection) should be re-assigned to the EDV Microkernel in Layer A Trusted Content Storage Kernel.

This version of this whitepaper will be completed in this problem left intact. A future version of this document and its diagram will be updated to fix this issue. The text of this document will assume the Content Change Detection is implemented in the EDV Microkernel.

The scope of the architecture variations for Layer A Content Detection is illustrated below.

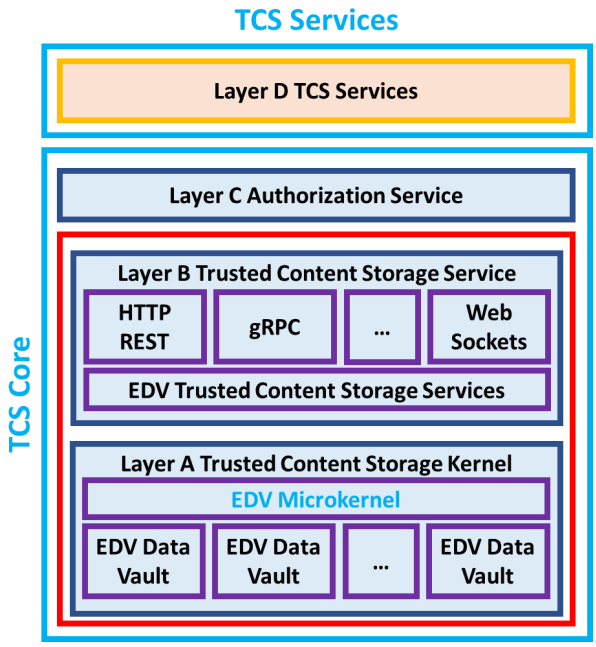


Figure 32. Layer A Content Change Detection

Content Change Detection's responsibility is to detect or otherwise sense changes being made to any content being made in an EDV Data Vault. This includes changes to configuration data, indices, etc. in addition to everyday content resources stored in a data vault.

In the specific context of the TCS Stack, this also includes Read operations as well as both successful operation attempts and unsuccessful operation attempts.

There are 3 variations (or models) for Layer B Content Change Detection/Tracking/Notification as depicted below.

- CRUD Events
- Update Sequence Numbers (USNs)
- Crawler Events

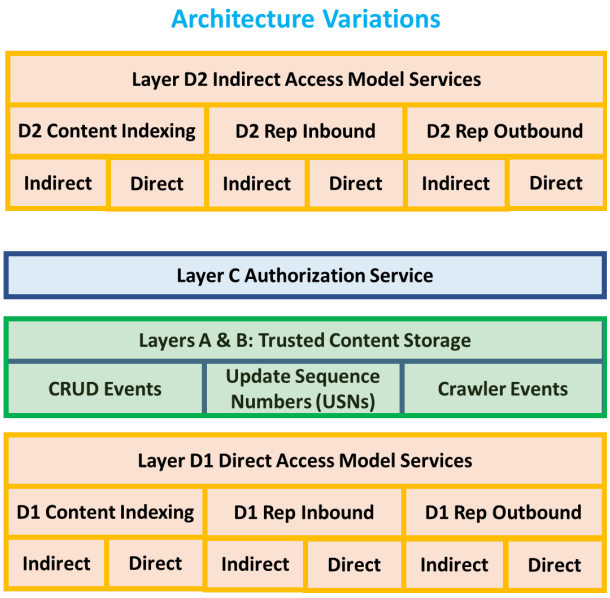


Figure 33. Layer A Content Change Detection/Tracking/Notification

The following is the enumeration of the architecture variations for Layer A Content Detection.

Table 1. Layer A Content Change Detection/Tracking/Notification: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct D2 Indirect	●5	●7	●4

Layer A Content Change Detection: CRUD Events Model

The Layer A CRUD Events Model is based on the simple callback-delegate pattern found in most software platforms (e.g. especially UI frameworks like Windows Forms, etc.). In the case of TCS Core Layer A, whenever a Consuming Client App or Service calls upon the Layer B Trusted Content Storage Service Service to perform an operation against a resource in a Layer A Trusted Content Storage Kernel, the EDV Kernel also calls out to any delegate that is registered to receive a CRUD Event (Create, Read, Update, Delete) against Trusted Content Storage. The delegate is either:

- Layer A Trusted Content Storage Kernel Logging Service
- Layer B Content Change Tracking function in the Layer B Trusted Content Storage Service

The CRUD Events Model variation of the Layer A Trusted Content Storage Kernel is illustrated in the following figure.

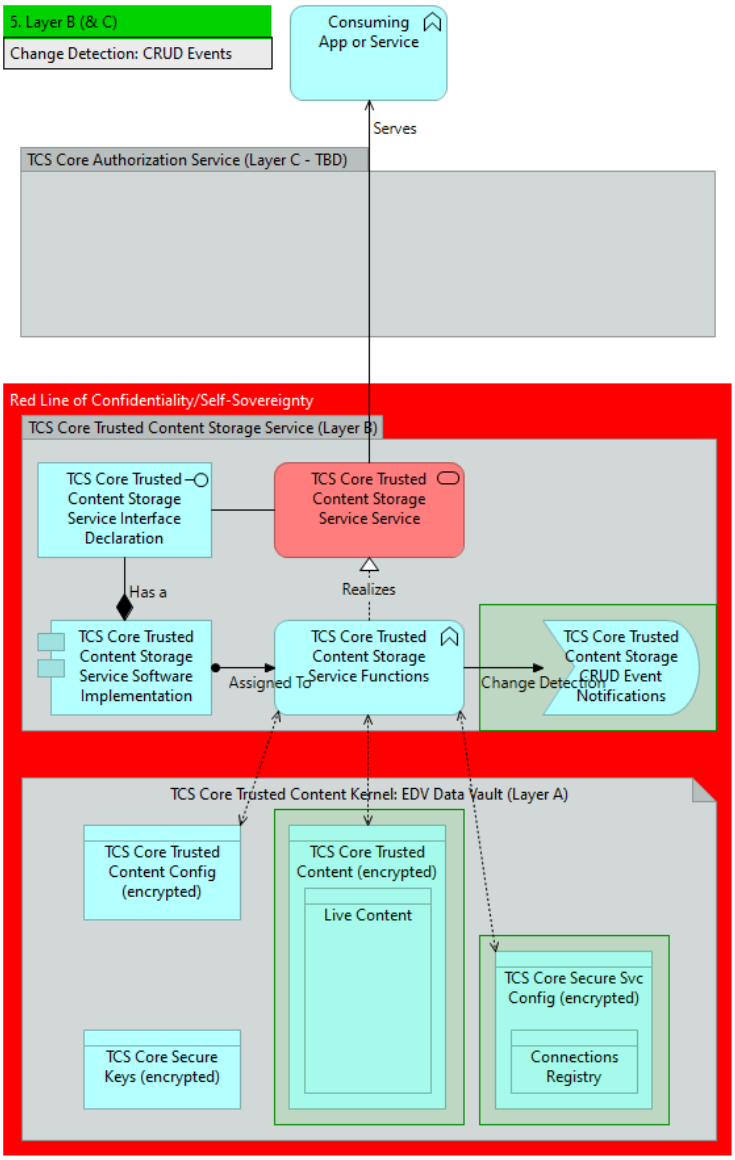


Figure 34. Layer A Content Change Detection: CRUD Events Model [5]

Layer A Content Change Detection/Tracking: USN Model

The Layer B Update Sequence Number (USN) Model is *not* based on the simple callback-delegate pattern found in most software platforms.

In the case of TCS Core Layer A, whenever a Consuming Client App or Service calls upon the Layer B Trusted Content Storage Service Service to perform an operation against a resource in a Layer A Trusted Content Storage Kernel, the service updates a *sequence* property associated with each resource in the Layer A Trusted Content Storage Kernel. The sequence property is updated with the value of the Next USN, an incremental counter stored in the Layer A Trusted Content Config container in a data vault mounted by the Layer A Trusted Content Storage Kernel. Next USN is incremented. Next USN is a unique counter that is scoped to each unique data vault mounted by the Layer A Trusted Content Storage Kernel.

A USN is typically a 64-bit unsigned integer. Similarly, Next USN is a 64-bit unsigned integer.

The USN sequence property ensures that there is a unique time-sequenced order to all modifications made across a particular instance of a Layer A Trusted Content Storage Kernel.

In addition, each USN and corresponding resource key are stored in a USN-Content Map Index maintained in the TCS Core Trusted Content Indices Container. The USN-Content Map is used to efficiently retrieve the resource key for a particular USN or list of resource keys corresponding to all the resources with a particular *Start USN* and higher.

The list of resource keys starting with the resource with sequence property = Start USN can then be used by a Layer D service to retrieve the most recently modified resources in the Live Content.

When a resource is deleted, it is not removed; rather the resource marked as “tombstoned” and logically or conceptually moved from Live Storage to Tombstone Storage. It is given a final new USN based on the current value of Next USN. Next USN is incremented.

The Update Sequence Number (USN) Model variation of the Layer A Trusted Content Storage Kernel is illustrated in the following figure.

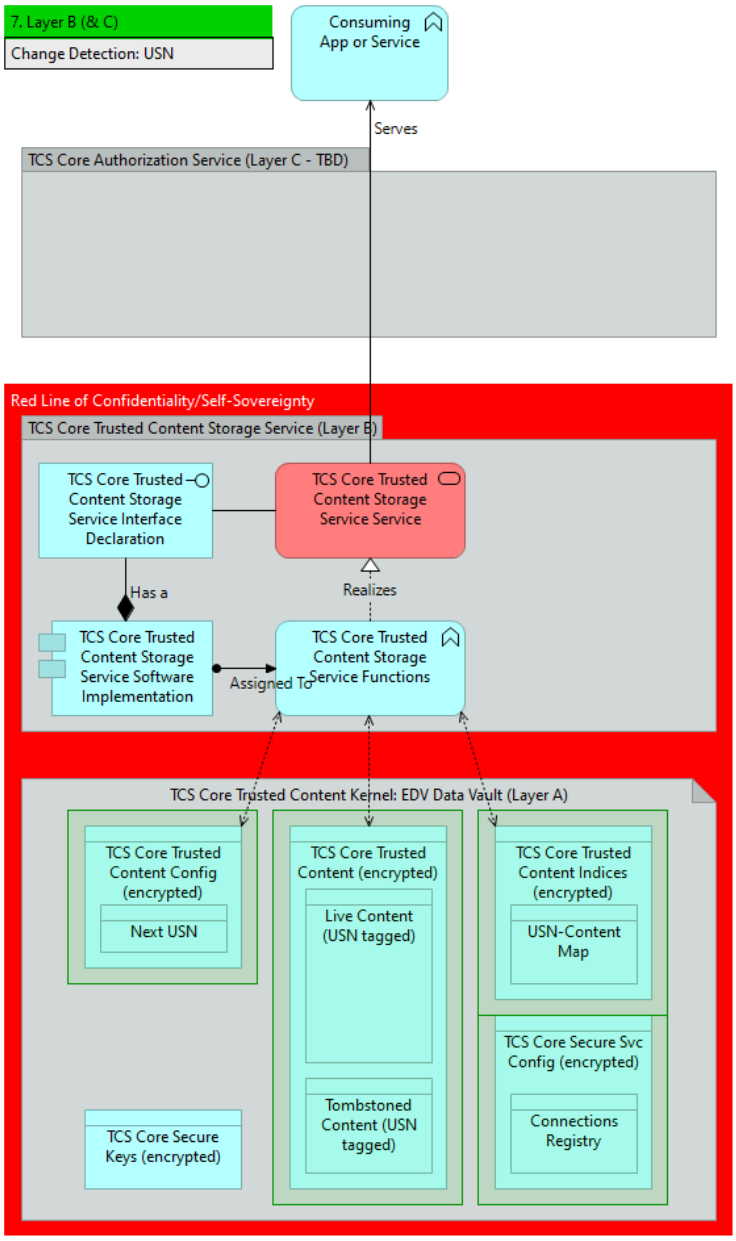


Figure 35. Layer A Content Change Detection: USN Model [7]

Layer A Content Change Detection: Crawler Model

The Crawler Model (or polling model) requires an active, running Layer D Trusted Content Crawler Service capable of periodically scanning the live content to update the Crawler Database in the TCS Core Trusted Content Indices container. This pattern is sometimes referred to as a polling pattern (vs. an event-driven pattern).

The Crawler Model for Content Change Detection requires no specific support in the Layer A Trusted Content Kernel (nor the EDV Kernel).

The Crawler Model is depicted in the figure below.

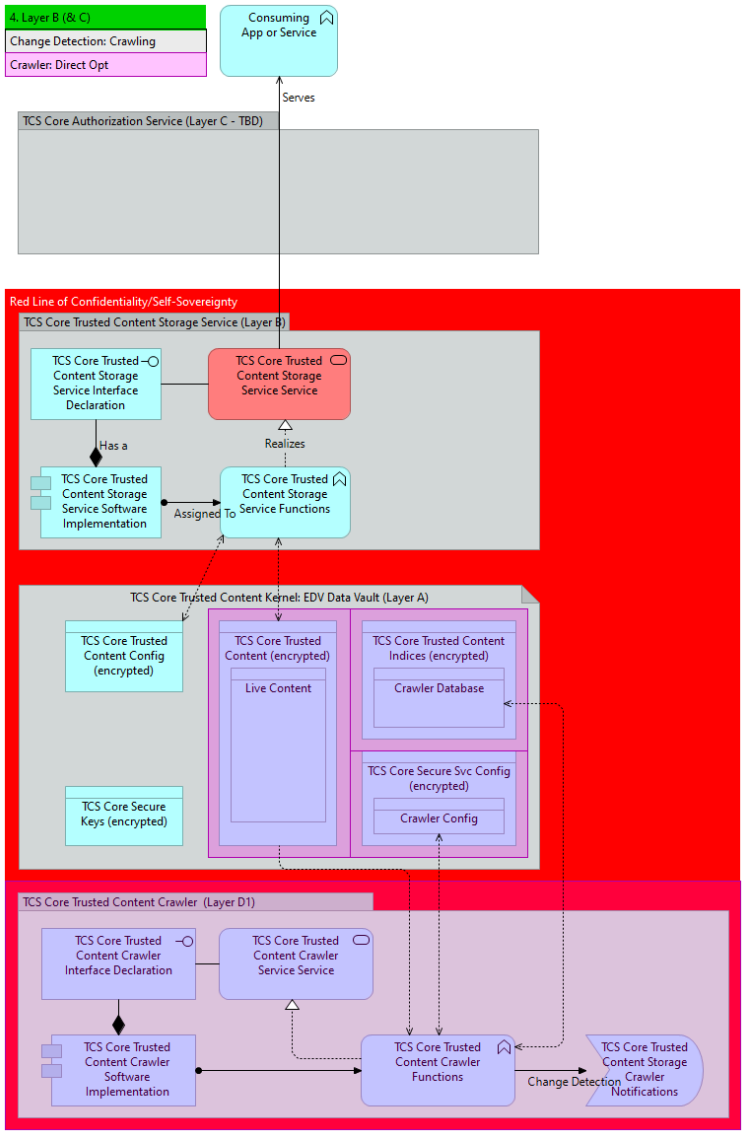


Figure 36. Layer B Content Change Detection/Tracking/Notification: Crawler Model [4]

Heterogeneous Multi-vaults/Server Instance Support

The first real architecture variation in the TCS Stack is the ability to host either:

- A homogeneous collection of EDV Data Vaults
  - Same EDV technology provider
  - Same underlying EDV technology
  - The same major version of the EDV technology
- A heterogeneous collection of EDV Data Vaults
  - Multiple EDV technology providers
  - Multiple underlying EDV technologies
  - The minimum version of a specific underlying EDV technology (compatible with the specification)

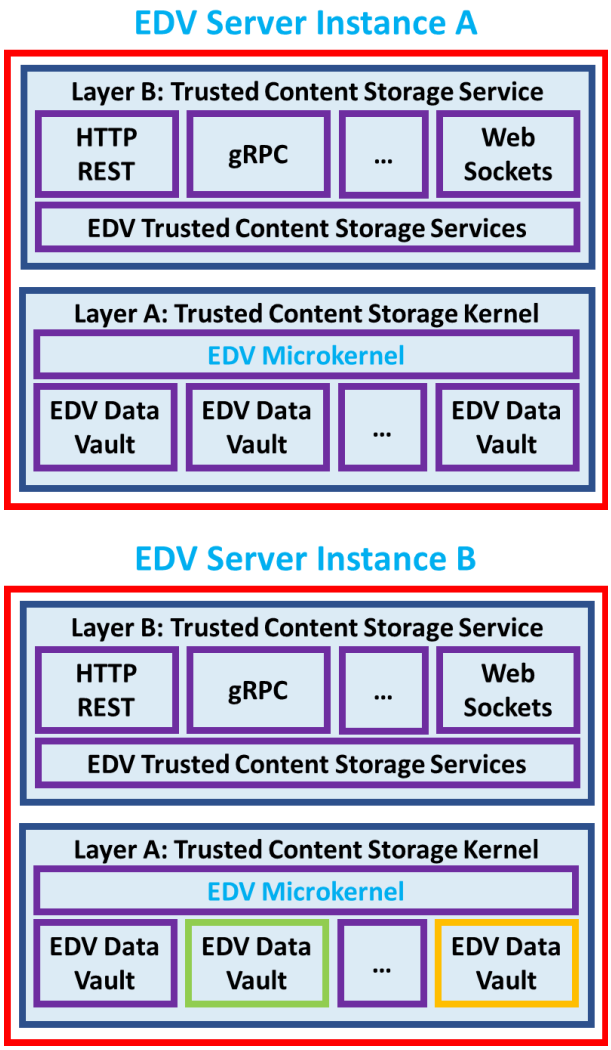


Figure 37. (a) Homogeneous Multi-vaults/Server Instance and (b) Heterogenous Multi-vaults/Server Instance

NOTE: If the latter variation is supported (heterogenous multi-vaults/service instance), the implication is that the EDV Microkernel is the lowest level interoperability interface in the TCS Stack.

TCS Core Layer B Architecture Variations

There are 2 variations for TCS Core Layer B (Trusted Content Storage Service):

- Content Detection, Tracking, and Notification Models
- Supported Protocol Endpoints

ISSUE: In this version of the document, Content Change Detection is assumed to be a Layer A Trusted Content Storage Kernel function (although the ARM diagrams may not all reflect this). Content Change Tracking and Notification are relegated to the Layer B Trusted Content Storage Service.

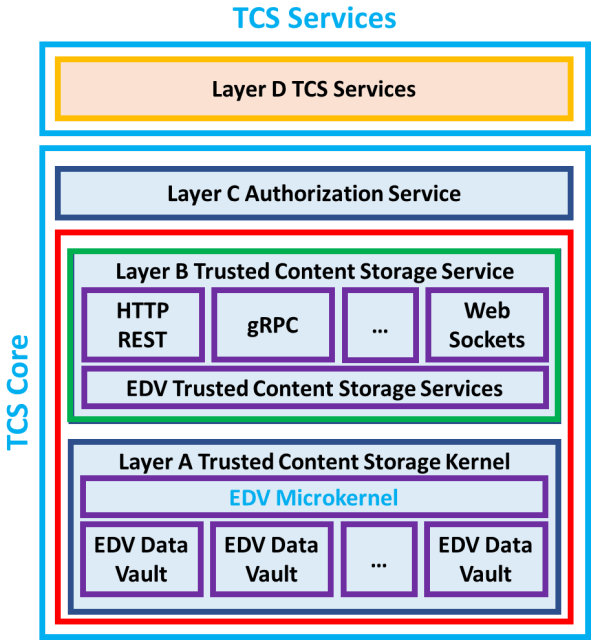


Figure 38. TCS Core Layer B Architecture Variations

Layer B Trusted Content Storage Service

There are 2 variations for TCS Core Layer B (Trusted Content Storage Service):

- Content Detection, Tracking, and Notification Models
  - CRUD Events
  - USNs
  - Crawler based
- Supported Protocol Endpoints, for example,
  - HTTP REST
  - gRPC
  - Web Sockets
  - Etc.

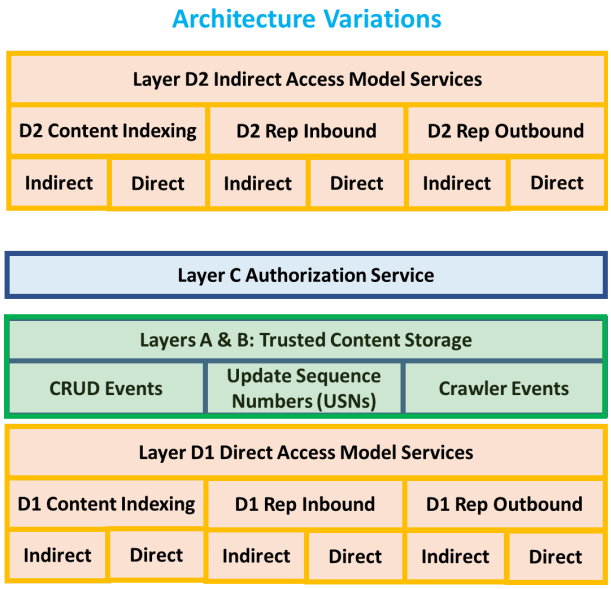


Figure 39. TCS Core Layer B Variations

The ARMs for these variations appear below.

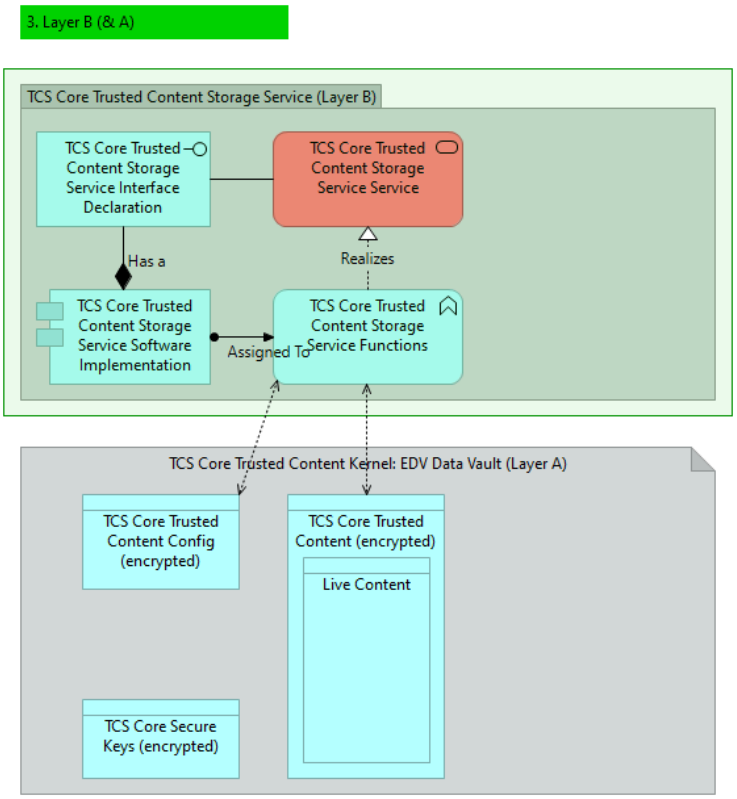


Figure 40. Layer B Trusted Content Storage Service [3]

Looking specifically at Layer B Trusted Content Storage Service, the "service" is comprised of 4 elements:

- Application Component (software implementation)
- Application Service (public capabilities defined by an API definition)
- Application Interface (public API definition)
- Application Function (component capabilities)
- Application Event (content change detection trigger) – optionally

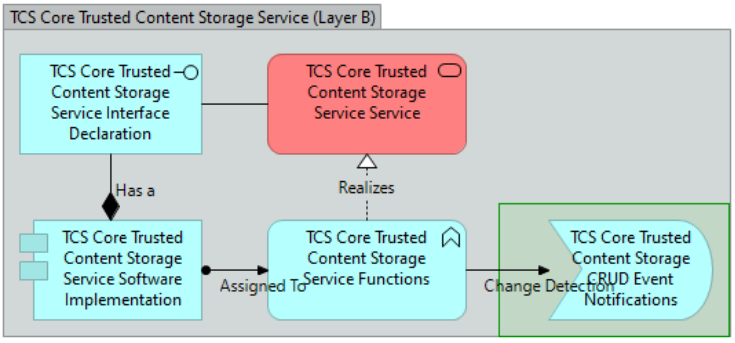


Figure 41. Layer B Trusted Content Storage Service Elements

The Application Component represents the software implementation of the Layer B Trusted Content Storage Service (the lower-left component in the diagram below). The Application Component has or

composes with an Application Interface: the Layer B Trusted Content Storage Service Interface Declaration (the top-left component in the diagram below). In addition, the Application Component is assigned to an Application Function element (bottom-center), Layer B Trusted Content Storage Service Functions, representing the functionality exposed by the Application Service (top-center), the Layer B Trusted Content Storage Service.

The functionality exposed by the Layer B Trusted Content Storage Service is declared by the Application Interface (top-left), Layer B Trusted Content Storage Service Interface Declaration, described earlier.

> *NOTE: These 4 components represent a repeating pattern that is re-used throughout the ARMs presented in this document.*

Lastly, an Application Event (lower-right corner) is used to represent the Layer B Trusted Content Storage CRUD Event Notifications.

Layer B Content Change Tracking and Notification

ISSUE: In this version of the document, Content Change Detection is assumed to be a Layer A Trusted Content Storage Kernel function (although the ARM diagrams may not all reflect this). Content Change Tracking and Notification are relegated to the Layer B Trusted Content Storage Service.

Once a change has been detected, Content Change Tracking is responsible for tracking the change by either: (a) raising or triggering an event, (b) using an update sequence number (USN) to track changes on a resource-by-resource basis, or (c) recording the event in a secondary container in the data vault.

In the TCS Stack, the responsibility for Content Change Tracking is assigned to the Layer B Trusted Content Storage Service.

Once a change has been detected and tracked, Content Notification is responsible for notifying TCS Services Layer D Services so that the change can be acted upon. These may be Layer D1 Services with direct access to the data vault via the EDV Microkernel or Layer D2 Services with indirect access to the data vault via the Layer B Trusted Content Storage Service Service.

In the TCS Stack, the responsibility for Content Change Notification is assigned to the Layer B Trusted Content Storage Service.

This basic capability enables a diverse range of TCS Services Layer D Services to be triggered based on content changes occurring in Trusted Content Storage. A possible list of TCS Services Layer D Services includes:

- Indexing
- Replication Outbound Processing Service
- Audit History Logging
- Resource Version History
- Enactment of event-triggered Workflows (Business Processes)
- Security Monitoring

Layer B Content Change Tracking and Notification: CRUD Events Model

The next figure is an illustration (an example) of how the CRUD Events model might be used to trigger a Layer D Content Indexing Service Service.

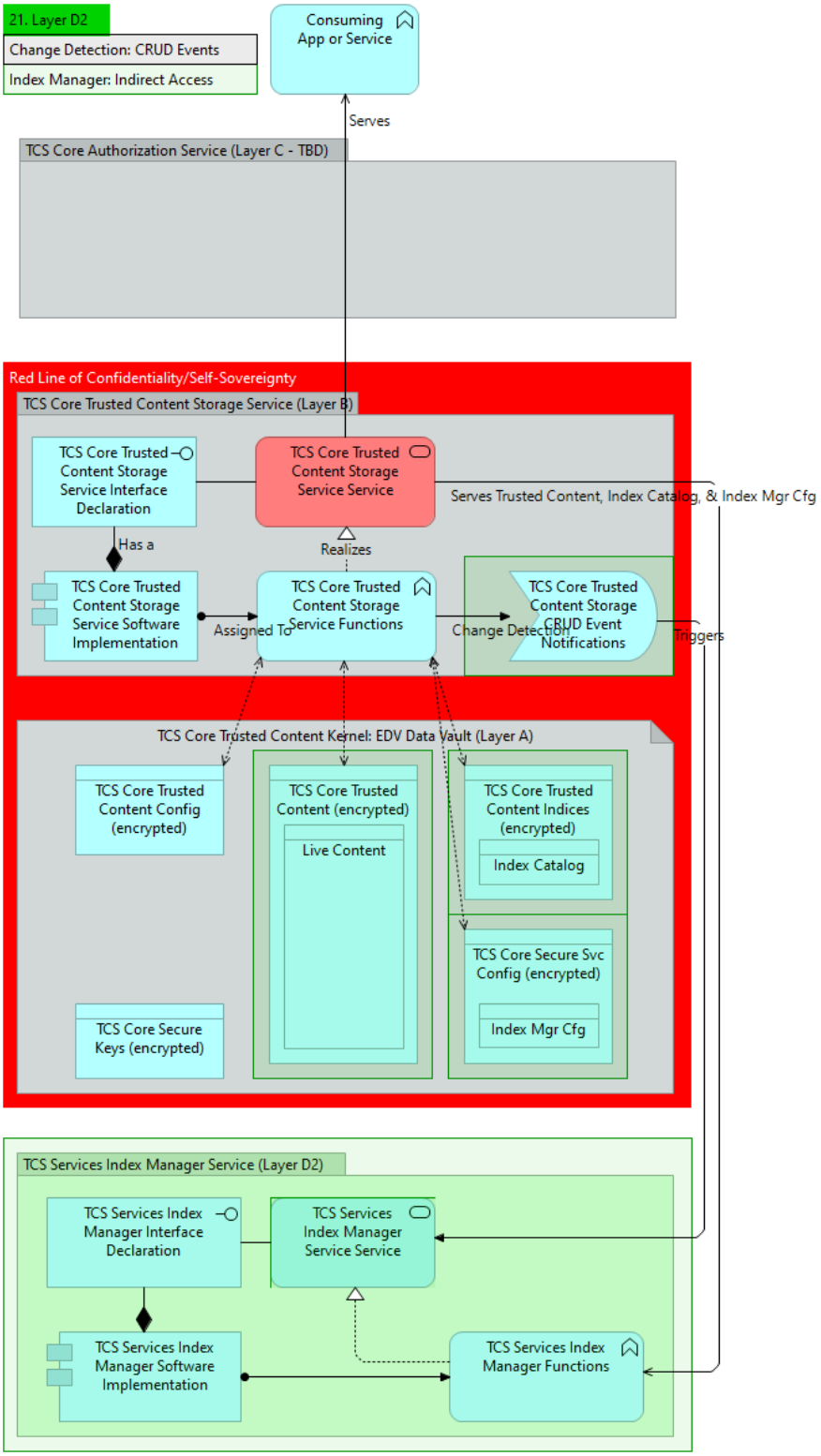


Figure 42. Layer B CRUD Events Model: Layer D2 Index Manager Service Example [21]

In the above example, Layer A Content Change Detection triggers the Layer B Content Change Tracking and Notification function to raise an event with one or more keys to identify the resource that was modified as well as information about the operation that caused the modification.

In turn, Layer B Content Change Notifications calls into the TCS Services Index Manager Service Service passing in the resource keys and operation information. The TCS Services Index Manager Service Functions then perform the updating operations (add/change/delete) on the Index Catalog in Trusted Content Storage by calling back into the Layer B Content Secure Storage Service Service.

This repeats for every Content Change Notification (or batch of notifications) generated by Layer A Content Change Detection logic.

Layer B Content Change Notification: USN Model

The next figure is an illustration (an example) of how the Update Sequence Number (USN) model might be used to trigger a Layer D Replication Outbound Processing Service. Each service that needs to consume USN sequenced content relative to a particular USN Start value first needs to create a Connection in the Connections Registry in the TCS Core Secure Service Config container. A Connection contains information like the filters to be used in selecting the content to be made available to the calling service but most important the subject service needs to keep track of the Last USN it has received and presumably processed from TCS Core Trusted Content Storage (via the Layer B Trusted Content Storage Service). The Last USN value needs to be tracked on a Connection by Connection basis. The value of Last USN + 1 becomes the Start USN value used to retrieve newly modified content from Layer A Trusted Content Storage Kernel via the Layer B Trusted Content Storage Service. The newly modified content will mostly include resources from Live Content but if a newly modified resource is *deleted* resource, the Layer B Trusted Content Storage Service will return a *tombstoned* resource.

> *NOTE: In this architecture variation, all access to the above-named resources in Trusted Content Storage is performed via the Layer B Trusted Content Storage Service Service[7].*

[7] The Layer B Trusted Content Storage Service Service, in effect, acts as the portal or gateway for accessing content in Layer A Trusted Content Storage.

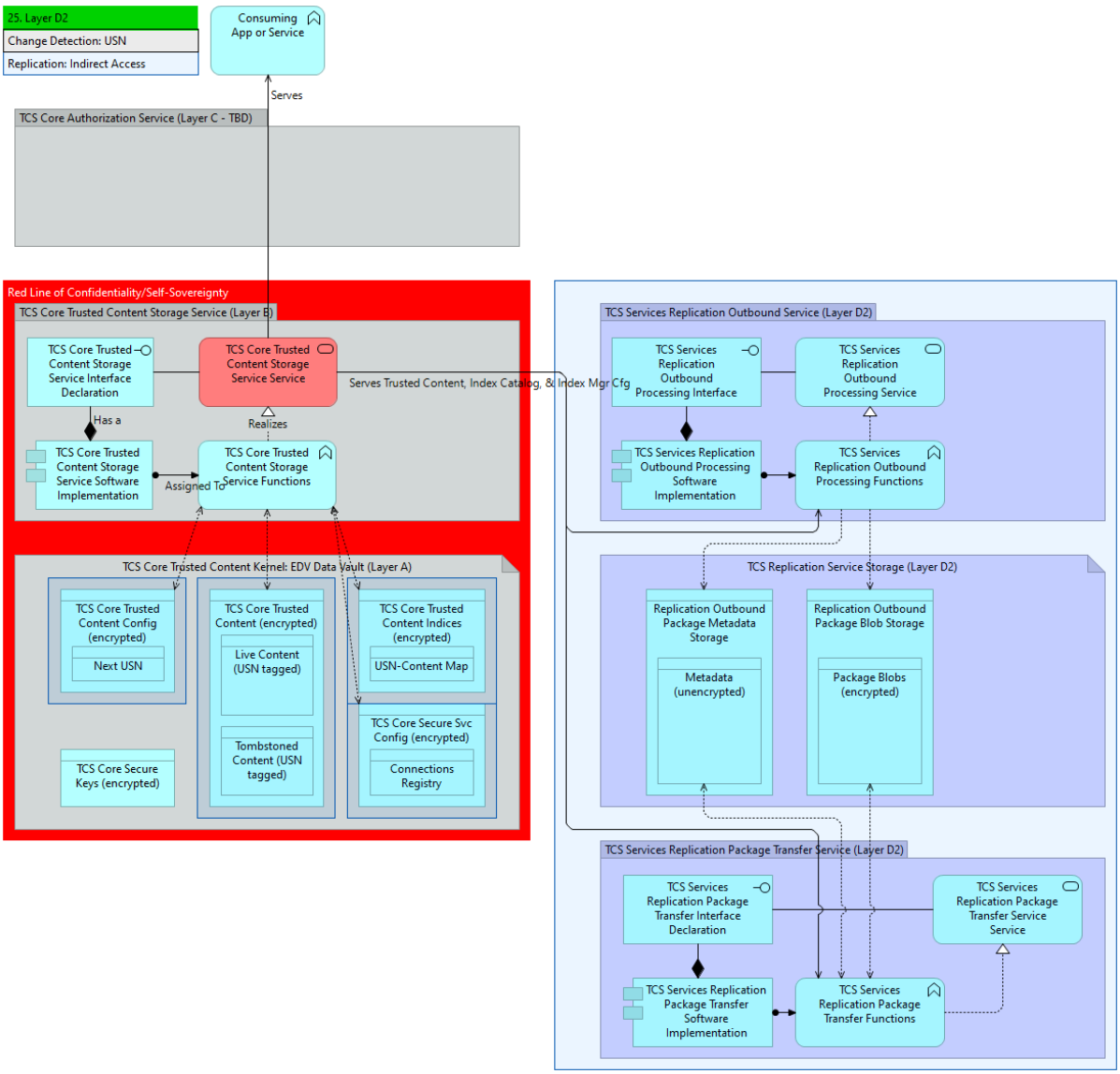


Figure 43. Layer B USN Model: Layer D2 Replication Outbound Processing Service Example [25]

Layer B Content Change Detection, Tracking, and Notification: Crawler Model

The next figure is an illustration (an example) of how the Crawler model might be used to trigger a TCS Services Layer D Services Index Manager service. Every resource found during the Crawler scanning process (possibly filtered) is detected by Content Change Detection and, in turn, triggers a Crawler Model Content Change Notification.

In this example, the Crawler Model Content Change Notification triggers the Layer D1 Index Manager Service Service to enable the service to update the Index Catalog in Trusted Content Indices container in Layer A Trusted Content Storage Kernel.

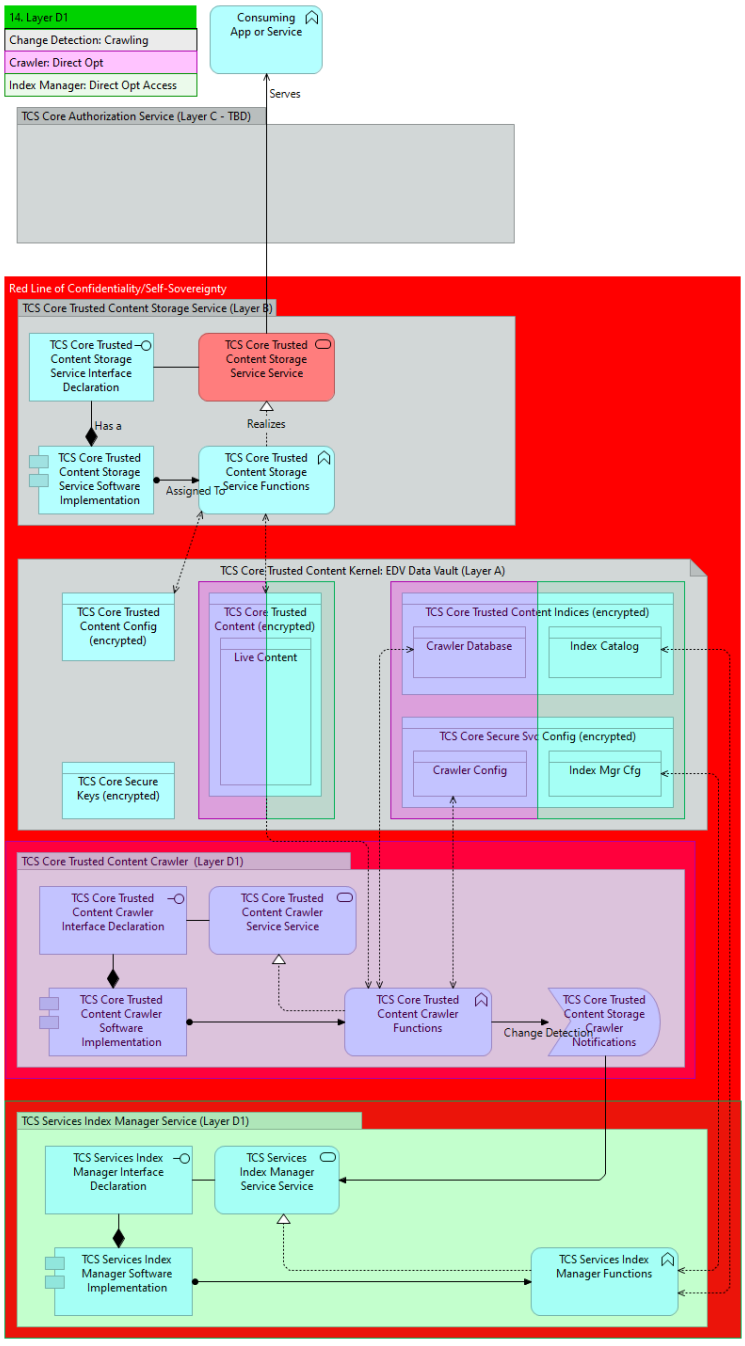


Figure 44. Layer B Crawler Model: Layer D1 Index Manager Service Example [14]

Red Line of Confidentiality/Self-Sovereignty

The Red Line of Confidentiality/Self-Sovereignty, illustrated in the Red Line of Confidentiality/Self-Sovereignty figure below, is a key concept used through-out this whitepaper.

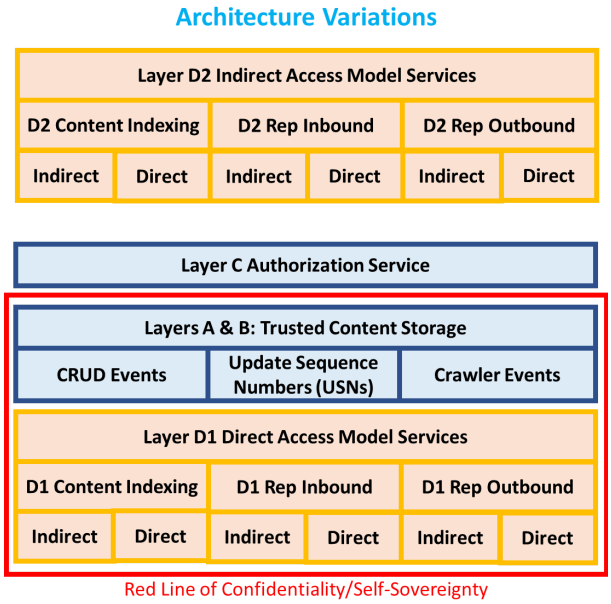


Figure 45. Red Line of Confidentiality/Self-Sovereignty

As a visual aid, the Red Line of Confidentiality/Self-Sovereignty is needed to ensure no service outside TCS Core attempts to access a secure content storage resource “from the side”.

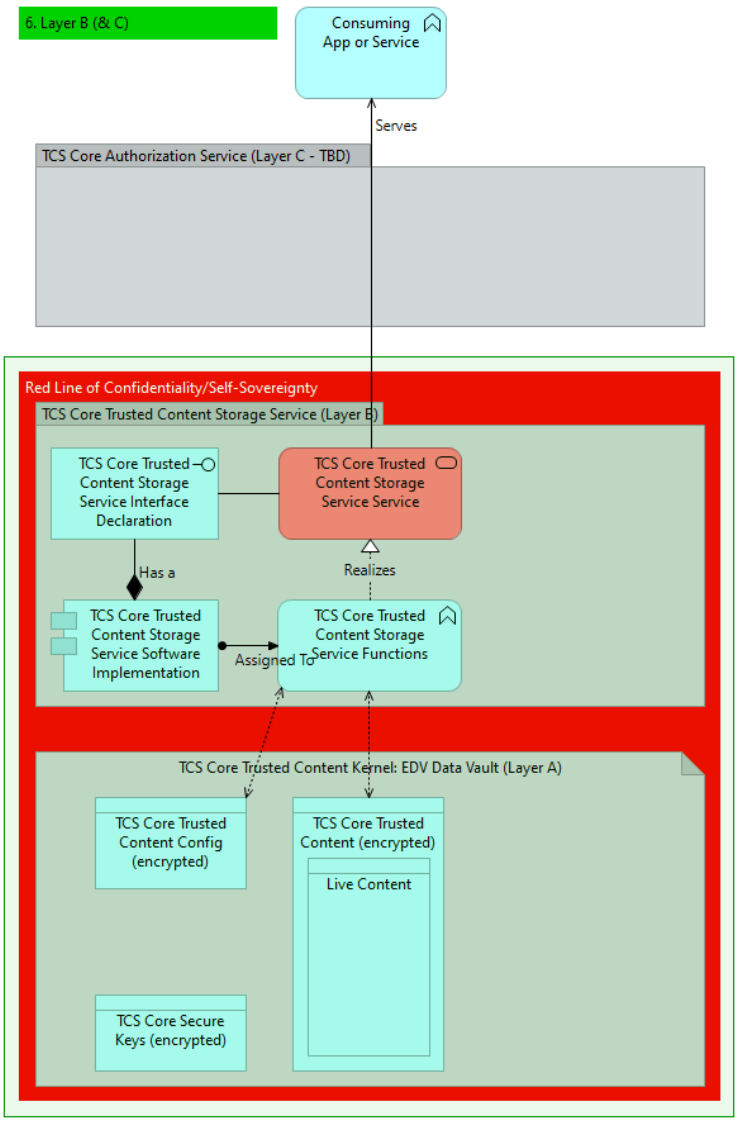


Figure 46. Figure 47. TCS Core Layers A and B: The Red Line of Confidentiality/Self-Sovereignty [6]

By default, all access to Trusted Content Storage is expected to take place via the Layer B Trusted Content Storage Service; more specifically, the Layer B Trusted Content Storage Service Service.

TCS Core Layer C Architecture Variations

NOTE: This section is a placeholder for future discussion and elicitation of the architecture variations for Layer C Authorization Service.

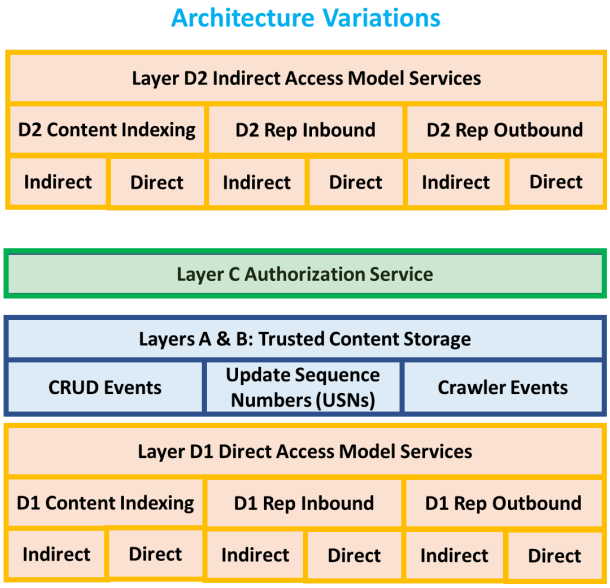


Figure 48. TCS Core Layer C Architecture Variations

Layer C Authorization Service

NOTE: This section is a placeholder for future discussion and elicitation of the architecture variations for Layer C Authorization Service.

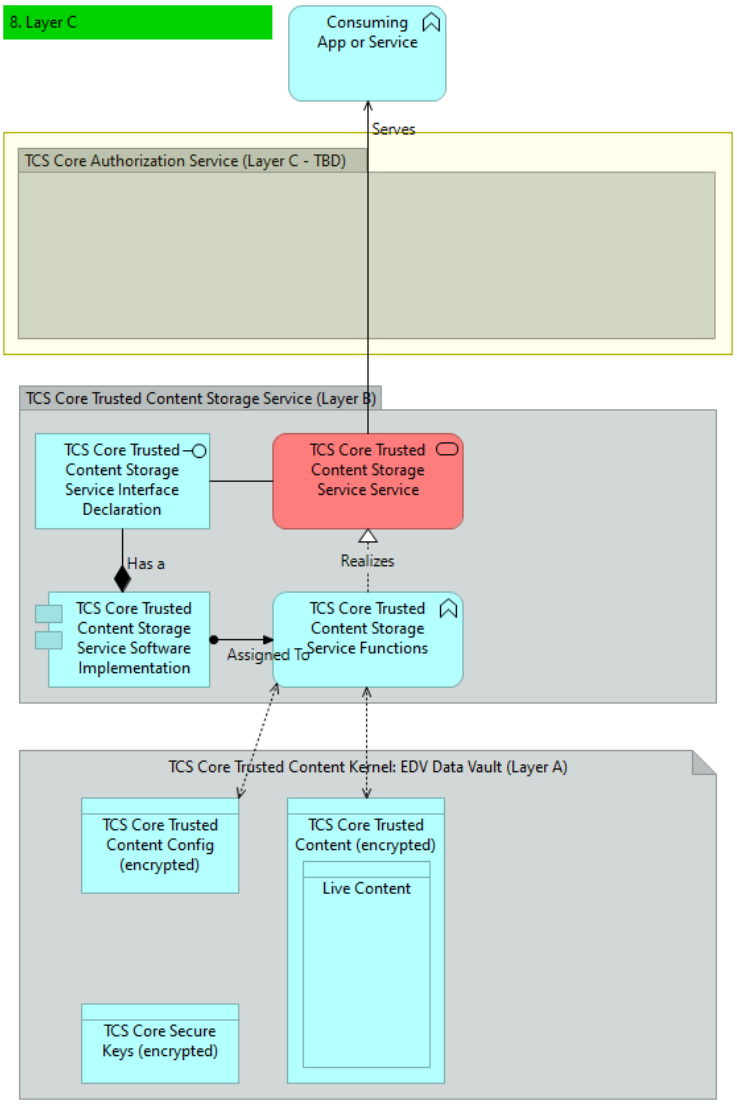


Figure 49. Layer C Authorization Service [8]

TCS Services Layer D Architecture Variations

The primary dimensions of the architecture variations in TCS Services Layer D Services are the following:

1. TCS Core
    a. Layer A Trusted Content Storage Kernel
        i. Layer A Content Change Detection Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    b. Layer B Trusted Content Storage Service
        i. Layer B Content Change Tracking and Notification Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    c. Layer C Authorization Service[8]
2. TCS Services Layer D Access Models
    a. Layer D1 Direct Access Model
    b. Layer D2 Indirect Access Model
3. TCS Services
    a. Layer D Replication Services
        i. Layer D Replication Outbound Processing Service
        ii. Layer D Replication Package Transfer Service
        iii. Layer D Replication Inbound Processing Service
    b. Layer D Indexing & Search Services
        i. Layer D Content Indexing Service

[8] NOTE: TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper.

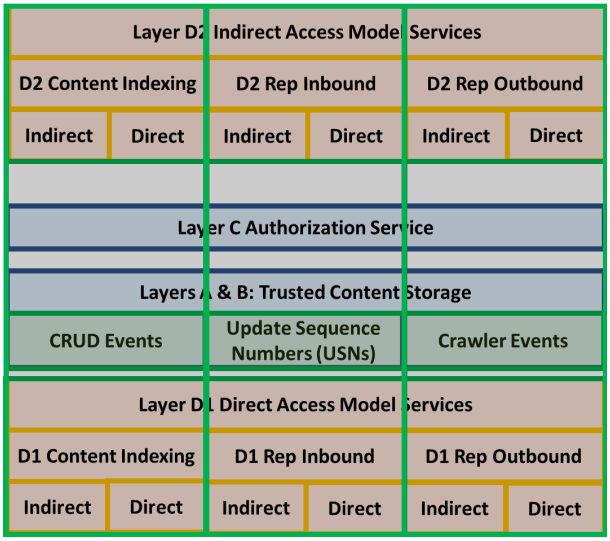


Figure 50. TCS Services Layer D Architecture Variations

Two important TCS Services Layer D Services not mentioned in the above list include:

- Layer D Replication Package Transfer Service
- Layer D Search Query Processing Service

A detailed discussion of these 2 TCS Services Layer D Services is not necessary from an architecture variations perspective – there is only one generic architecture option for the Layer D Replication Package Transfer Service and Layer D Search Query Processing Service. These solitary options will be described in the section Architecture Recommendations later in this document.

Layer A and Layer B Content Change Detection/Tracking/Notification Model Implications

The implications of the Layer A and Layer B Content Change Detection/Tracking/Notification Model on the range of TCS Services Layer D Services are significant.

The architectural implications for the Layer D Services are initially driven by the choice of the Content Change Detection/Tracking/Notification Model implemented in Layer A and Layer B:

1. CRUD Event Model
2. USN Model
3. Crawler Model

The next most important dimension is the Layer D Access Models:

a. Layer D1 Direct Access Model
b. Layer D2 Indirect Access Model

Followed by the individual Layer D Services themselves.

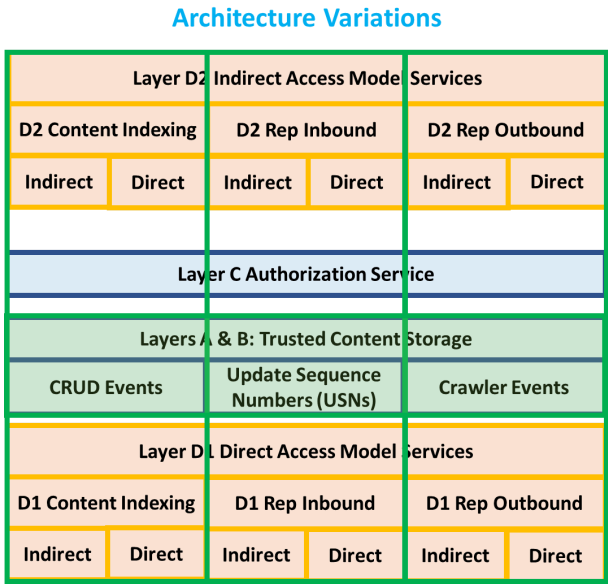


Figure 51. Layer D Services and Layer B Content Change Detection/Tracking/Notification Models

Layer D Direct and Indirect Access Service Models

TCS Services Layer D Services is further partitioned into 2 sublayers:

- Layer D1 Direct Access Services
- Layer D2 Indirect Access Services

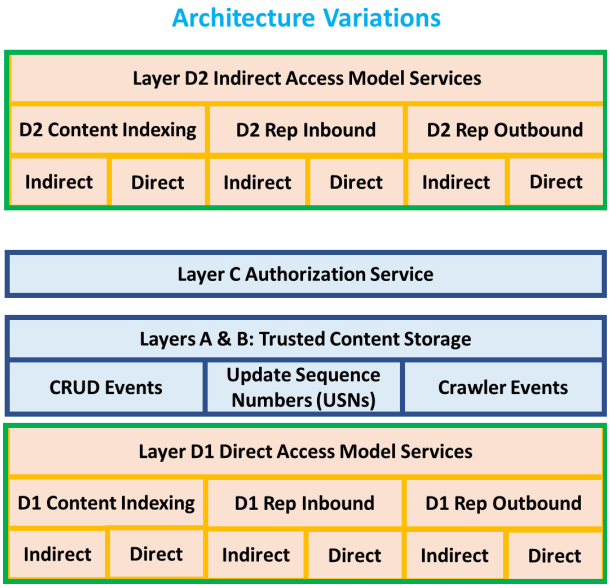


Figure 52. Layer D Direct and Indirect Access Service Models

Layer D1 Direct Access Service Model

Layer D1 Direct Access Services is a subset of TCS Services Layer D Services that, effectively, run inside the Red Line of Confidentiality/Self-Sovereign, or otherwise, have access to the Layer A storage via the Layer B API (indirect access). This, potentially, includes optimized direct access to Layer A storage (direct access).

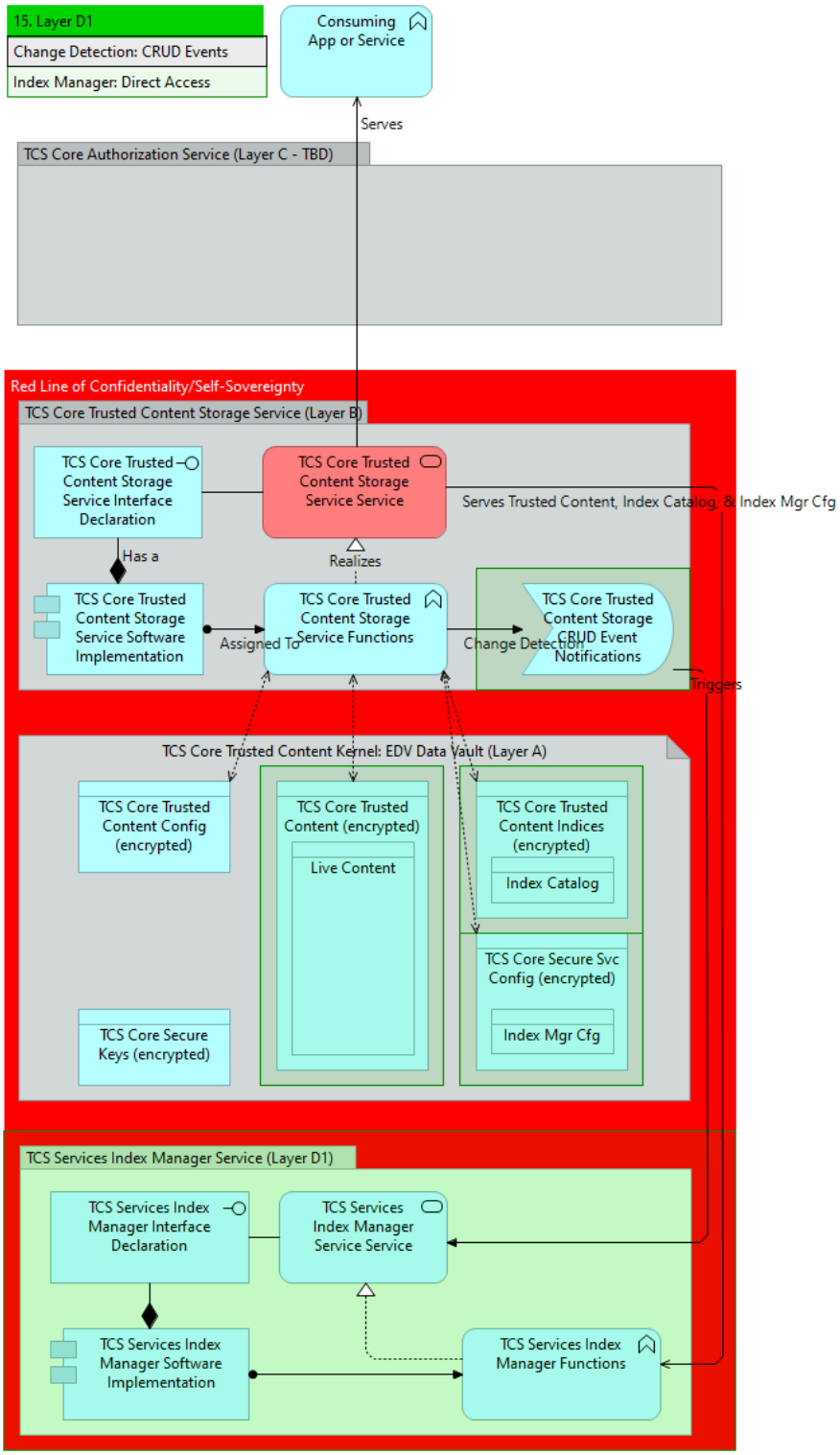


Figure 53. Layer D1: Indirect Access Service Model Example [15]

Layer D2 Indirect Access Service Model

Layer D2 Indirect Access Services is a subset of TCS Services Layer D Services that run outside the Red Line of Confidentiality/Self-Sovereign, or otherwise, never have direct access to the Layer A storage other than through the Layer B API (indirect access).

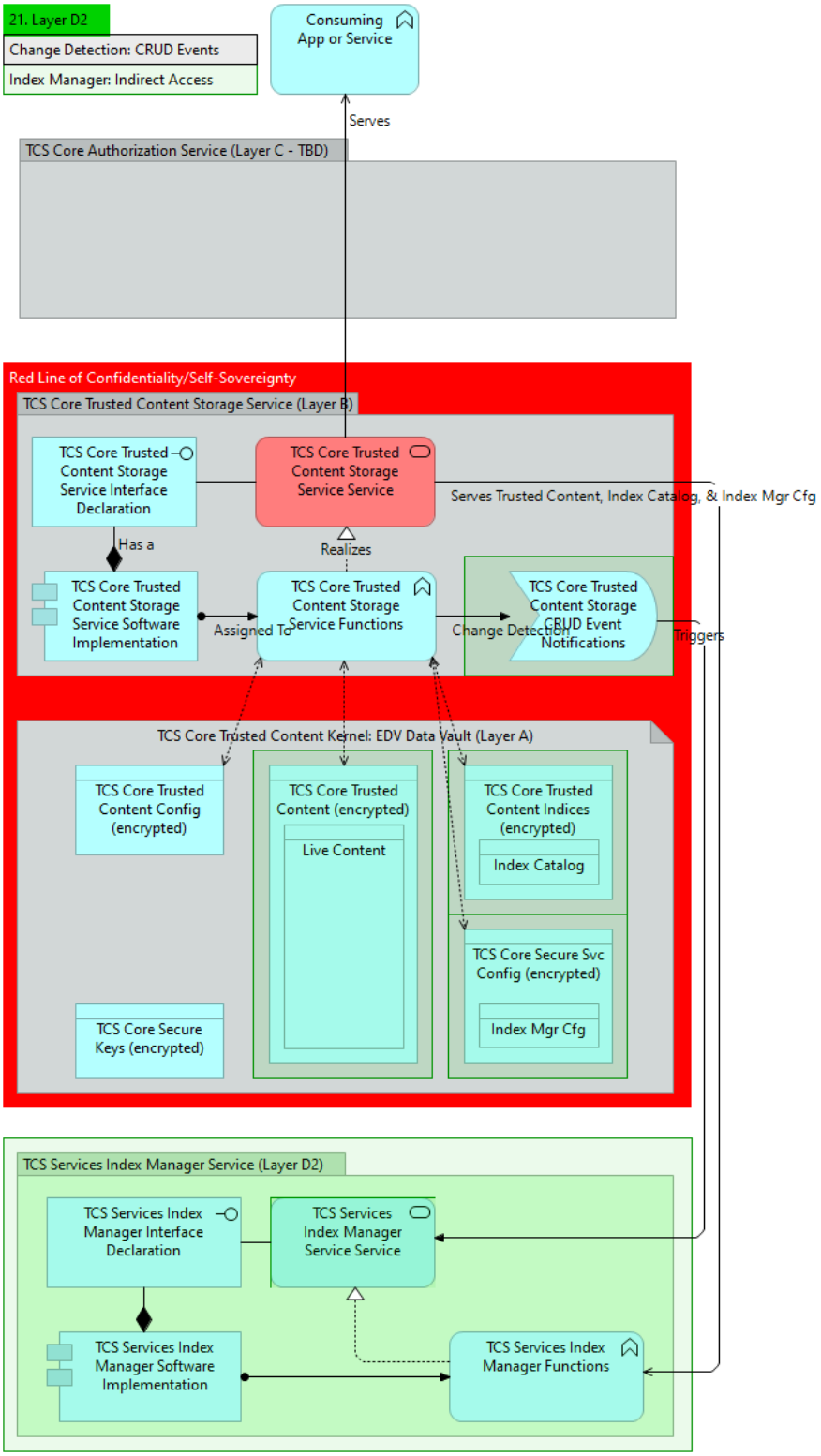


Figure 54. Layer D2 Indirect Access Service Model Example [21]

Layer D Replication Services

Layer D Replication Services includes:

i. Layer D Replication Outbound Processing Service
ii. Layer D Replication Package Transfer Service
iii. Layer D Replication Inbound Processing Service

The ARMs for each of the Layer D Replication Services architecture variations (depicted in the following figure) are presented in the following sections.

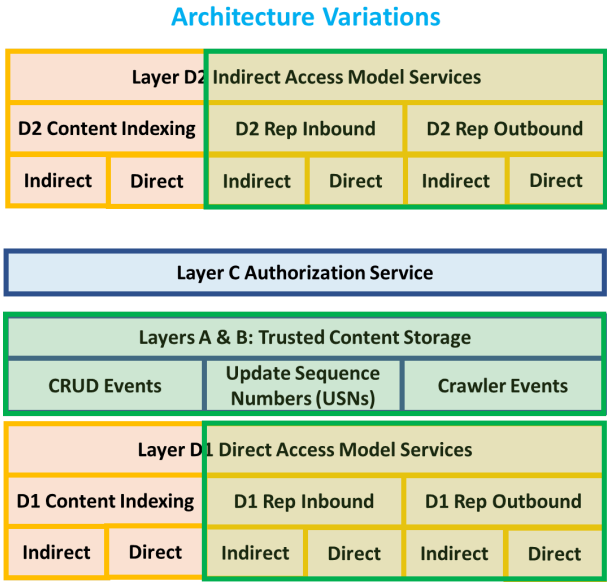


Figure 55. Layer D Replication Services: Architecture Variations

The Replication Package Transfer Service doesn't appear in the above diagram because it has a single architecture variation that is independent of the chosen Layer A and Layer B Content Change Detection/Tracking/Notification Model and Layer D Access Service Model.

Replication Outbound Processing Service

The ARMs for the architecture variations for the Replication Outbound Processing Service are described below.

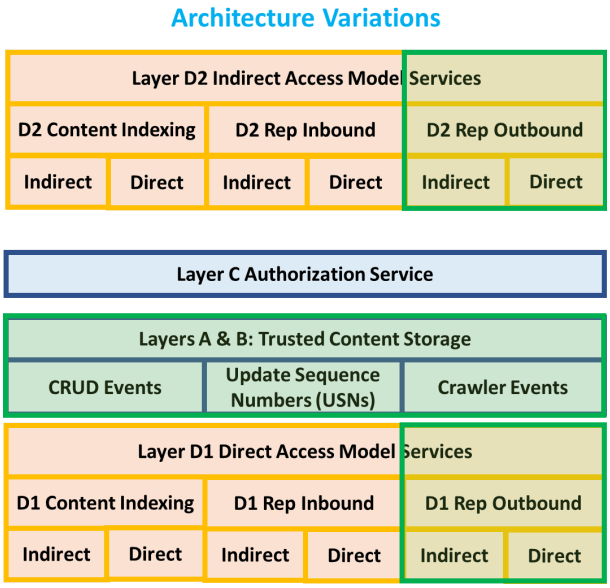


Figure 56. Replication Outbound Processing Service: Architecture Variations

ARMs for the following 4 Replication Outbound Processing Service architecture variations appear below.

Table 2. Replication Outbound Processing Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct		●20	●13
D2 Indirect	●23	●25	

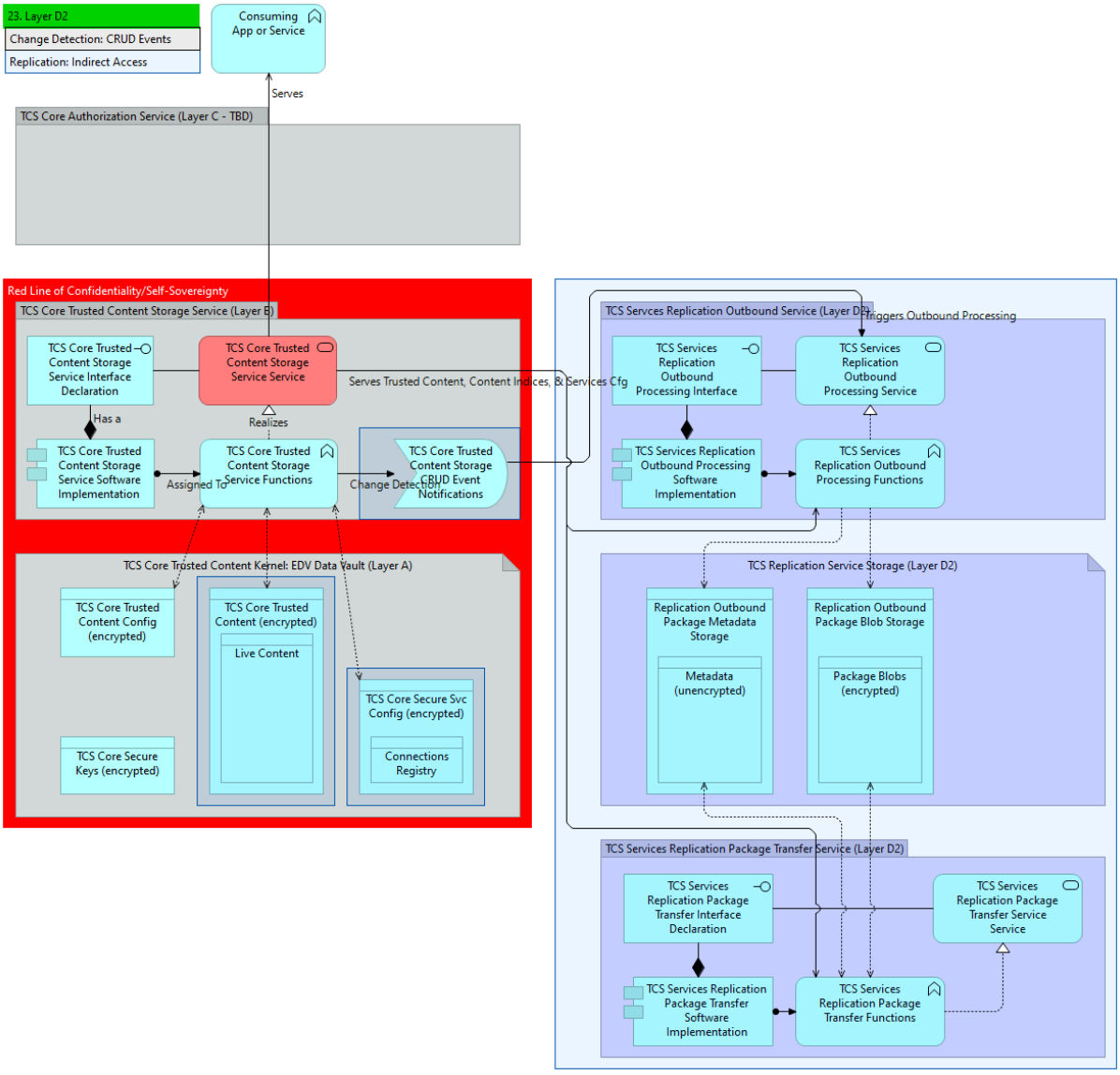


Figure 57. Replication Outbound Processing Service: CRUD Event Model, D2 Indirect Access [23]

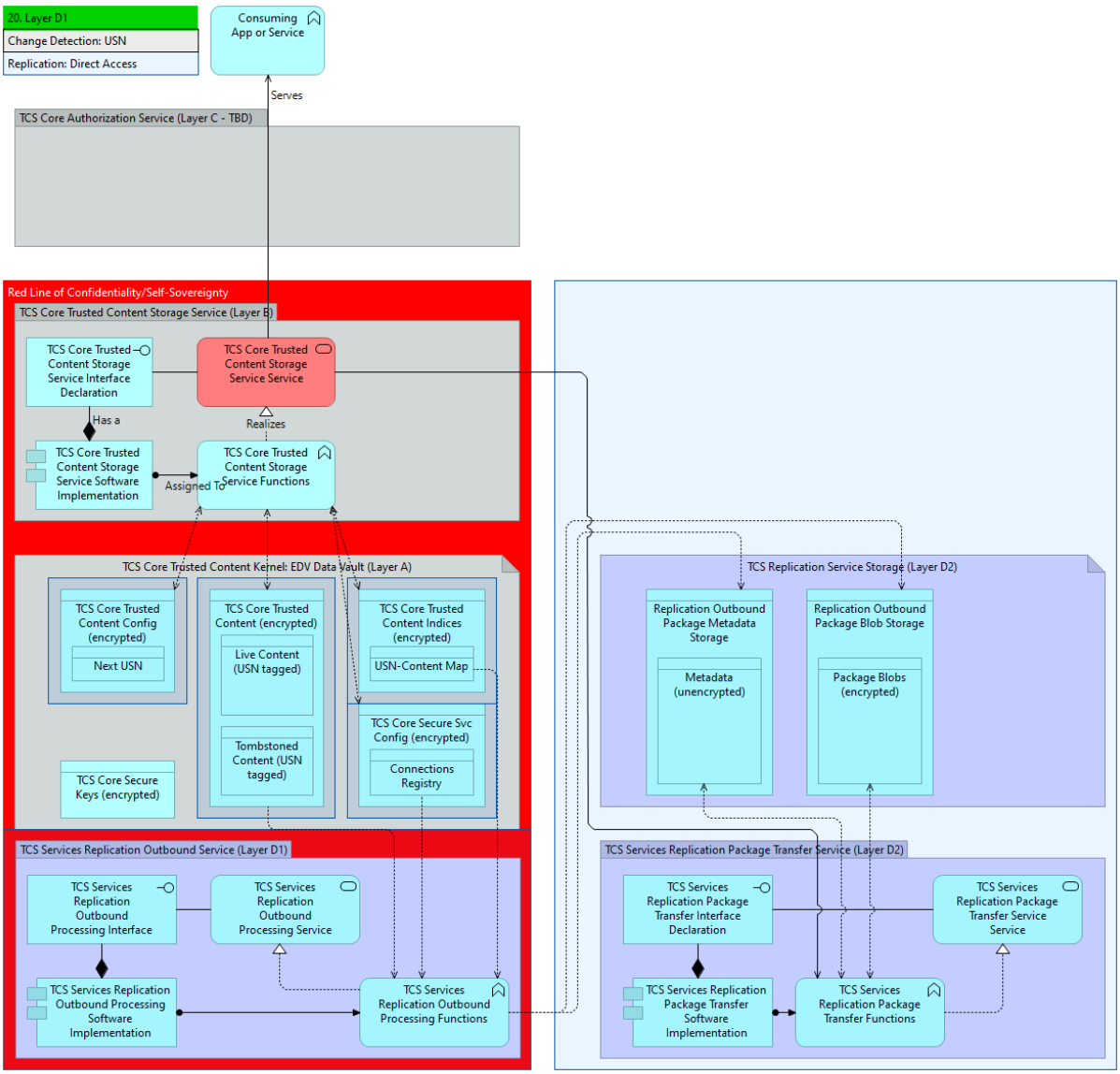


Figure 58. Replication Outbound Processing Service: USN Model, D1 Direct Access [20]

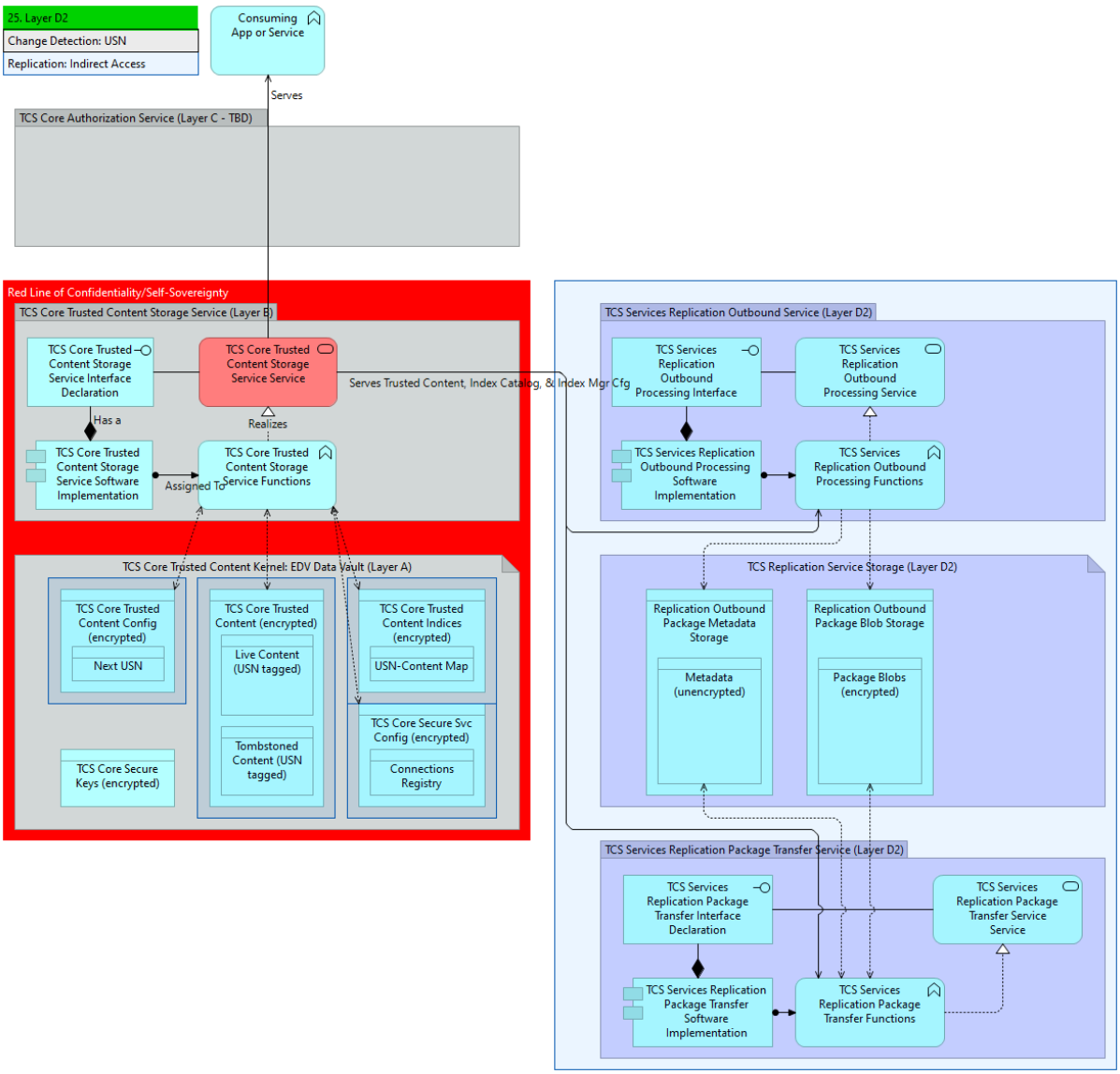


Figure 59. Replication Outbound Processing Service: USN Model, D2 Indirect Access [25]

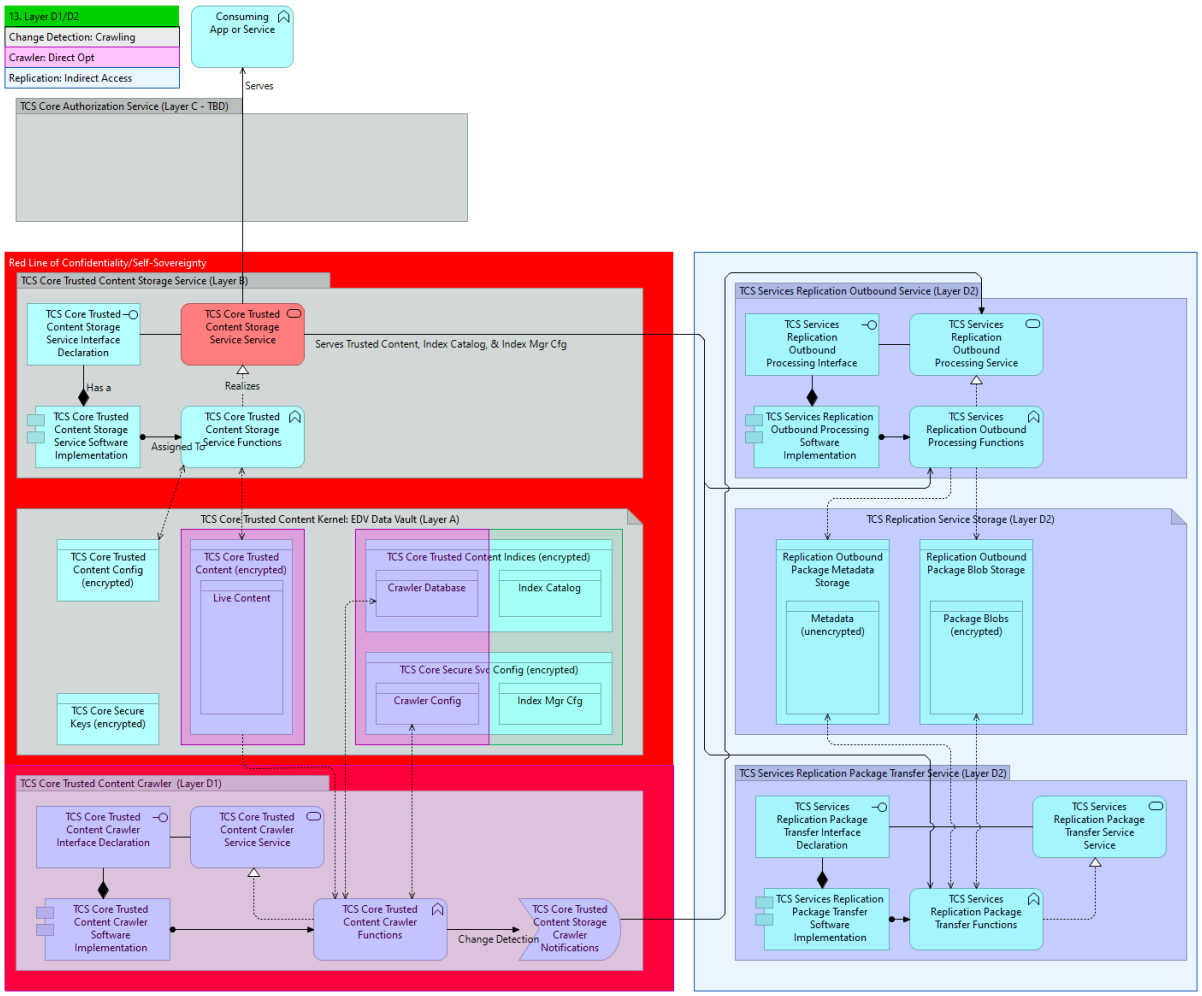


Figure 60. Replication Outbound Processing Service: Crawler Model, D2 Indirect Access [13]

Replication Package Transfer Service

The ARMs for the architecture variations for the Replication Package Transfer Service are described below.

Technically, there is only 1 architecture variation for the Replication Package Transfer Service because:

- The service always runs using the Layer D2 Indirect Access Service Model
- The service design and deployment does not vary with the chosen Layer A and Layer B Content Change Detection/Tracking/Notification Model.

ARMs for 2 Replication Package Transfer Service scenarios appear below. The two scenarios are presented using 2 different Replication Outbound Process Service access service models (D1 and D2) – both of which use, as an example, the USN Model Content Change Detection/Tracking/Notification Model.

Table 3. Replication Package Transfer Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct	---		
D2 Indirect		●20.25	

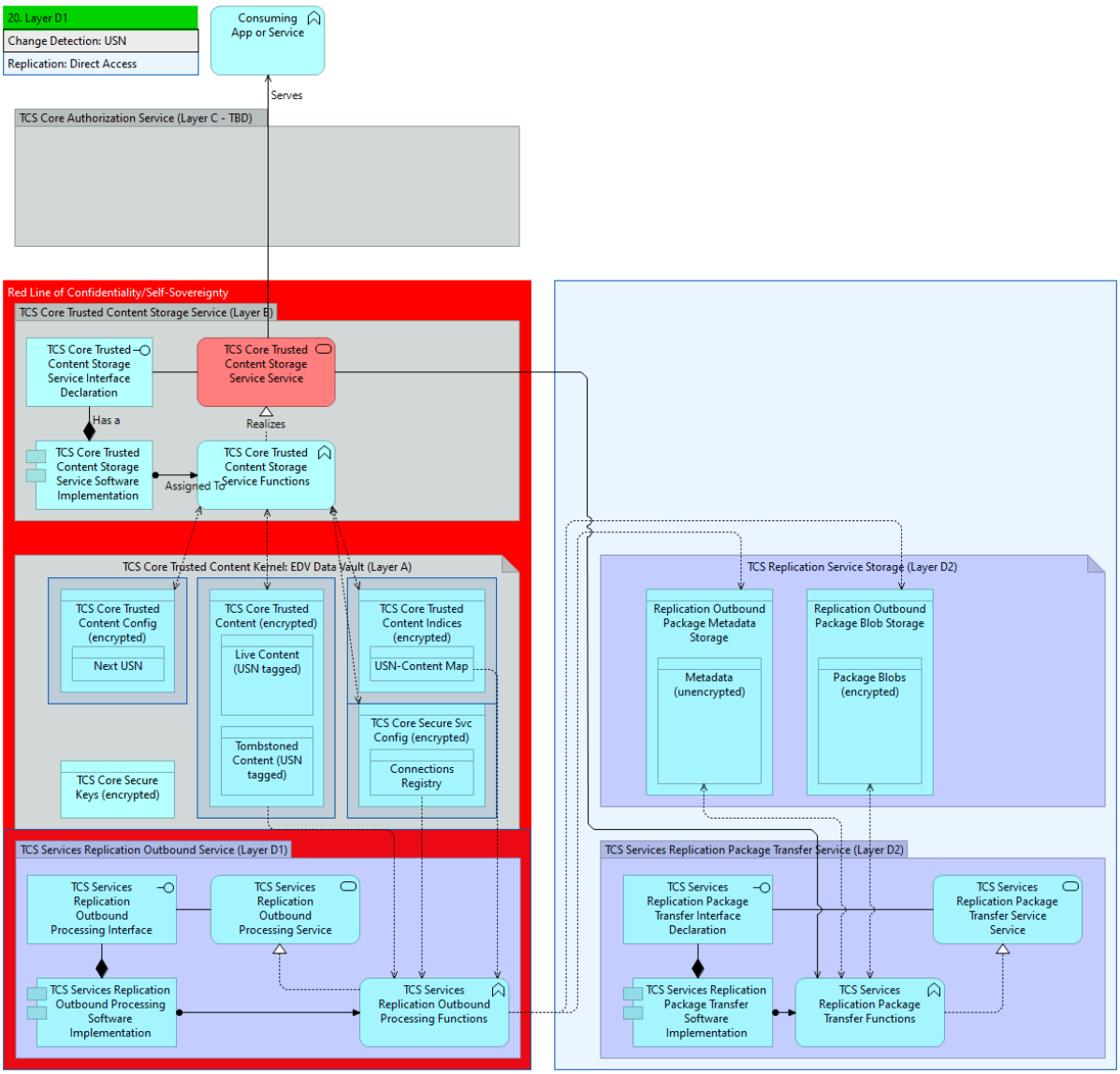


Figure 61. Replication Package Transfer Service: D2 Indirect (with D1 Outbound Replication, USN Model) [20]

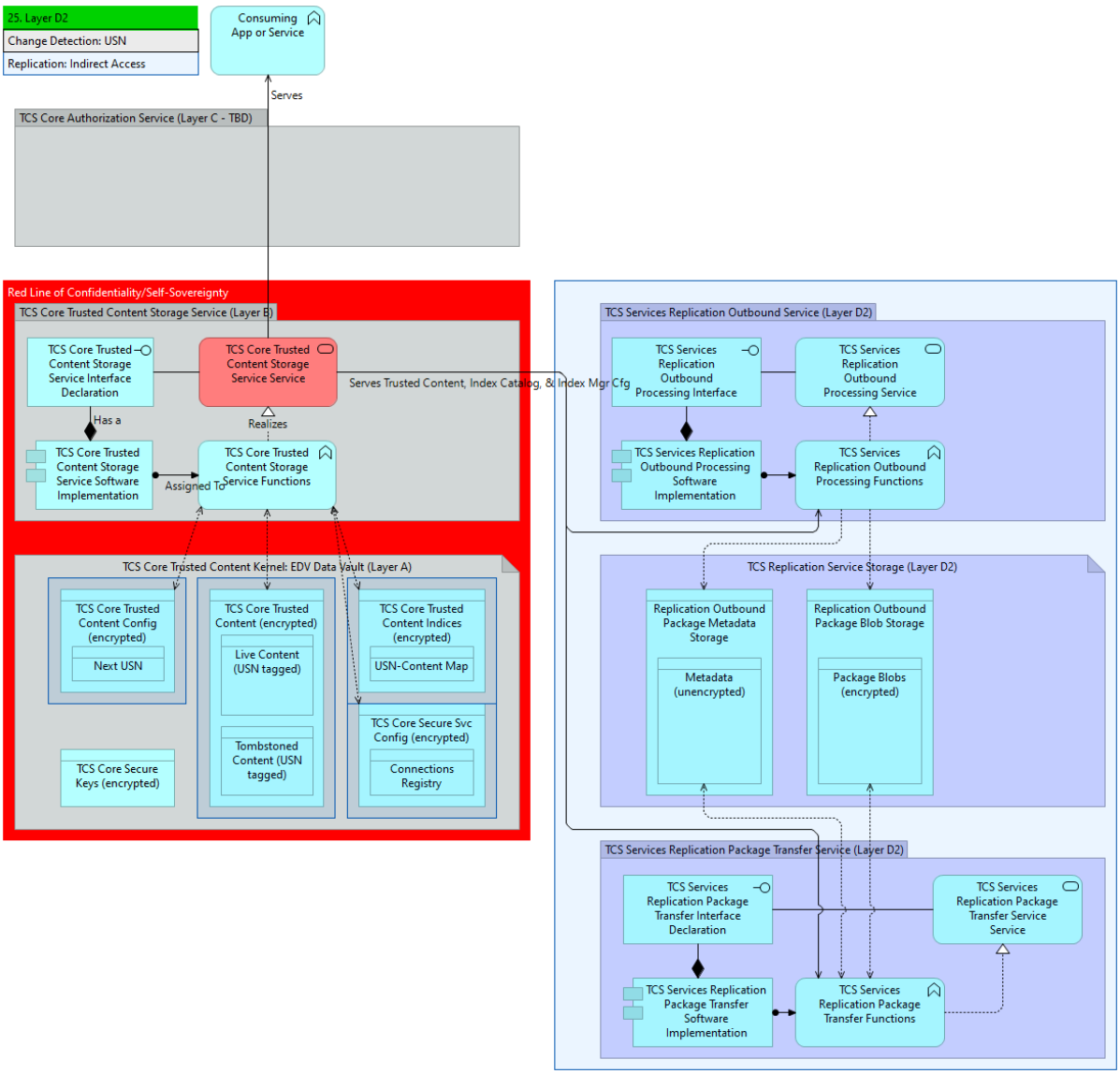


Figure 62. Replication Package Transfer Service: D2 Indirect (with D2 Outbound Processing Service, USN Model) [25]

Replication Inbound Processing Service

The ARMs for the architecture variations for the Replication Inbound Processing Service are described below.

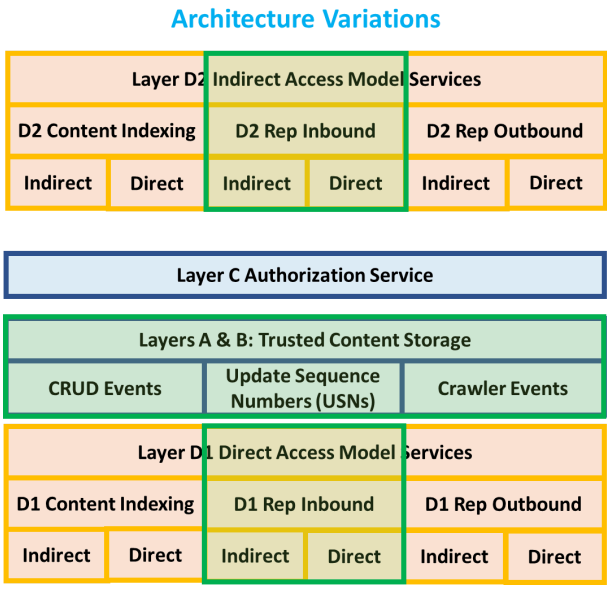


Figure 63. Replication Inbound Processing Service: Architecture Variations

ARMs for the following 3 Replication Inbound Processing Service architecture variations appear below.

Table 4. Replication Inbound Processing Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct		●19	
D2 Indirect	●22	●24	

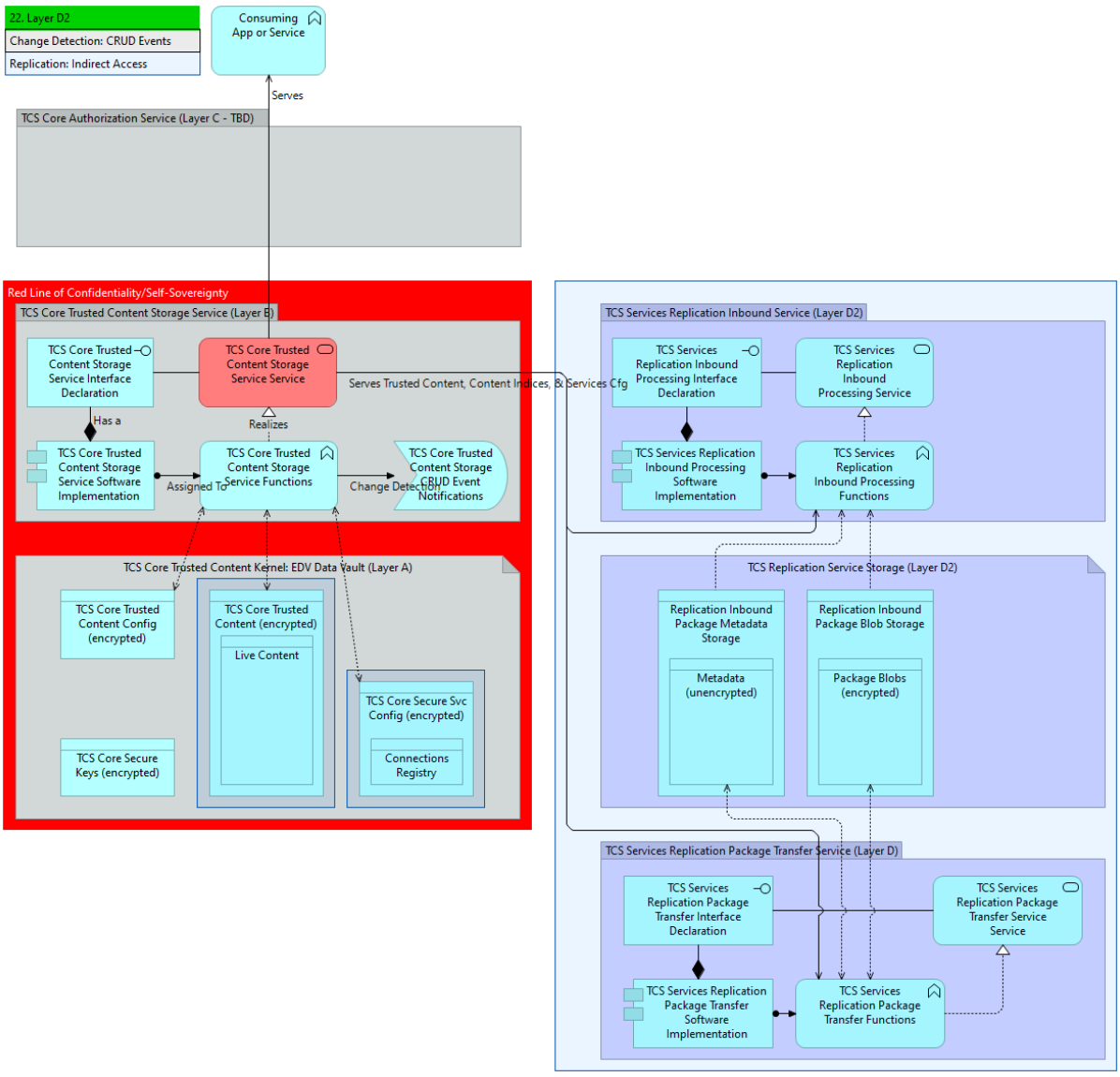


Figure 64. Replication Inbound Processing Service: D2 Indirect Access (CRUD Event Model Server) [22]

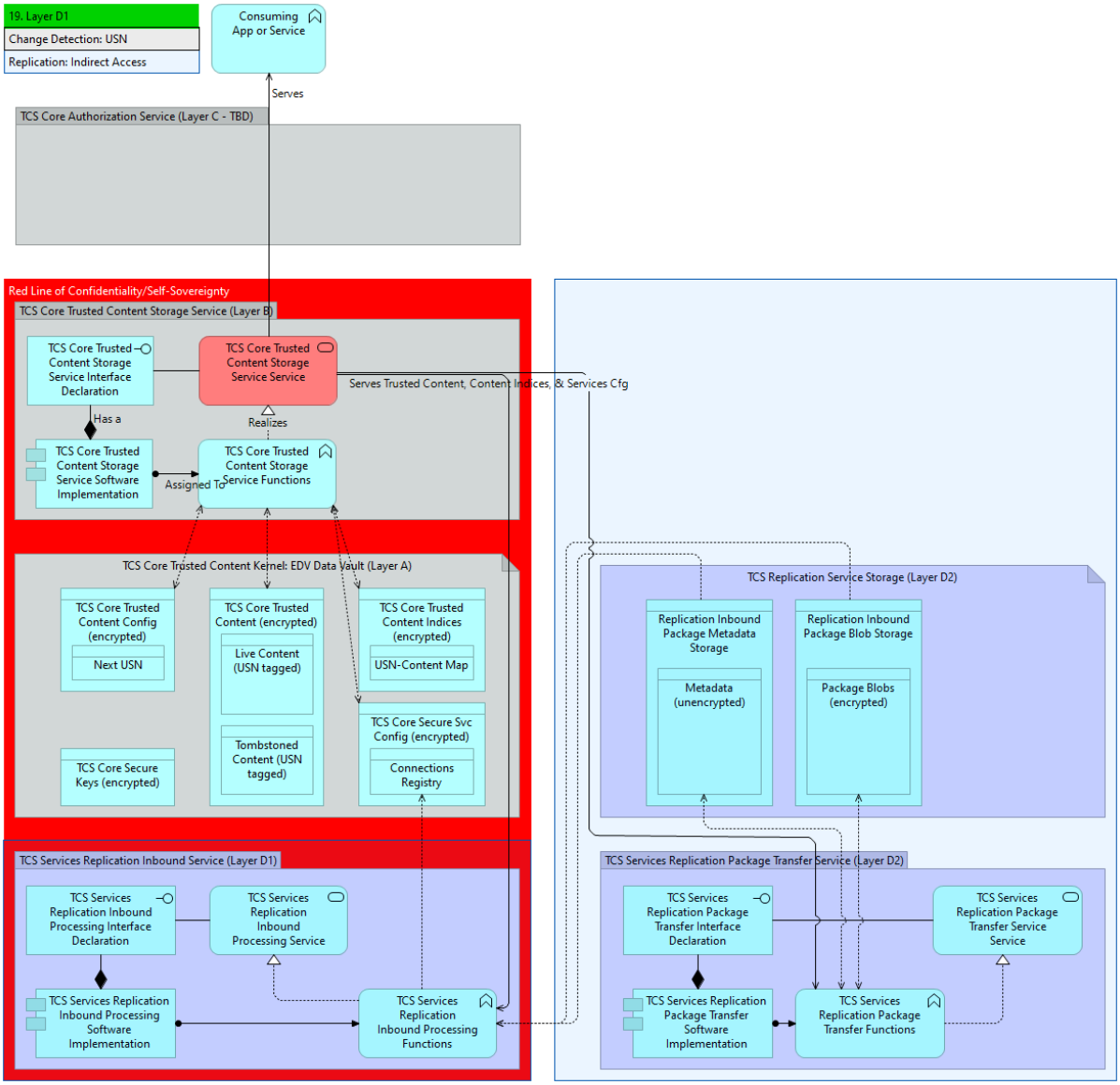


Figure 65. Replication Inbound Processing Service: D1 Direct Access (USN Model Server) [19]

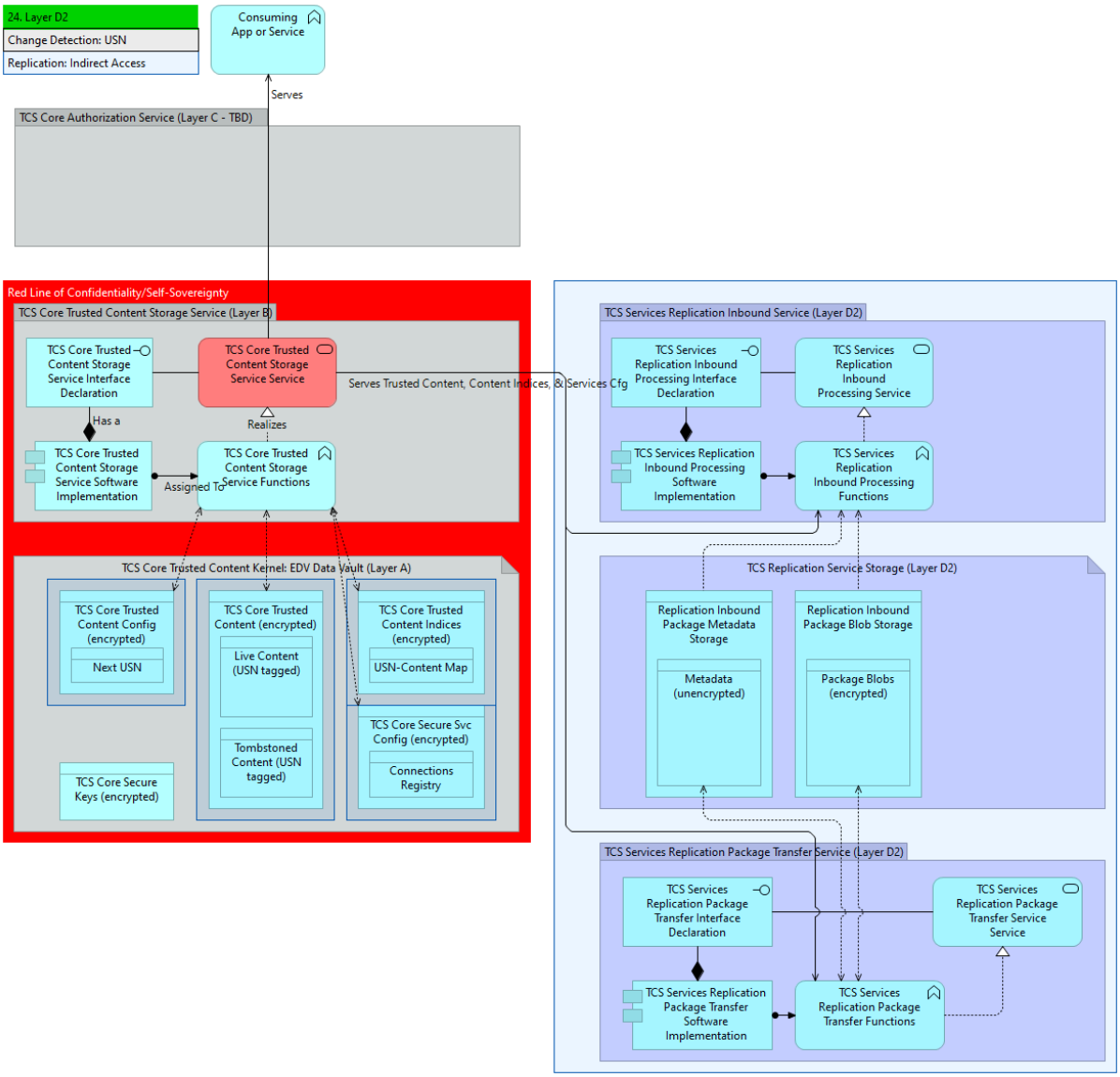


Figure 66. Replication Inbound Processing Service: D2 Indirect Access (USN Model Server) [24]

Layer D Indexing and Search Services

Layer D Indexing and Search Services includes:

i. Layer D Content Indexing (Index Manager) Service
ii. Layer D Search Query Processing Service

The ARMs for each of the Layer D Indexing and Search Services architecture variations (depicted in the following figure) are presented in the following sections.

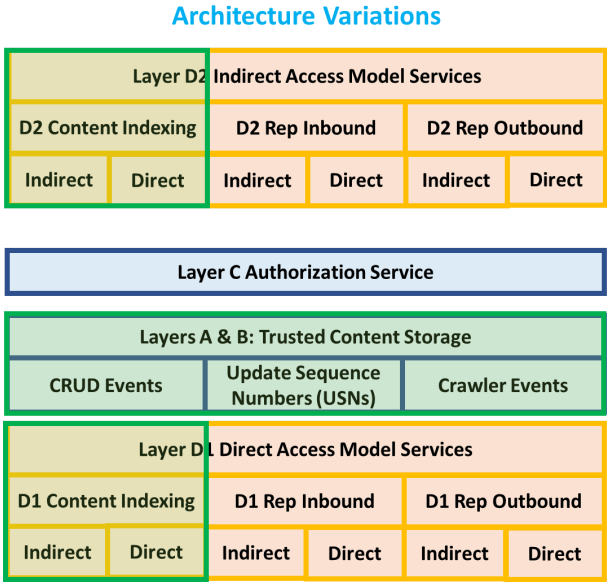


Figure 67. Layer D Indexing and Search Services: Architecture Variations

The Search Query Processing Service doesn't appear in the above diagram because it has a single architecture variation that is independent of the chosen Layer A and Layer B Content Change Detection/Tracking/Notification Model and Layer D Access Service Model.

Content Indexing Service (Index Manager) Service

The ARMs for the architecture variations for the Content Indexing Service (Index Manager) Service are described below.

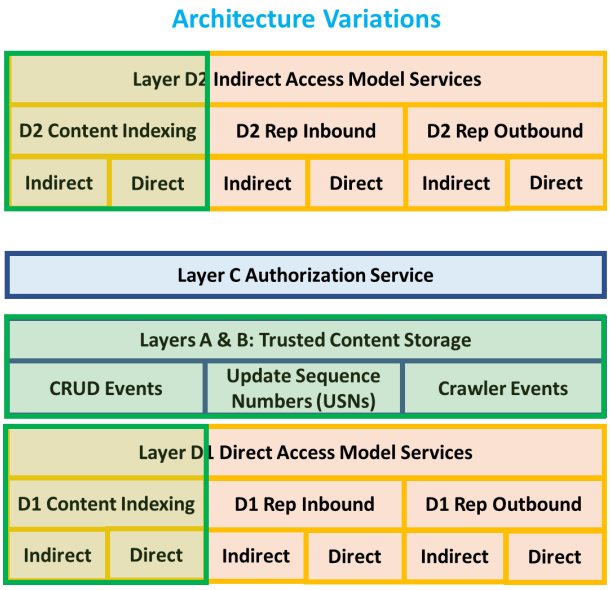


Figure 68. Content Indexing Service (Index Manager) Service: Architecture Variations

ARMs for the following 6 Content Indexing Service (Index Manager) Service architecture variations appear below.

Table 5. Content Indexing Service (Index Manager) Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct	●15,16	●17,18	●14
D2 Indirect	●21		

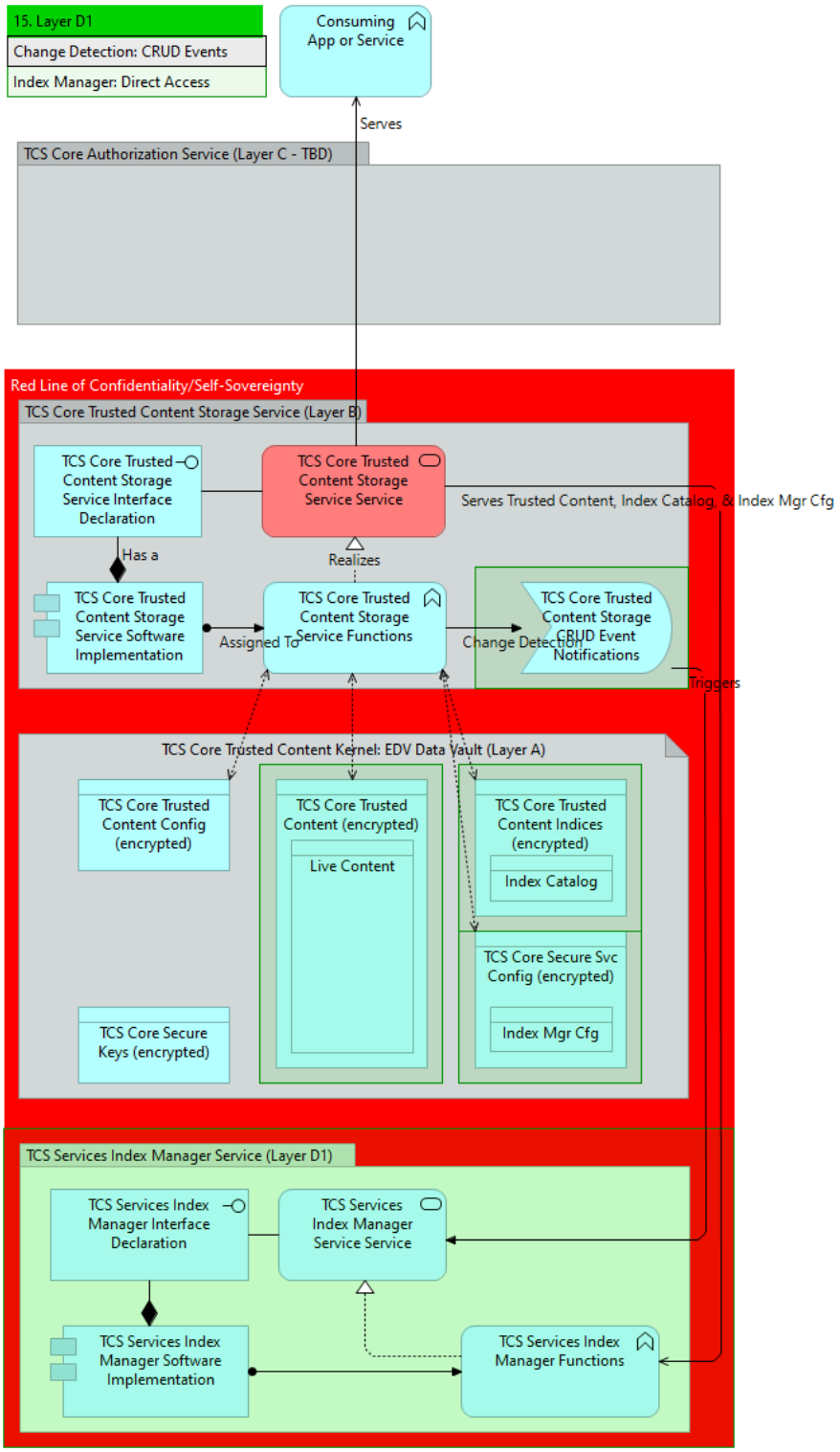


Figure 69. Content Indexing Service (Index Manager): CRUD Event Model, D1 Direct Access [15]

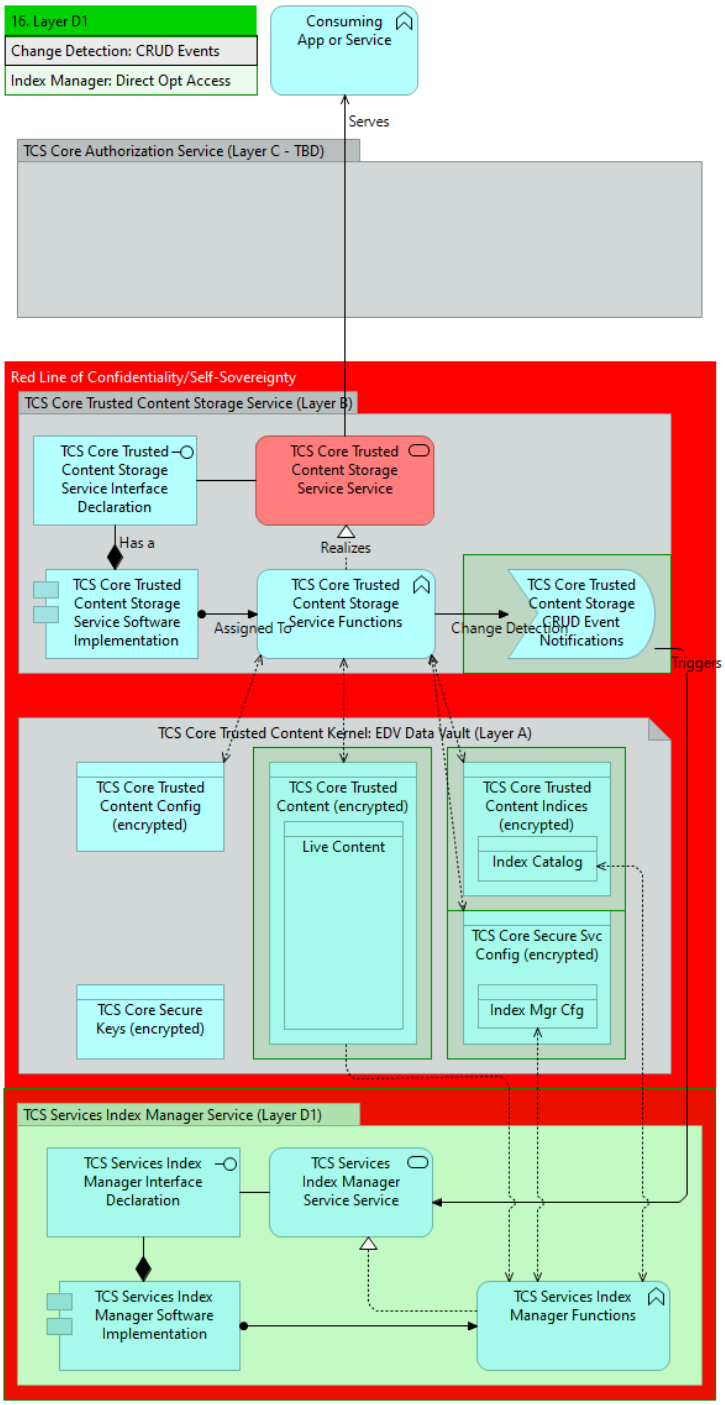


Figure 70. Content Indexing Service (Index Manager): CRUD Event Model, D1 Direct Access (optimized) [16]

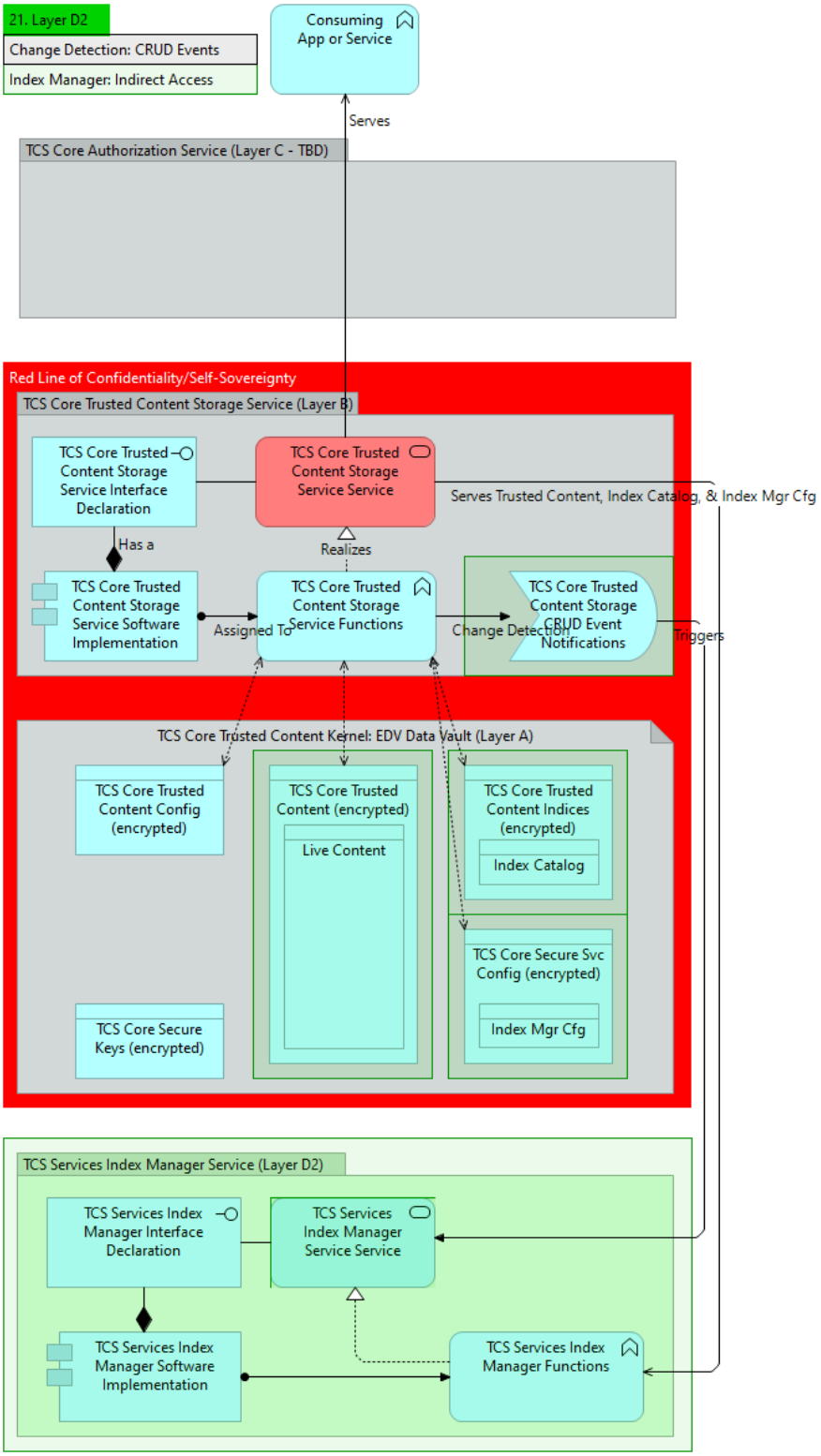


Figure 71. Content Indexing Service (Index Manager): CRUD Event Model, D2 Indirect Access [21]

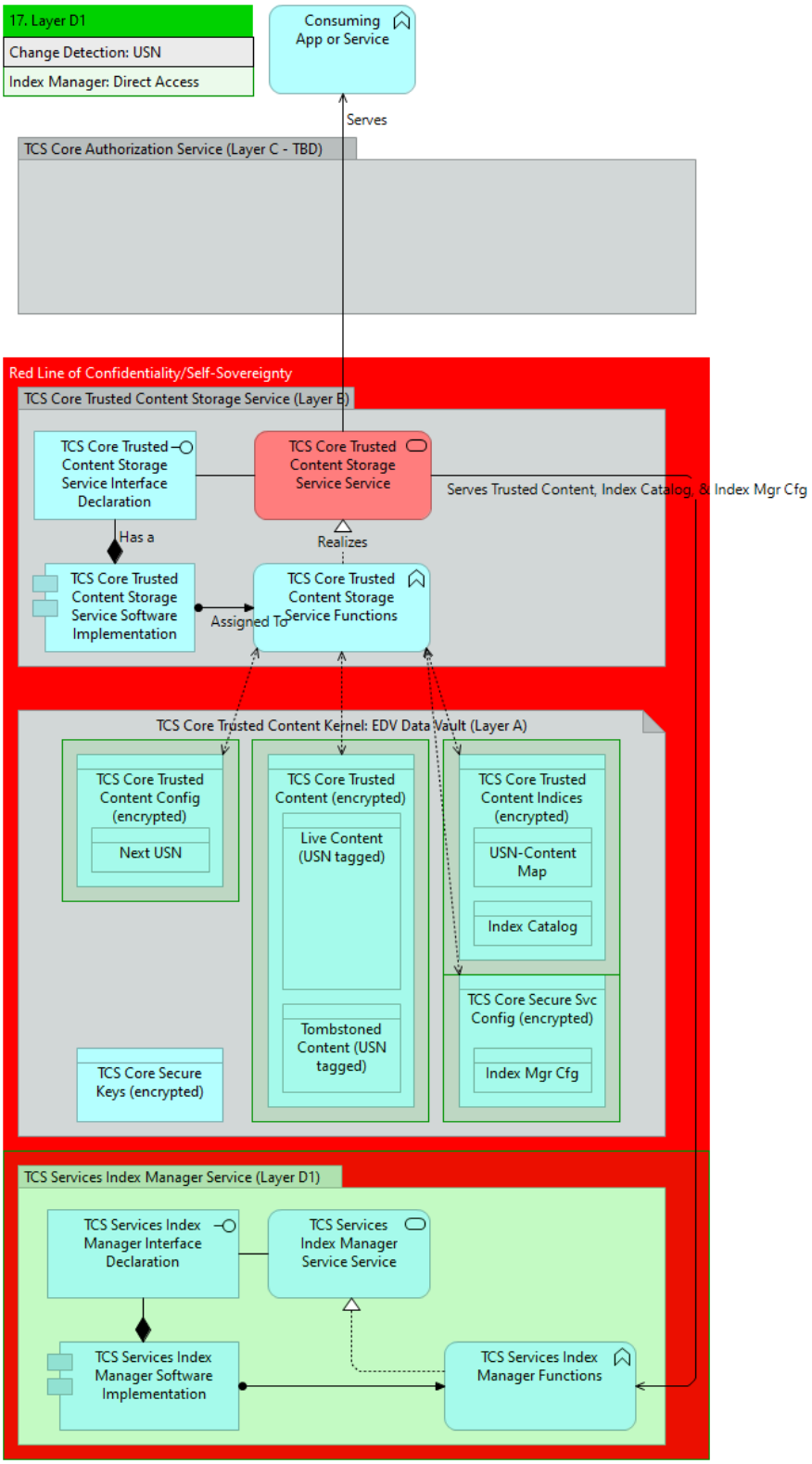


Figure 72. Content Indexing Service (Index Manager): USN Model, D1 Direct Access [17]

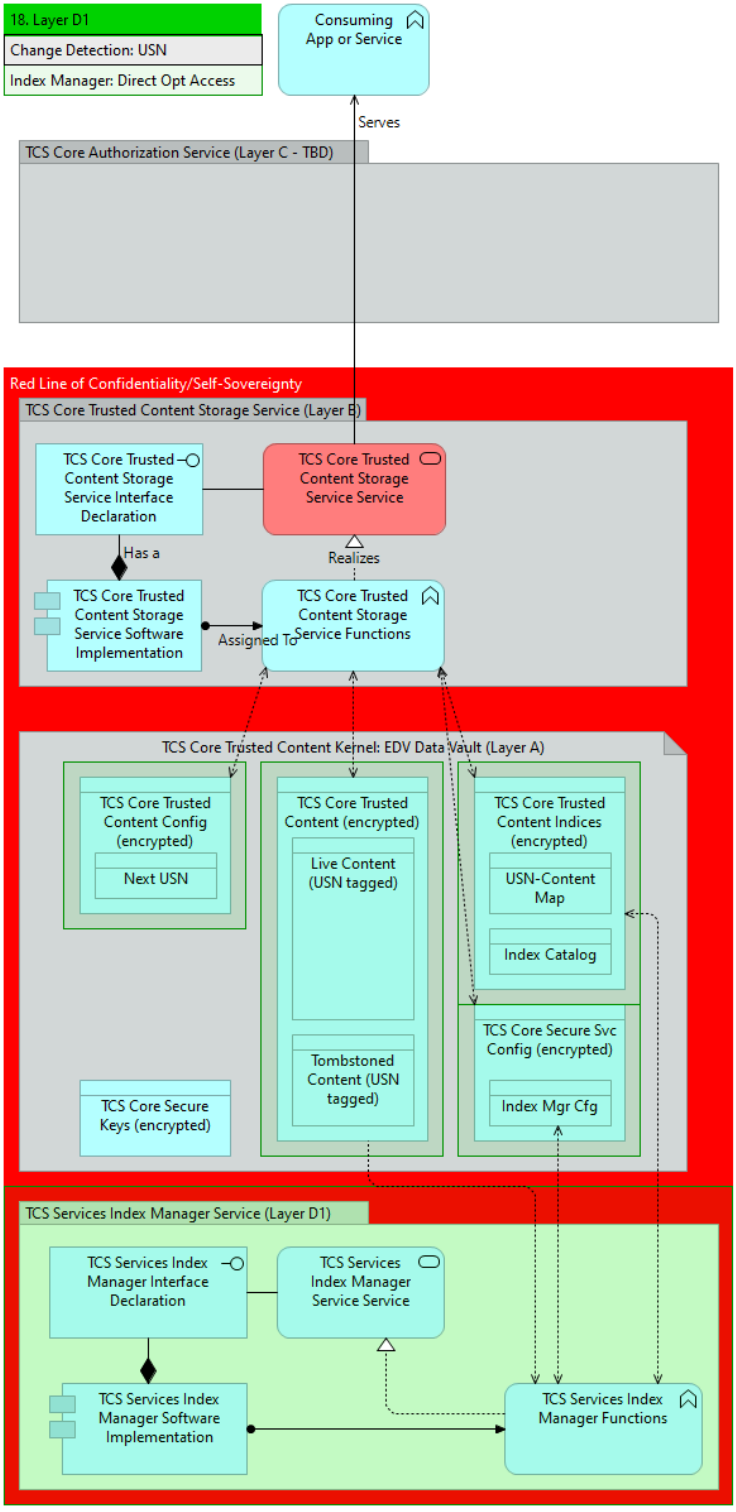


Figure 73. Content Indexing Service (Index Manager): USN Model, D1 Direct Access (optimized) [18]

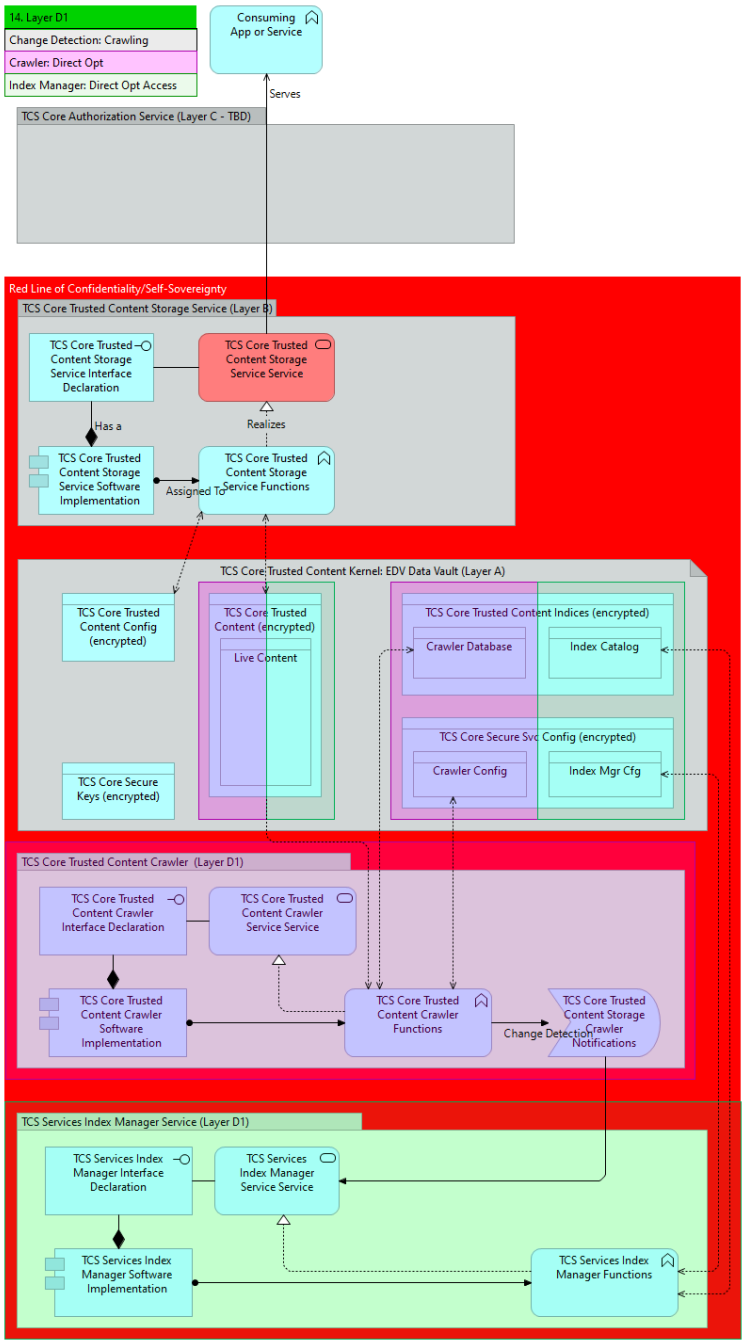


Figure 74. Content Indexing Service (Index Manager): Crawler Model, D1 Direct Access (optimized) [14]

Search Query Processing Service

The ARMs for the architecture variations for the Content Indexing Service (Index Manager) Service are described below.

Technically, there is 2 architecture variation for the Search Query Processing Service because:

- The service can run using either the Layer D1 Direct Access Service Model or Layer D2 Indirect Access Service Model
- The service design and deployment does not vary with the chosen Layer A and Layer B Content Change Detection/Tracking/Notification Model.
- The primary purpose of the Search Query Processing Service is to process search queries against the TCS Core Trusted Content Indices (of which, there is only one variation).

ARMs for 2 Search Query Processing Service scenarios appear below.

Table 6. Search Query Processing Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct	●[27]		
D2 Indirect	●[28]		

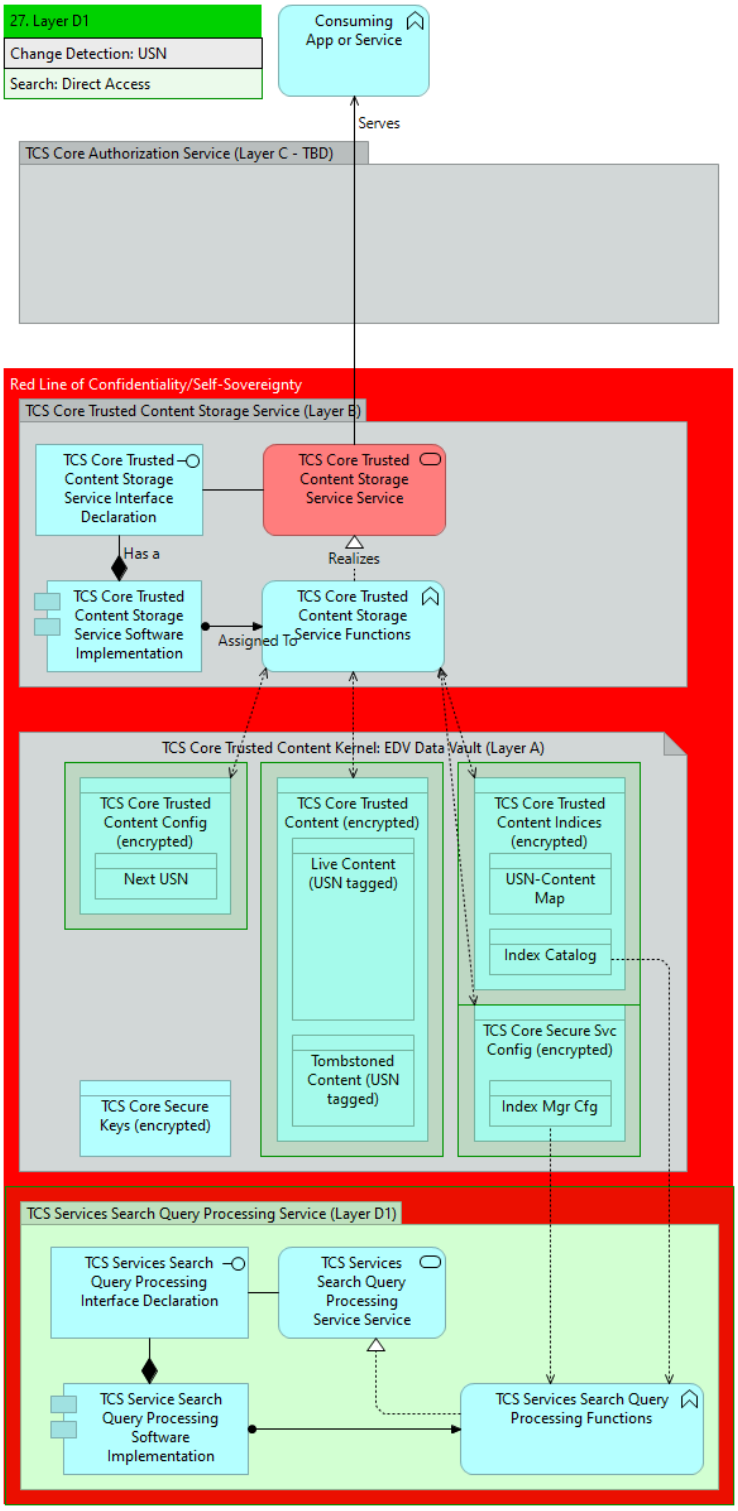


Figure 75. Search Query Processing: Crawler Model, D1 Direct Access (optimized) [27]

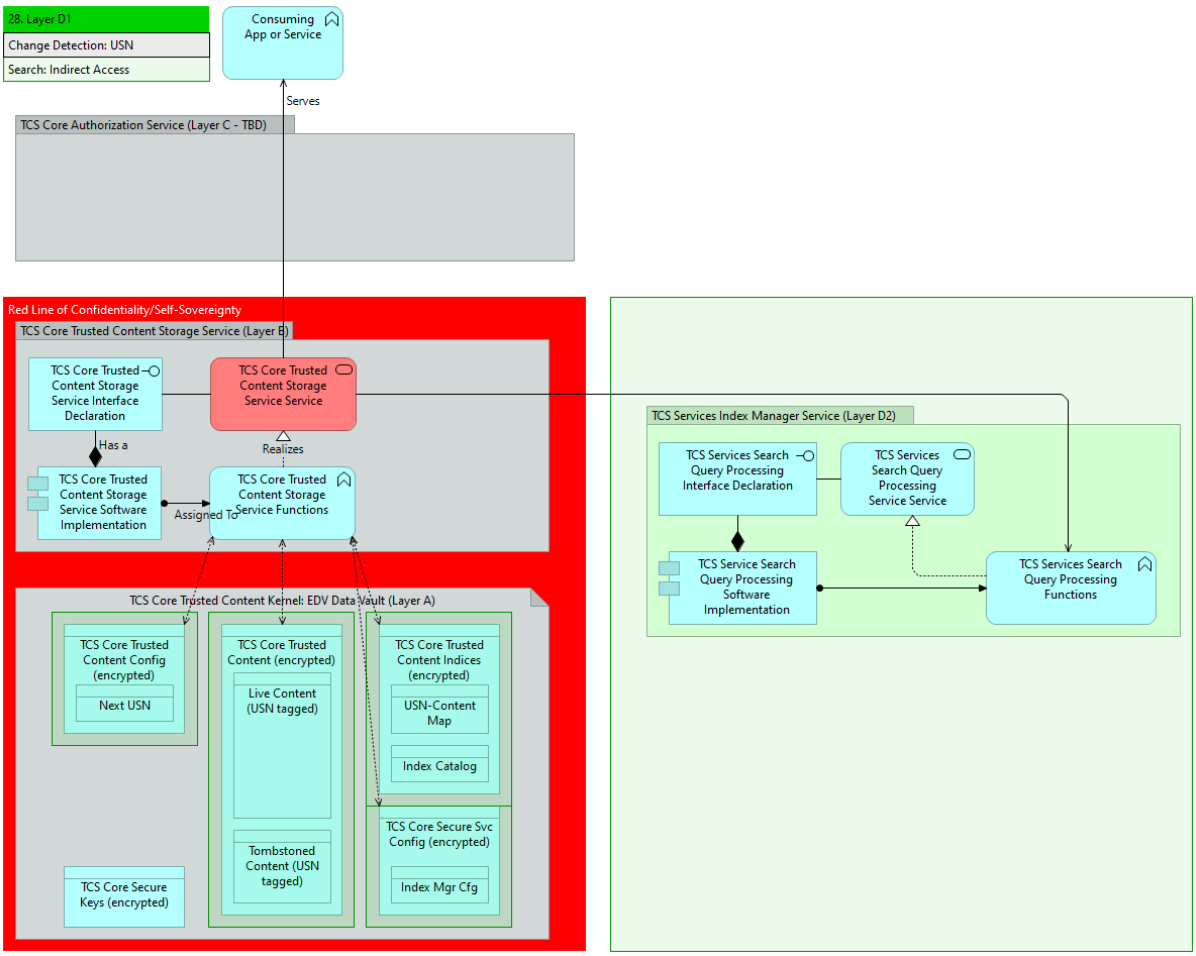


Figure 76. Search Query Processing: Crawler Model, D2 Indirect Access [28]

Architecture Assessments

The section documents the assessments of the architecture variations described in the previous section.

The following 3 versions of the TCS Stack architecture diagrams will provide the context for describing the architecture variations for each layer in the TCS (Extended) Stack.

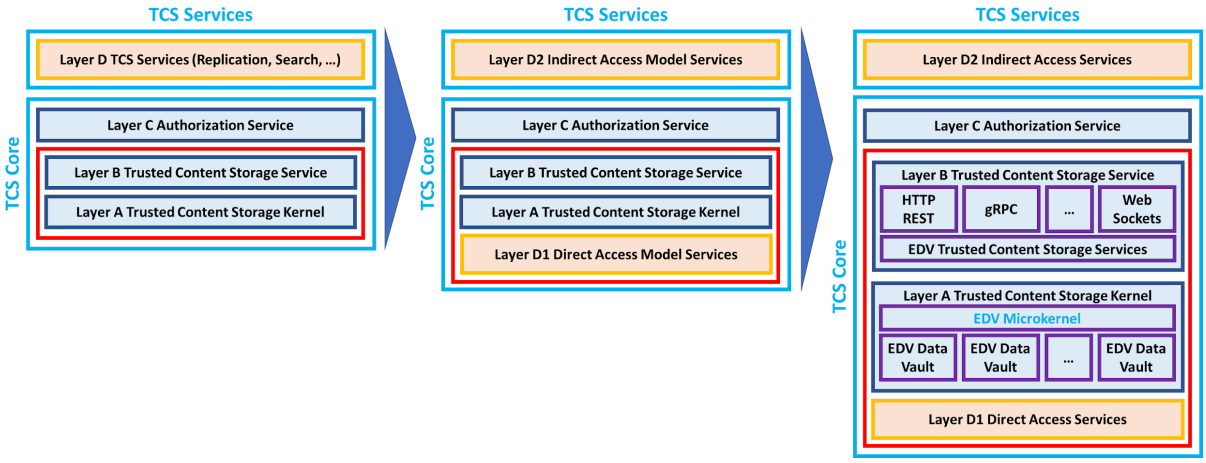


Figure 77. (a) TCS Stack, (b) TCS Extended Stack, (c) Detailed TCS Extended Stack

The same organization was also used in the previous Architecture Variations section and it will also be used for the Architecture Recommendations section of this document.

The rest of this section enumerates several alternative architecture reference models (ARMs) using the following dimensions:

1. TCS Core
    a. Layer A Trusted Content Storage Kernel
        i. Layer A Content Change Detection Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    b. Layer B Trusted Content Storage Service
        i. Layer B Content Change Tracking and Notification Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    c. Layer C Authorization Service[9]

[9] NOTE: TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper.

2. TCS Services Layer D Access Models
    a. Layer D1 Direct Access Model
    b. Layer D2 Indirect Access Model
3. TCS Services
    a. Layer D Replication Services
        i. Layer D Replication Outbound Processing Service
        ii. Layer D Replication Package Transfer Service
        iii. Layer D Replication Inbound Processing Service
    b. Layer D Indexing & Search Services
        i. Layer D Content Indexing Service
        ii. Layer D Search Query Processing Service

The TCS Stack architecture variations landscape is depicted in the following diagram.

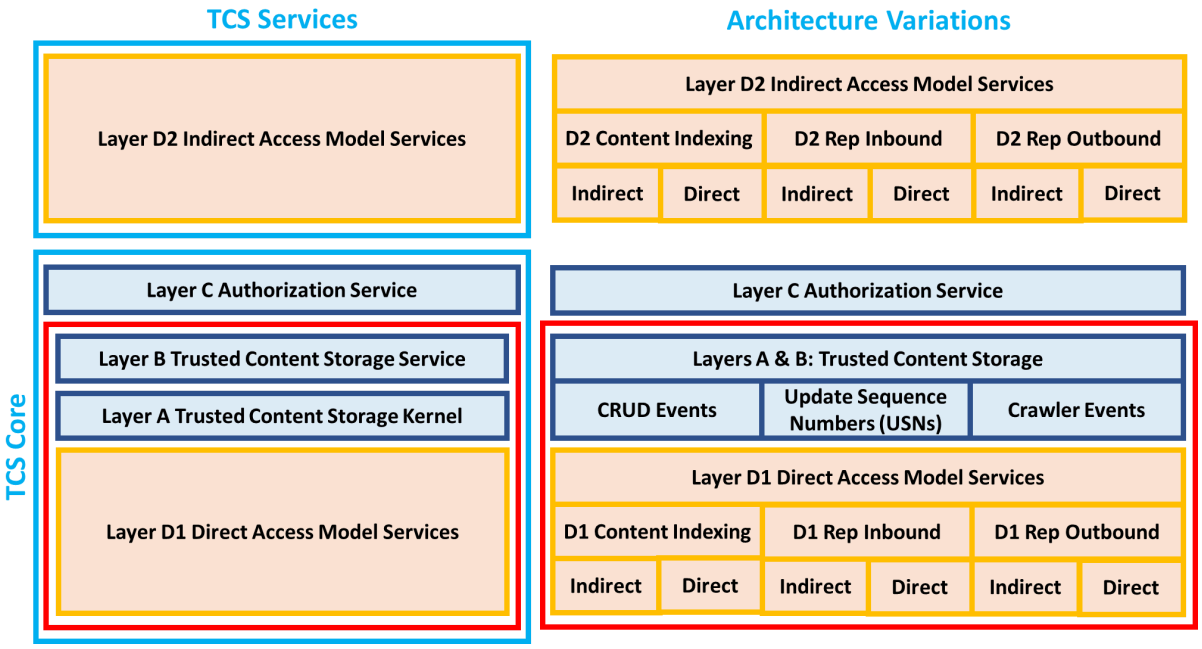


Figure 78. TCS Stack Architecture Variations Landscape

The high-order determinant for classifying the variations includes the following:

- Content Change Detection/Tracking/Detection Models
    - Layer A Content Change Detection Models
    - Layer B Content Change Tracking and Notification Models

The Content Change Detection Model for Layer A Trusted Content Storage Kernel and the Content Change Tracking and Notification Model chosen for Layer B Trusted Content Storage Service are the primary determinates of the architecture variations for the other dimensions.

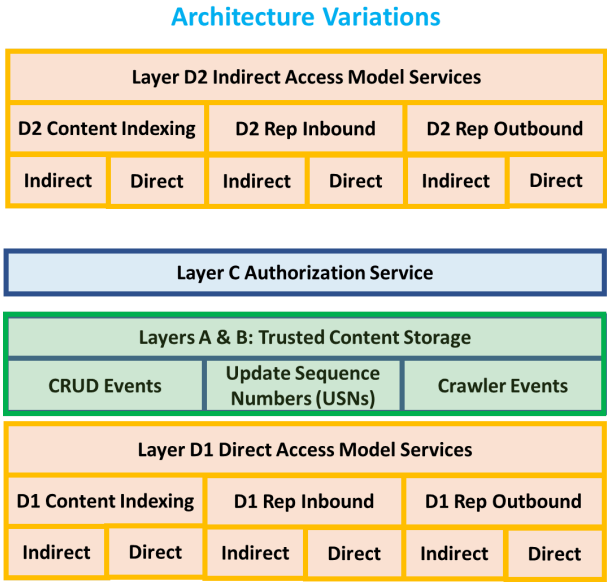


Figure 79. Layer B Content Change Detection/Tracking/Notification

In particular, these primary TCS Services Layer D Services:

- Layer D Replication Outbound Processing Service
- Layer D Replication Package Transfer Service
- Layer D Replication Inbound Processing Service
- Layer D Content Indexing Service

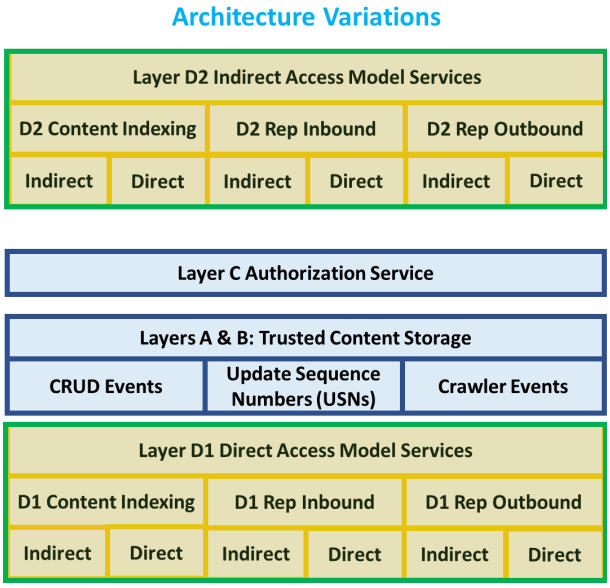


Figure 80. Primary Layer D Services

When the cross-product of the primary TCS Services Layer D Services and the Layer B Content Change Detection/Tracking/Notification variations is performed, a very large total number of possible architecture variations is the result (as depicted below).

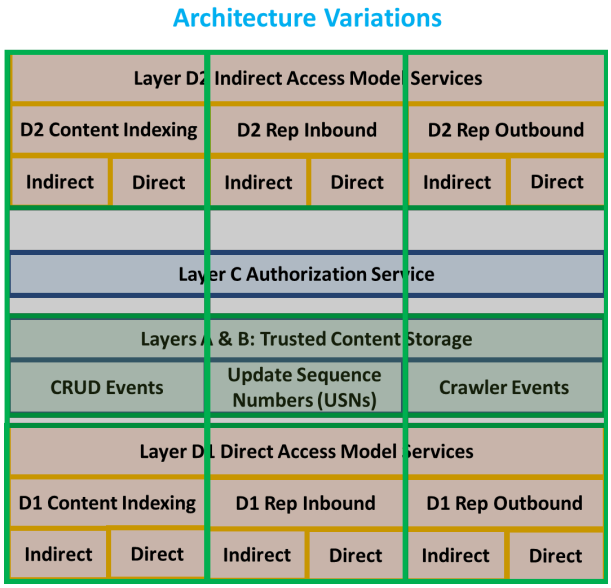


Figure 81. Layer B Content Change Detection/Tracking/Notification Variations X Layer D Services

The remainder of this section on Architecture Variations is framed by the above dimensions.

TCS Core Layer A Architecture Assessments

There are 2 variations for TCS Core Layer A (Trusted Content Storage Kernel):

- EDV Microkernel
- Heterogeneous Multi-vaults/Server Instance Support

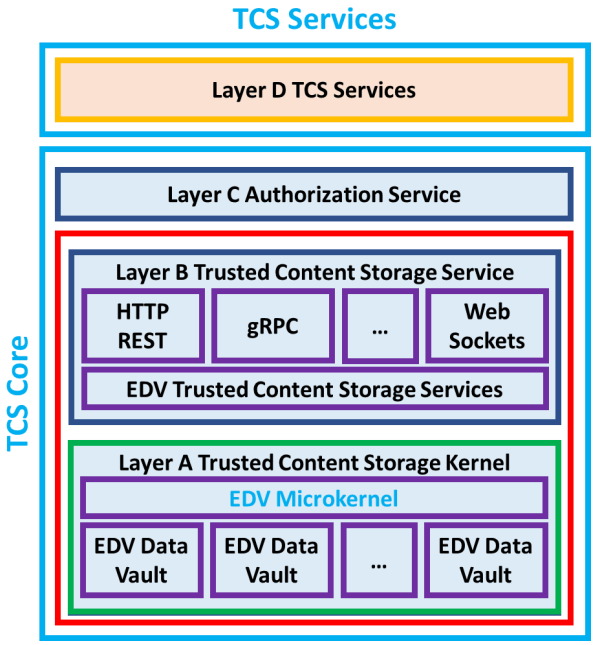


Figure 82. TCS Core Layer A Architecture Variations

Layer A Trusted Content Storage Kernel Assessments

The formal name for Layer A is TCS Core Layer A Trusted Content Storage Kernel. Its primary subcomponents are:

- EDV Microkernel
- EDV Data Vaults

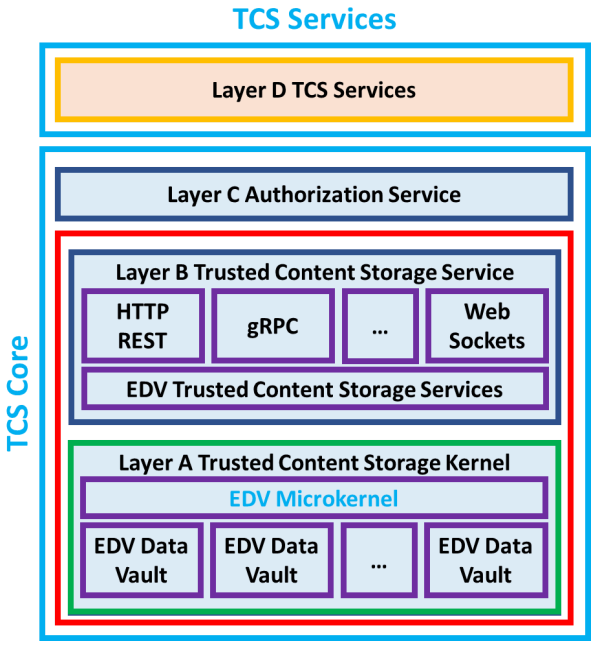


Figure 83. TCS Core Layer A Architecture Variations: EDV Microkernel and EDV Data Vaults

EDV Microkernel Architecture: Assessment

For the EDV Microkernel architecture, there are no architecture variations along the defined dimensions.

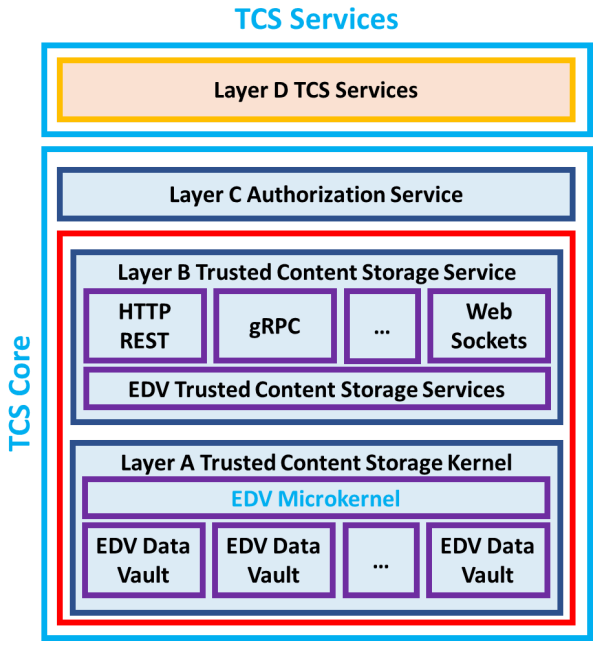


Figure 84. TCS Stack Layer A EDV Microkernel Architecture: Multiple EDV Data Vaults per EDV Server Instance

Considerations, Principles, and Requirements

Other variations include:

1. Adoption of the CARUMDRIGF operation set (https://github.com/decentralized-identity/confidential-storage/issues/172)
2. Choice of Content Change Detection (and Tracking and Notification) model
3. Approach for propagating/raising Content Change Detection events to Layer B Trusted Content Storage Service (dependent on the overall Content Change Detection Model chosen)
4. Whether to expose the EDV Microkernel interface as the lowest level public interface
5. Whether to support Homogeneous Multi-vaults per Server Instance or Heterogeneous Multi-vaults per Server Instance

Prior Art

- Definition: Microkernel (https://en.wikipedia.org/wiki/Microkernel)

Assessment

- Support the CARUMDRIGF operation set (https://github.com/decentralized-identity/confidential-storage/issues/172)
- Expose the EDV Microkernel interface as the lowest level public interface
- support Homogeneous Multi-vaults per Server Instance

Layer A Content Change Detection: Assessments

ISSUE: In this version of the document, Content Change Detection is assumed to be a Layer A Trusted Content Storage Kernel function (although the ARM diagrams may not all reflect this). Content Change Tracking and Notification are relegated to the Layer B Trusted Content Storage Service.

Content Change Detection's responsibility is to detect or otherwise sense changes being made to any content being made in an EDV Data Vault. This includes changes to configuration data, indices, etc. in addition to everyday content resources stored in a data vault. In the specific context of the TCS Stack, this also includes Read operations as well as both successful operation attempts and unsuccessful operation attempts.

Variations

There are 3 variations (or models) for Layer B Content Change Detection/Tracking/Notification as depicted below.

- CRUD Events
- Update Sequence Numbers (USNs)
- Crawler Events

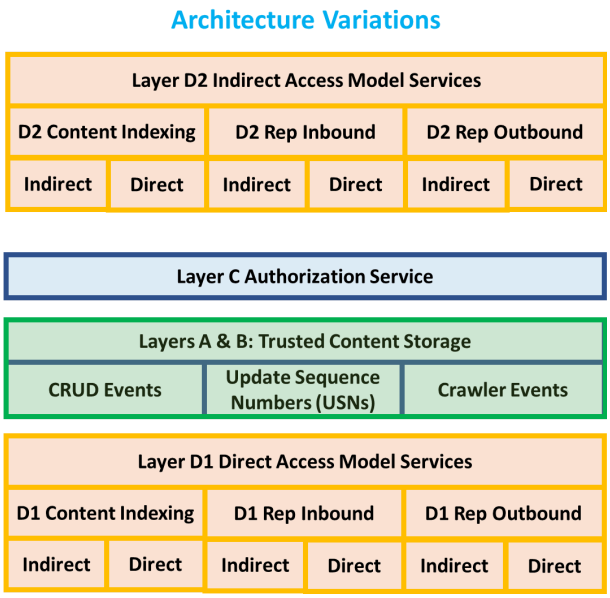


Figure 85. Layer B Content Change Detection/Tracking/Notification

Considerations, Principles, Requirements, and Assumptions

The considerations for choosing the overall Content Change Detection, Tracking, and Notification model for Layer A Trusted Content Storage Kernel and Layer B Trusted Content Storage Service in the TCS Stack architecture are manyfold. These include:

1. Data model
2. Data storage
3. Execution performance

4. Data vault recovery (playback replication from another replica)
5. Support for the Replication Pipeline and Layer D Replication Services (specifically, the Replication Outbound Processing Service)
6. Support for Layer D Content Indexing (Index Manager) Service
7. Support for Layer D1 Secure Logging Service
8. Support for additional Layer D Services (following the #OpenToInnovation principle (https://hyperonomy.com/2019/03/12/internet-protocols-and-standards-not-only-need-to-be-open-but-more-importantly-open-to-innovation/))

Prior Art

- See the bullets in each section below

Layer A Content Change Detection: CRUD Events Model Variation: Assessment

The CRUD Events Model variation of the Layer A Trusted Content Storage Kernel is illustrated in the following figure.

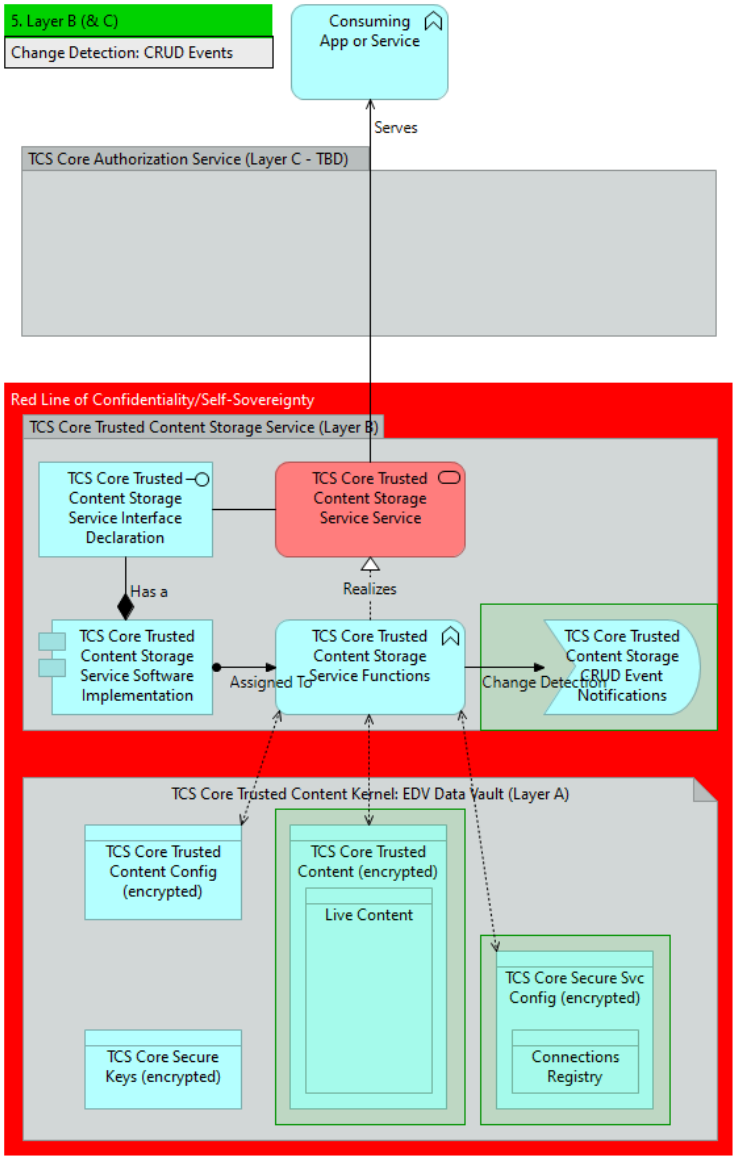


Figure 86. Layer A Content Change Detection: CRUD Events Model [5]

Considerations, Principles, and Requirements

- See the previous section's Considerations, Principles, and Requirements

Prior Art

- CRUD Event Model: Syntergy Replicator for SharePoint – see Appendix A

Assessment

Consideration	Pros	Cons
1	- No data model requirements	
2	- No data storage requirements	
3	- Moderate to minimal performance impact	
4		- No implicit or explicit support for EDV recovery
5	- Strong support for Replication Outbound Processing Service	
6	- Strong support for Content Indexing (Index Manager) Service	
7	- Moderate support for the Secure Logging Service (which needs to be supported more directly in the EDV Microkernel)	
8	- Strong support for additional Layer D Services	

In summary, the final (interim) assessment is the CRUD Events Model is a moderate to very strong valued architecture variation.

Layer A Content Change Detection/Tracking: USN Model Variation: Assessment

The USN Events Model variation of the Layer A Trusted Content Storage Kernel is illustrated in the following figure.

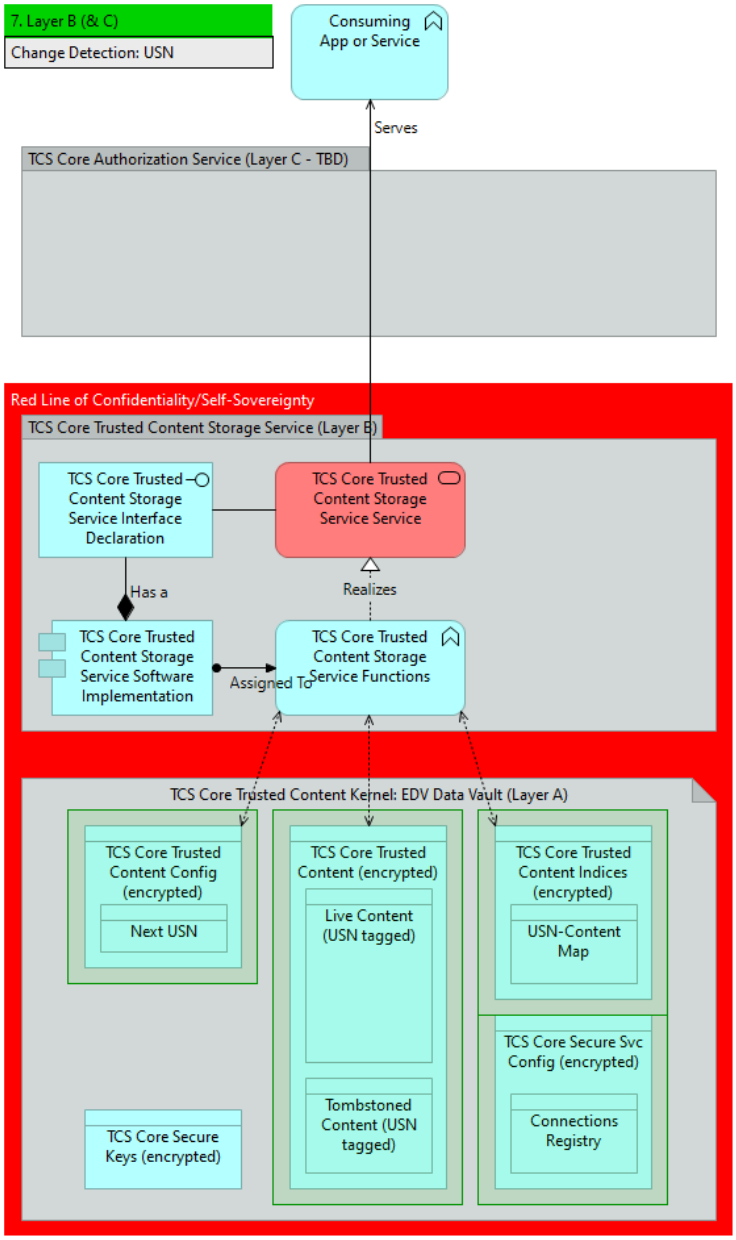


Figure 87. Layer A Content Change Detection: USN Model [7]

Considerations, Principles, and Requirements

- See the previous section's Considerations, Principles, and Requirements

Prior Art

- Microsoft Active Directory – see Appendix A

Assessment

Consideration	Pros	Cons
1	- Moderate data model requirements (Update Sequence Numbers unique to each EDV data vault)	
2	- Moderate data storage requirements (Update Sequence Number for each resource in each EDV data vault)	
3	- Moderate to minimal performance impact	
4	- Strong support for data vault recovery	
5	- Strong support for Replication Outbound Processing Service	
6	- Strong support for Content Indexing (Index Manager) Service	
7	- Moderate support for the Secure Logging Service (which needs to be supported more directly in the EDV Microkernel)	
8	- Strong support for additional Layer D Services	

In summary, the final (interim) assessment is the USN Model is a strong to very strong valued architecture variation.

Layer A Content Change Detection: Crawler Model Variation: Assessment

The Crawler Model variation of the Layer A Trusted Content Storage Kernel is illustrated in the following figure.

NOTE: The Crawler Model for Content Change Detection requires no specific support in the Layer A Trusted Content Kernel (nor the EDV Kernel).

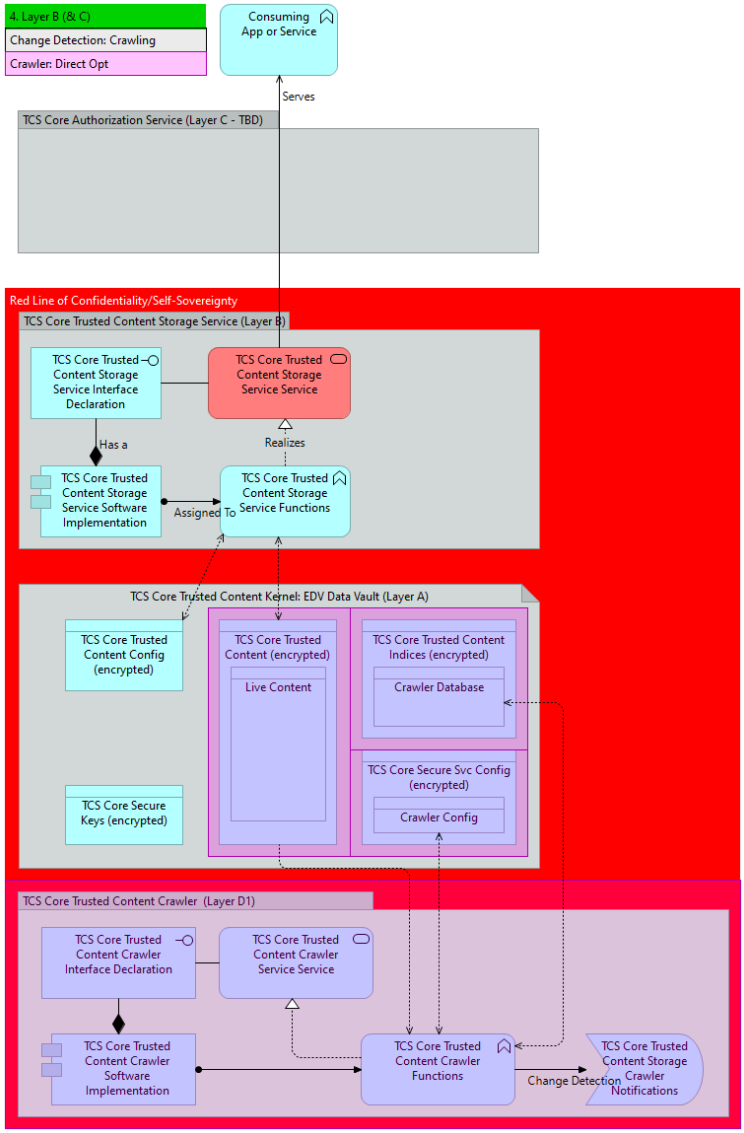


Figure 88. Layer B Content Change Detection/Tracking/Notification: Crawler Model [4]

Considerations, Principles, and Requirements

- See the previous section's Considerations, Principles, and Requirements

Prior Art

- Microsoft SharePoint Search – see Appendix A

Assessment

Consideration	Pros	Cons
1	- Moderately data model for the USN-Content Index	
2	- Moderate to large data storage requirements for the USN-Content Index	
3	- Moderate to high-performance impact due to Crawler Service	
4	- Moderate support for data vault recovery	
5	- Moderate support for Replication Outbound Processing Service	
6	- Moderate support for Content Indexing (Index Manager) Service	
7	- No support for Secure Logging Service (which needs to be supported more directly in the EDV Microkernel)	
8	- Low to moderate support for additional Layer D Services	

In summary, the final (interim) assessment is the Crawler Model is a low to moderate valued architecture variation.

Heterogeneous Multi-vaults/Server Instance Support: Assessment

Variations

The Heterogeneous Multi-vaults/Server Instance Support variations of the Layer A Trusted Content Storage Kernel are illustrated in the following figure.

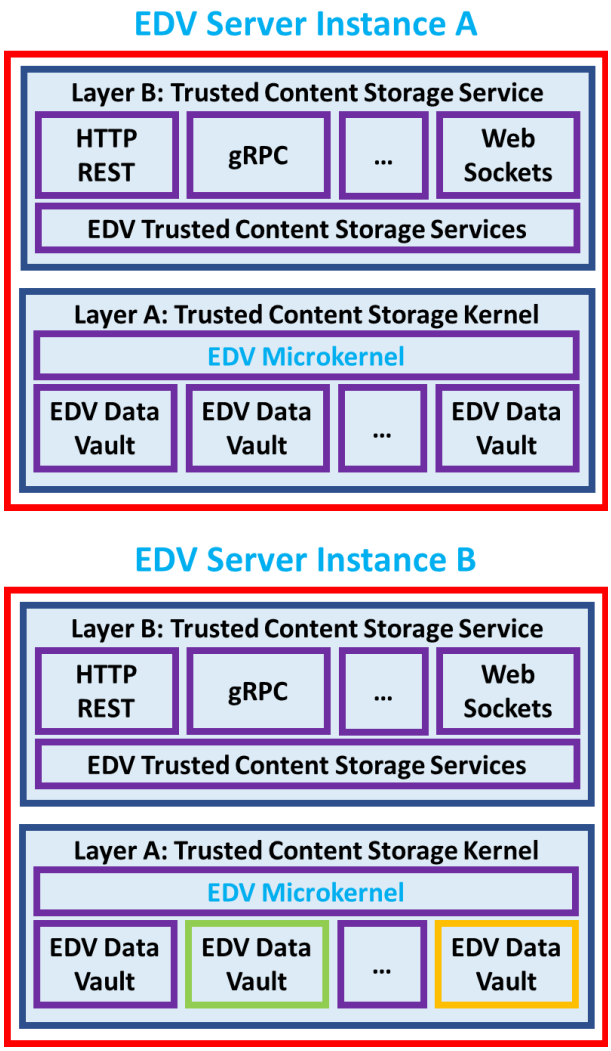


Figure 89. (a) Homogeneous Multi-vaults/Server Instance and (b) Heterogenous Multi-vaults/Server Instance

Considerations, Principles, and Requirements

1. Support for Homogeneous Multi-vaults per Server Instance
2. Support for Heterogenous Multi-vaults per Server Instance
3. Support for ACID transaction scopes (single data vault/single server instance scenarios)
    a. A change to a single resource in a single physical data vault on a single server instance
    b. A batch of multiple changes to a single resource in a single physical data vault on a single server instance

    c. A batch of multiple changes to multiple resources in a single physical data vault on a single server instance
4. Support for ACID transaction scopes (multiple data vault/single server instance scenarios)
    a. A batch of multiple changes to multiple resources across multiple physical data vaults on a single server instance
5. Support for ACID transaction scopes (multi-server instance scenarios)
    a. A batch of multiple changes to multiple resources across multiple physical data vaults across multiple server instances

Prior Art

- Microsoft SQL Server MDF Database Files

Assessment

Consideration	Pros	Cons
1	- Easier/simpler to support Homogeneous Multi-vaults per Server Instance	- Limiting support to Homogeneous Multi-vaults per Server Instance contributes to migration and upgrade challenges for service providers
2	- Simplifies many upgrade/migration scenarios - Eliminates the need for everyone to be running the same EDV data vault technology and same version of the technology	- More complex to support Homogeneous Multi-vaults per Server Instance
3	Single data vault/single server instance scenarios are easier/simpler	
4	Multiple data vaults/single server instance scenarios are more complex	
5		Multiple data vault/multiple server instance scenarios are even more complex

In summary, the final (interim) assessment for version 1 of the Trusted Content Storage Architecture (TCS Stack) is to:

- Support the EDV Microkernel Model
- Support 1. Homogeneous Multi-vaults per Server Instance
- Support 3b. A batch of multiple changes to a single resource in a single physical data vault on a single server instance

TCS Core Layer B Architecture Assessments

ISSUE: In this version of the document, Content Change Detection is assumed to be a Layer A Trusted Content Storage Kernel function (although the ARM diagrams may not all reflect this). Content Change Tracking and Notification are relegated to the Layer B Trusted Content Storage Service.

Once a change has been detected, Content Change Tracking is responsible for tracking the change by either: (a) raising or triggering an event, (b) using an update sequence number (USN) to track changes on a resource-by-resource basis, or (c) recording the event in a secondary container in the data vault.

In the TCS Stack, the responsibility for Content Change Tracking is assigned to the Layer B Trusted Content Storage Service.

Once a change has been detected and tracked, Content Notification is responsible for notifying TCS Services Layer D Services so that the change can be acted upon. These may be Layer D1 Services with direct access to the data vault via the EDV Microkernel or Layer D2 Services with indirect access to the data vault via the Layer B Trusted Content Storage Service Service.

In the TCS Stack, the responsibility for Content Change Notification is assigned to the Layer B Trusted Content Storage Service.

Layer B Trusted Content Storage Service Assessment

Variations

There are 2 variations for TCS Core Layer B (Trusted Content Storage Service):

- Content Detection, Tracking, and Notification Models
  - CRUD Events
  - USNs
  - Crawler based
- Supported Protocol Endpoints, for example,
  - HTTP REST
  - gRPC
  - Web Sockets
  - Etc.

These variations are illustrated below.

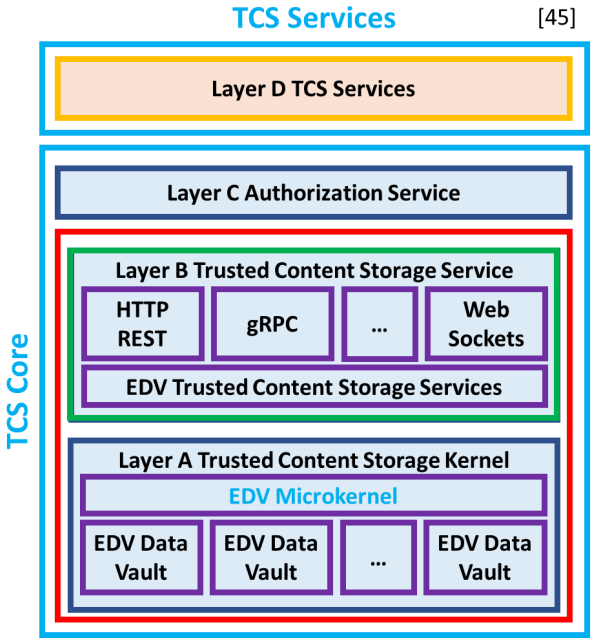


Figure 90. TCS Core Layer B Variations [45]

Considerations, Principles, and Requirements

1. Separation of responsibilities: Layer A TCS Trusted Content Storage Kernel vs. Layer B Trusted Content Service
2. Layer B Trusted Content Storage Service to host protocol endpoints (e.g. HTTP REST, gRPC, Web Sockets, etc.)
3. Choice of Content Change Detection (and Tracking and Notification) model
   a. CRUD Events
   b. USNs

c. Crawler based

4. Approach for propagating/raising Content Change Detection events to Layer B Trusted Content Storage Service (dependent on the overall Content Change Detection Model chosen)

Prior Art

- No referenceable prior art available at time of writing

Assessment

Consideration	Pros	Cons
1	- n/a	- n/a
2	- Clean separation of protocol endpoints from the EDV Microkernel interface - Easier/simpler to extend/add new protocols	
3	- Refer to Layer A Content Change Detection, Tracking, & Notification variations discuss (see above)	
4	Refer to Layer A Content Change Detection, Tracking, & Notification variations discuss (see above)	

In summary, the final (interim) assessment is to support:

- The separation of Layer B Trusted Content Storage Service from the Layer A Trusted Content Storage Kernel (and EDV Microkernel)
- The USN Model for Layer A Content Change Detection, Tracking, & Notification and Layer B Content Change Detection, Tracking, & Notification
- As a general model, Layer A Trusted Content Storage Kernel is responsible for Content Change Detection and for propagating these events to the Layer B Trusted Content Storage Service (where Content Change Tracking and Detection functions are located).

Layer B Content Change Tracking and Notification Assessments

There are 3 variations (or models) for Layer B Content Change Detection/Tracking/Notification as depicted below.

- CRUD Events
- Update Sequence Numbers (USNs)
- Crawler Events

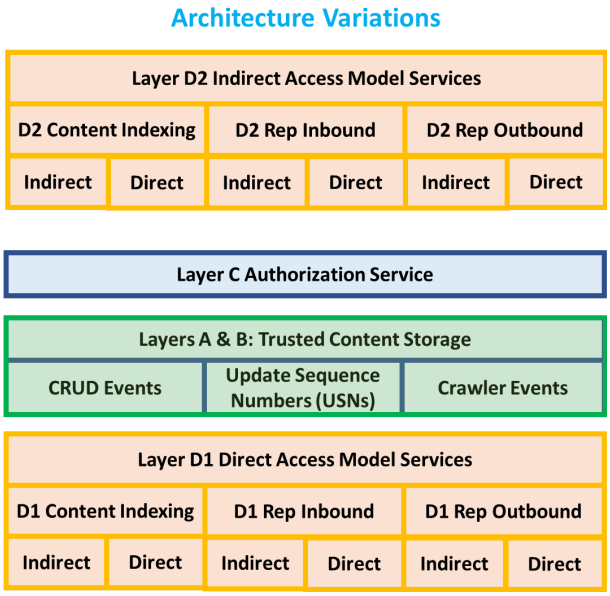


Figure 91. Layer B Content Change Detection/Tracking/Notification

Considerations, Principles, and Requirements

1. Separation of responsibilities: Layer A TCS Trusted Content Storage Kernel vs. Layer B Trusted Content Service
2. Layer B Trusted Content Storage Service to host protocol endpoints (e.g. HTTP REST, gRPC, Web Sockets, etc.)
3. Choice of Content Change Detection (and Tracking and Notification) model
    a. CRUD Events
    b. Update Sequence Numbers (USNs)
    c. Crawler Events
4. Approach for propagating/raising Content Change Detection events to Layer B Trusted Content Storage Service (dependent on the overall Content Change Detection Model chosen)

Prior Art

- See below for references to the prior art for each Content Change Detection, Tracking, and Notification model

Layer B: CRUD Event Model Variation: Assessment

The CRUD Events Model variation of the Layer B Trusted Content Storage Service is illustrated in the following figure.

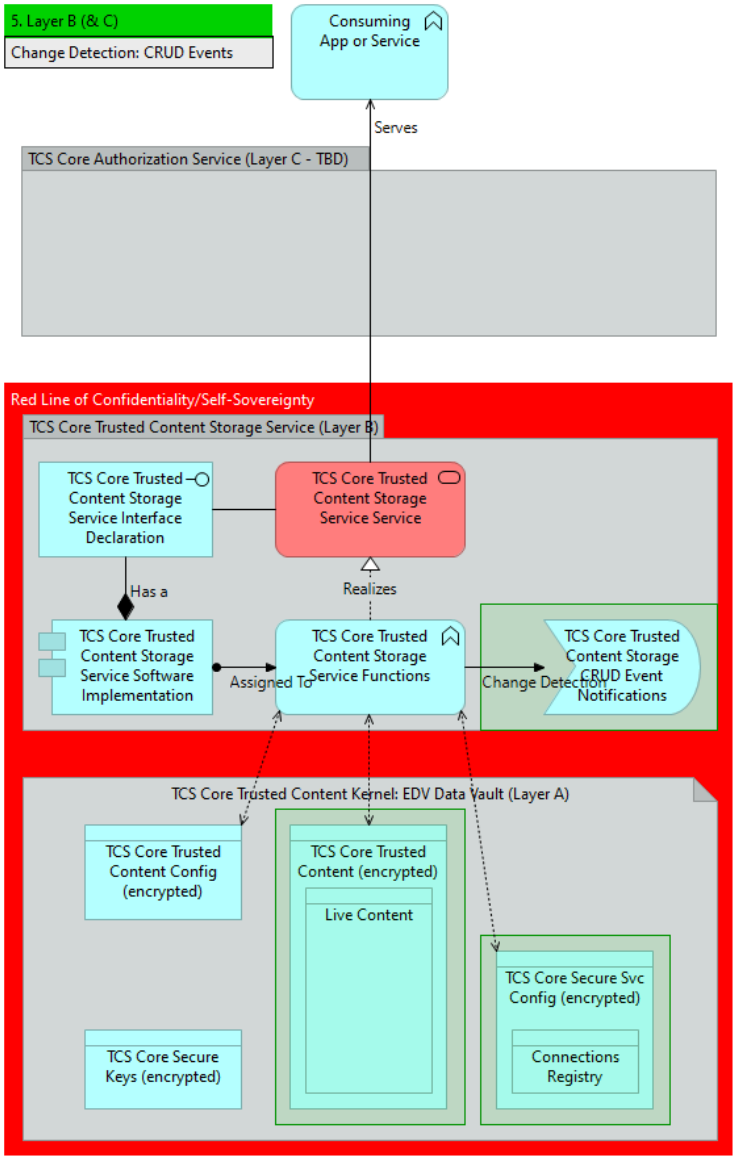


Figure 86. Layer B Content Change Detection: CRUD Events Model [5]

Considerations, Principles, and Requirements

- See the previous section's Considerations, Principles, and Requirements

Prior Art

- CRUD Event Model: Syntergy Replicator for SharePoint – see Appendix A

Assessment

Consideration	Pros	Cons
1	TODO	
2		
3		
4		

In summary, the final (interim) assessment is the CRUD Events Model is a moderate to very strong valued architecture variation.

Layer B: Update Sequence Number (USN) Model Variation: Assessment

Variations

The USN Events Model variation of the Layer A Trusted Content Storage Kernel is illustrated in the following figure.

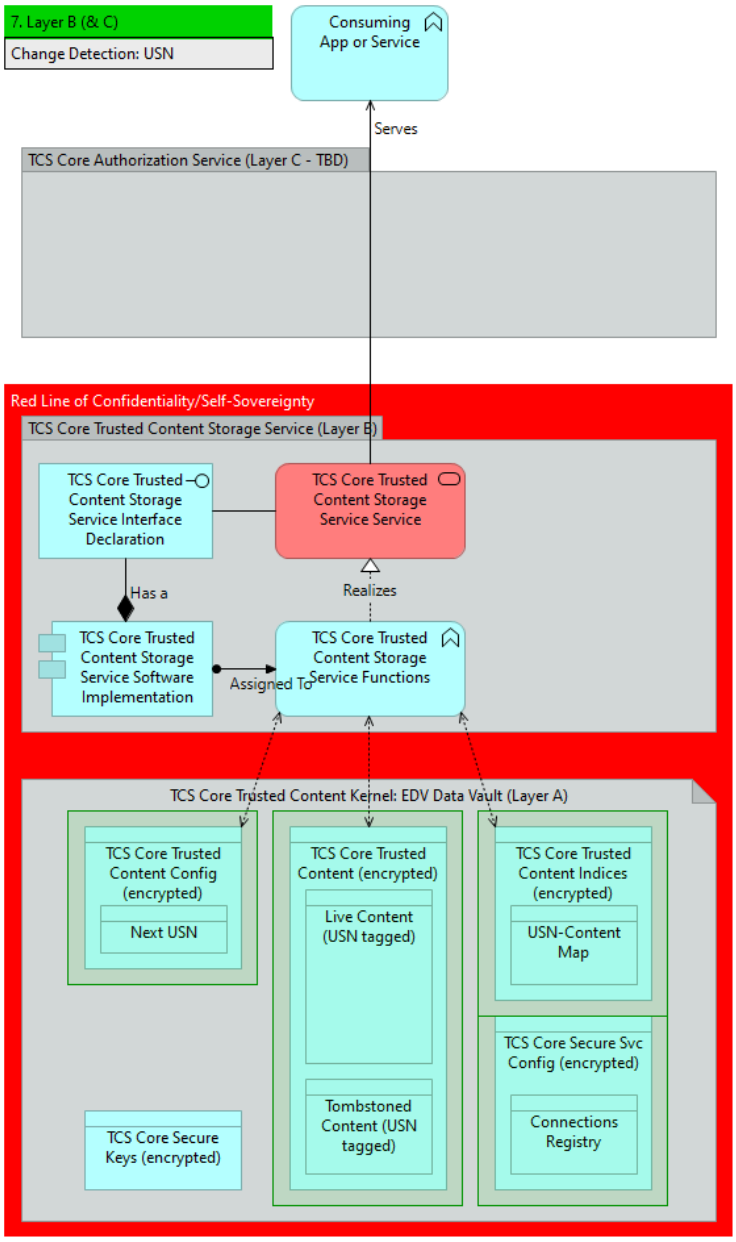


Figure 87. Layer B Content Change Detection: USN Model [7]

Considerations, Principles, and Requirements

- See the previous section's Considerations, Principles, and Requirements

Prior Art

- Microsoft Active Directory – see Appendix A

Assessment

Consideration	Pros	Cons
1	TODO	
2		
3		
4		

In summary, the final (interim) assessment is the USN Model is a strong to very strong valued architecture variation.

Layer B: Crawler Content Model Variation: Assessment

Variations

The Crawler Model variation of the Layer A Trusted Content Storage Kernel is illustrated in the following figure.

NOTE: The Crawler Model for Content Change Detection requires no specific support in the Layer A Trusted Content Kernel (nor the EDV Kernel).

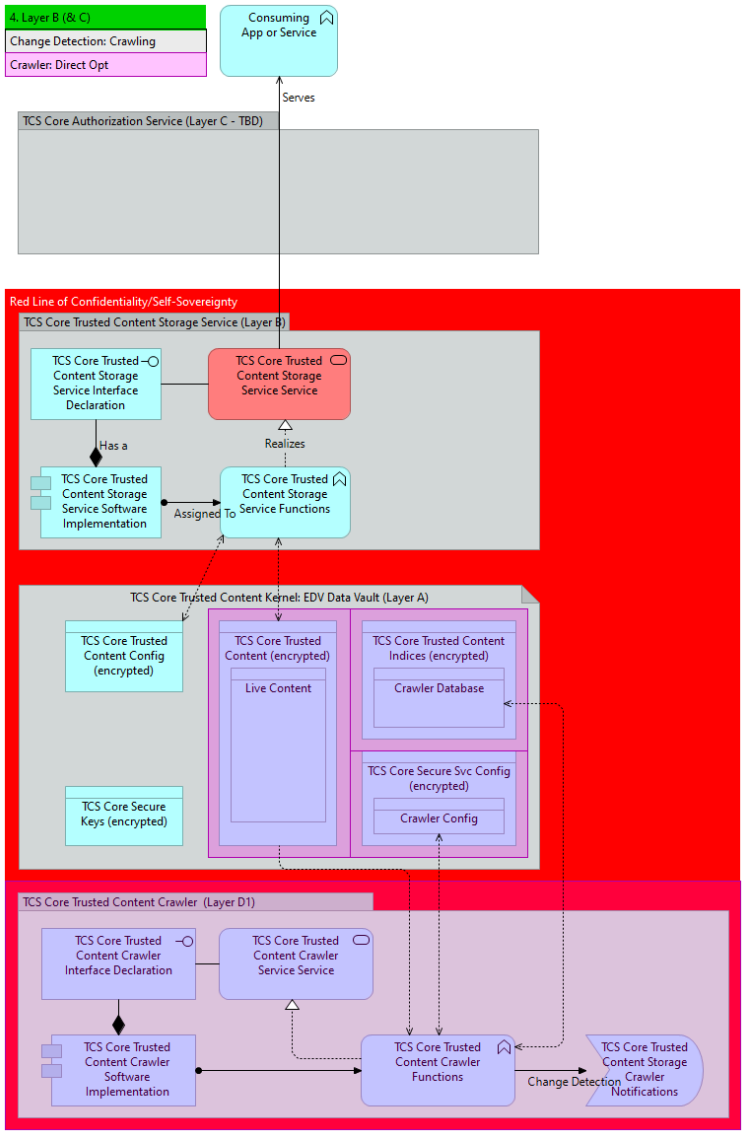


Figure 88. Layer B Content Change Detection/Tracking/Notification: Crawler Model [4]

Considerations, Principles, and Requirements

- See the previous section's Considerations, Principles, and Requirements

Prior Art

- Microsoft SharePoint Search – see Appendix A

Assessment

Consideration	Pros	Cons
1	TODO	
2		
3		
4		

In summary, the final (interim) assessment is the Crawler Model is a low to moderate valued architecture variation.

TCS Core Layer C Architecture Assessments

NOTE: This section is a placeholder for future discussion and elicitation of the architecture variations for Layer C Authorization Service.

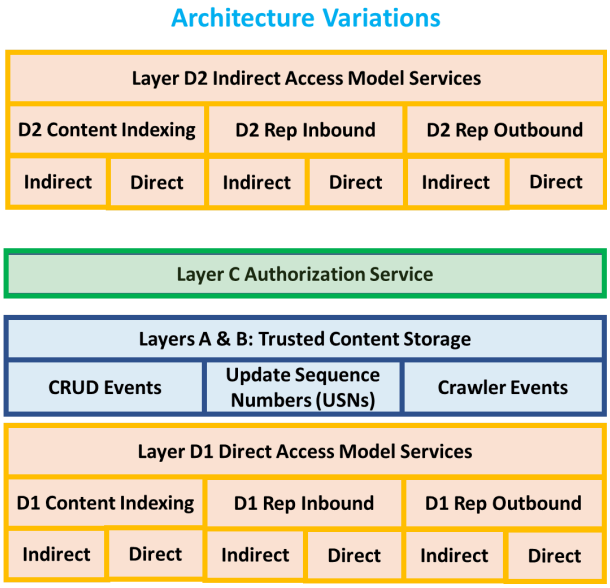


Figure 92. TCS Core Layer C Architecture Variations

Layer C Authorization Service

NOTE: This section is a placeholder for future discussion and elicitation of the architecture variations for Layer C Authorization Service.

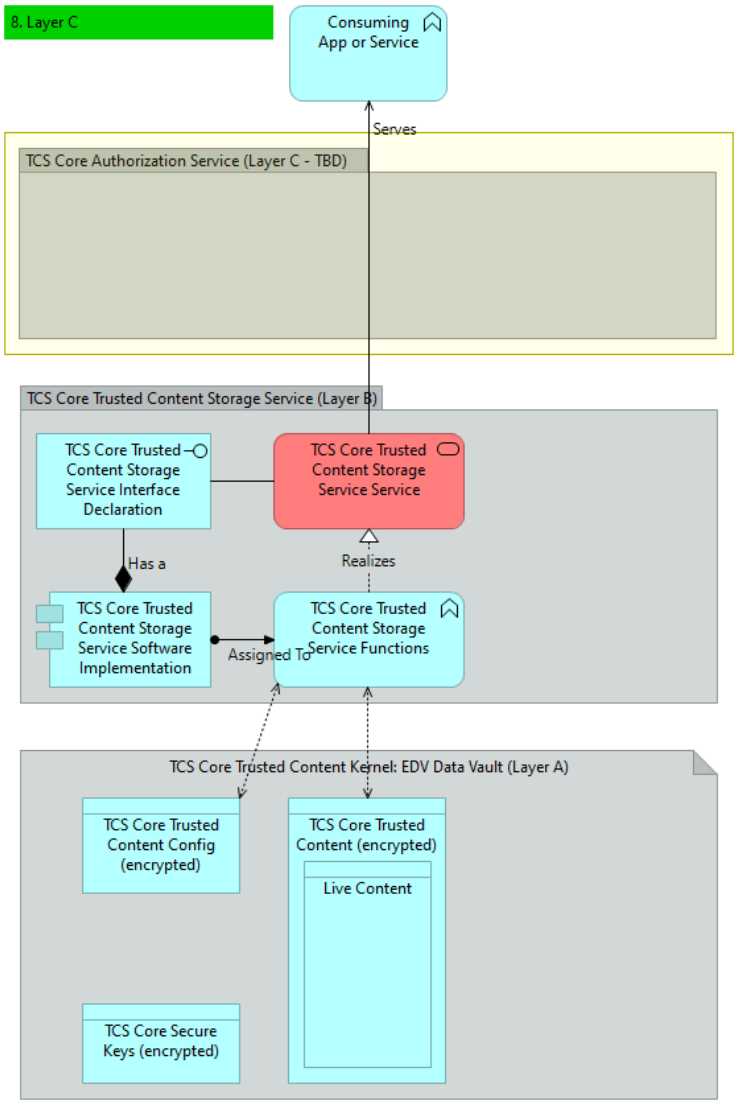


Figure 93. Layer C Authorization Service [8]

TCS Services Layer D Architecture Assessments

The primary dimensions of the architecture variations in TCS Services Layer D Services are the following:

4. TCS Core
    a. Layer A Trusted Content Storage Kernel
        i. Layer A Content Change Detection Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    b. Layer B Trusted Content Storage Service
        i. Layer B Content Change Tracking and Notification Models
            1. CRUD Event Model
            2. USN Model
            3. Crawler Model
    c. Layer C Authorization Service[10]
5. TCS Services Layer D Access Models
    a. Layer D1 Direct Access Model
    b. Layer D2 Indirect Access Model
6. TCS Services
    a. Layer D Replication Services
        i. Layer D Replication Outbound Processing Service
        ii. Layer D Replication Package Transfer Service
        iii. Layer D Replication Inbound Processing Service
    b. Layer D Indexing & Search Services
        i. Layer D Content Indexing Service

[10] NOTE: TCS Core Layer C Authorization Service is deemed out-of-scope for this version of the whitepaper.

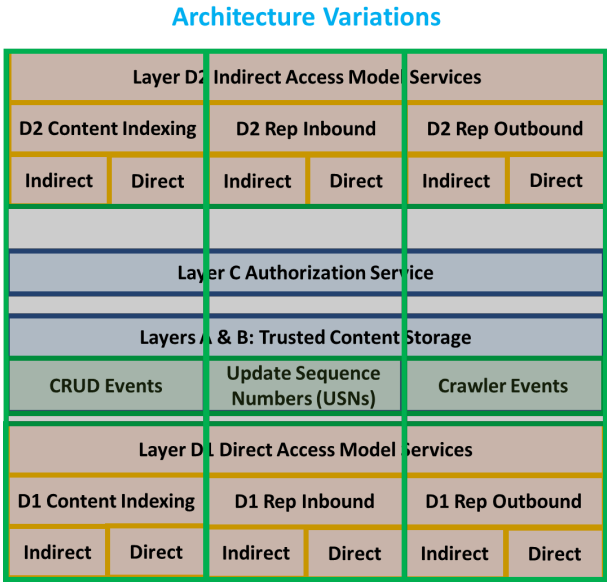


Figure 50. TCS Services Layer D Architecture Variations

Two important TCS Services Layer D Services not mentioned in the above list include:

- Layer D Replication Package Transfer Service
- Layer D Search Query Processing Service

Layer D Direct and Indirect Access Service Access Models: Assessment

TCS Services Layer D Services is further partitioned into 2 sublayers:

- Layer D1 Direct Access Services
- Layer D2 Indirect Access Services

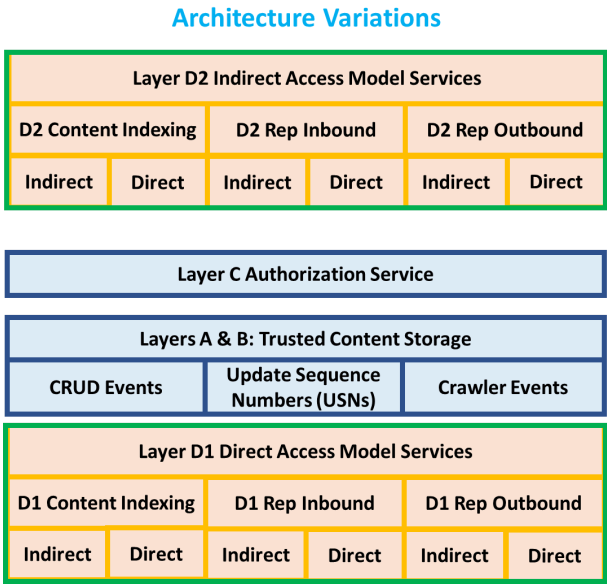


Figure 52. Layer D Direct and Indirect Access Service Models

Considerations, Principles, Requirements, and Assumptions

1. Efficient, secure direct access to Layer A Trusted Content Storage (via the Layer A Trusted Content Storage Kernel)
2. More secure indirect access to Layer A Trusted Content Storage (via the Layer B Trusted Content Storage Service)

Prior Art

- No referenceable prior at the time of writing

Layer D1 Direct Access Service Model Variation: Assessment

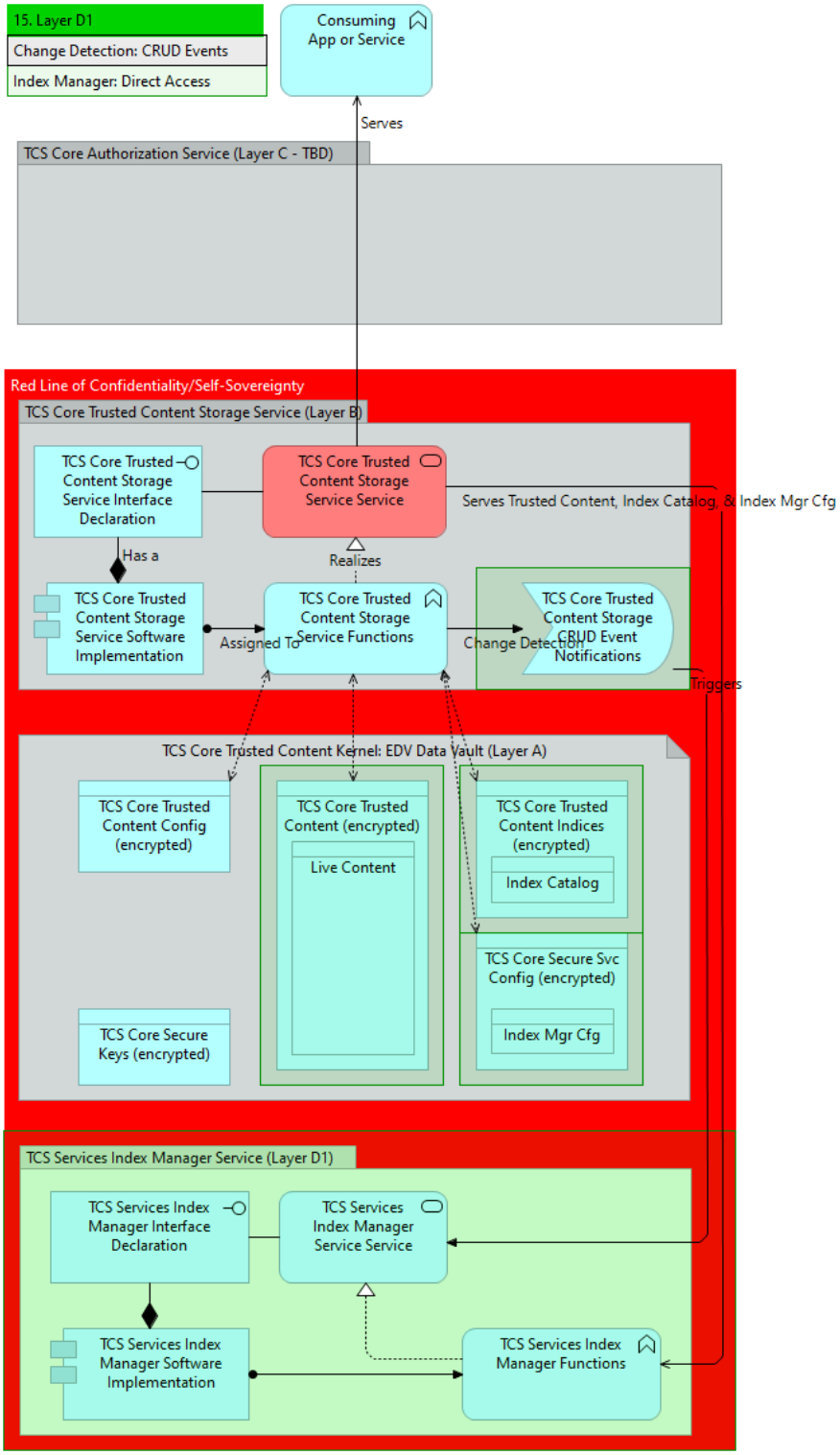


Figure 95. Layer D1: Direct Access Service Model Example [15]

Layer D2 Indirect Access Model Variation: Assessment

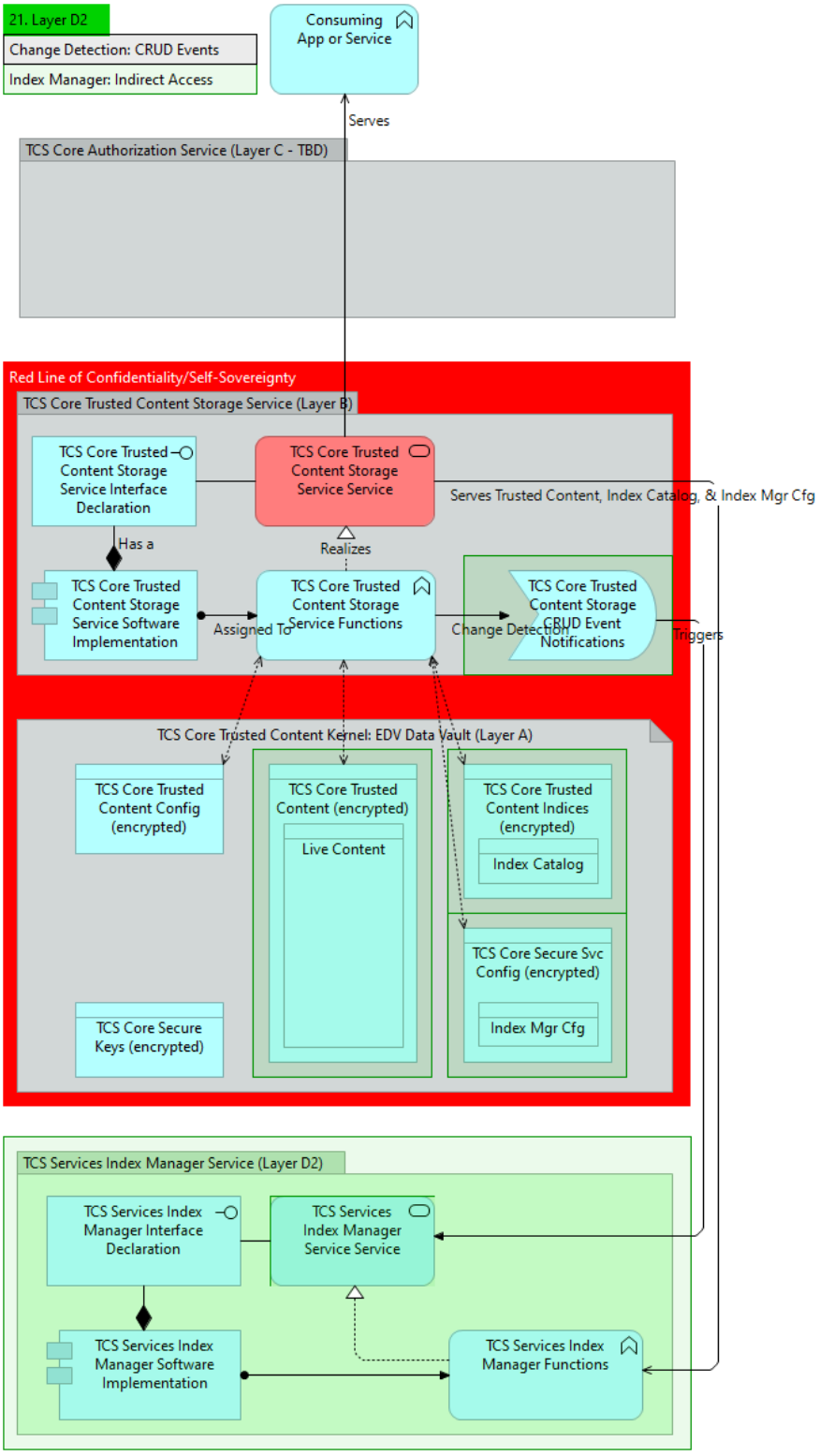


Figure 96. Layer D2: Indirect Access Service Model Example [21]

Assessment

Consideration	Pros	Cons
1	An efficient, secure direct access model makes sense for Layer D Services that need high-performance read and write access to data in Layer A Trusted Content Storage Kernel EDV Data Vaults	More complex/sophisticated software architecture and design.
2	A more secure direct access model makes sense for Layer D Services that need high-performance read and write access to data in Layer A Trusted Content Storage Kernel EDV Data Vaults	A less efficient direct access model makes sense for Layer D Services that need high-performance read and write access to data in Layer A Trusted Content Storage Kernel EDV Data Vaults

In summary, the final (interim) assessment is enables selected Layer “D1” Services to use the direct access model while relegating other services (Layer “D2” Services) to use the indirect access model.

Layer D Replication Services: Assessments

TCS Services Layer D Replication Services includes:

- Replication Outbound Processing Services
- Replication Package Transfer Service
- Replication Inbound Processing Service

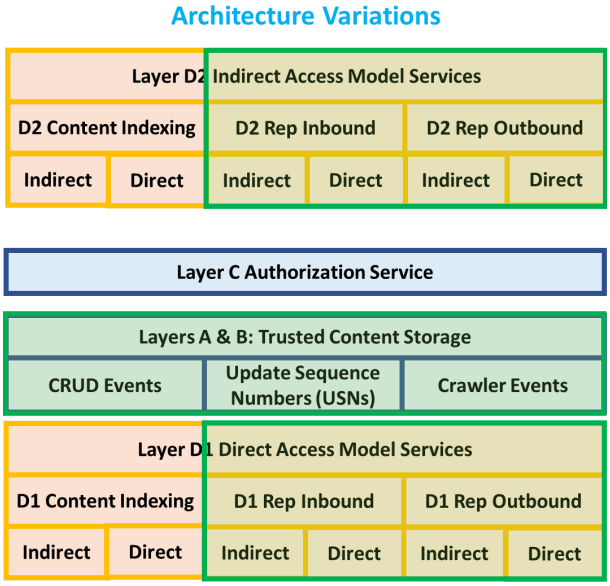


Figure 97. Layer D Replication Services: Architecture Variations

Considerations, Principles, Requirements, and Assumptions

TODO

1. Real-time replication
2. Offline replication
3. Air gap replication
4. Slow, high-latency network connections (e.g. Satellite communications to Antarctica)
    a. Content tub replication
5. Selective Replication (Outbound filtering and Inbound package acceptance)
    a. Content types: content, metadata, schema, container hierarchies, security permissions, security principles, roles, event handlers/workflow instances, configuration settings
6. Security
    a. Data at rest; data in motion (replication packages)
    b. Mapping security principles and roles
7. Bi-directional vs. Uni-directional
8. Performance Management: bandwidth throttling, CPU throttling, disk management
9. Failover/service recovery
10. Disaster recovery: feasibility, fidelity, time/bandwidth/processing required

11. Support for large objects (terabyte blob)
12. Support for large containers (petabytes)
13. High-frequency content changes
14. Dynamic, multi-hop routing
15. Content re-targeting (re-homing)
16. Cross-firewall replication
17. Multi-master support
18. Activity anonymity
19. Peer-to-Peer (P2P) topologies: Direct and Mediated
20. Conflict management (Inbound Processing Service)
    a. Conflict detection
    b. Conflict resolution
21. Multi-Master Replication

Reference:
https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649910(v=pandp.10)?redirectedfrom=MSDN

Prior Art

Syntergy Replicator for SharePoint TODO

Groove Workspace TODO

Microsoft Active Directory TODO

Microsoft Sync Foundation TODO

Layer D Replication Outbound Processing Service: Assessment Variations

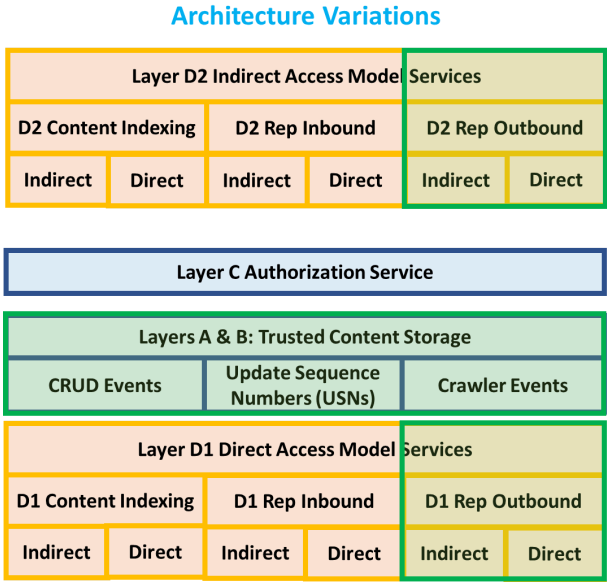


Figure 98. Layer D Replication Outbound Processing Service: Architecture Variations

TODO

Table 7. Replication Outbound Processing Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct		●20	●13
D2 Indirect	●23	●25	

TODO

Considerations, Principles, Requirements, and Assumptions

TODO

Prior Art

TODO

Assessment

TODO

Pros	Cons

TODO

In summary, the final (interim) assessment is TODO

Layer D Replication Package Transfer Service: Assessment Variations

TODO

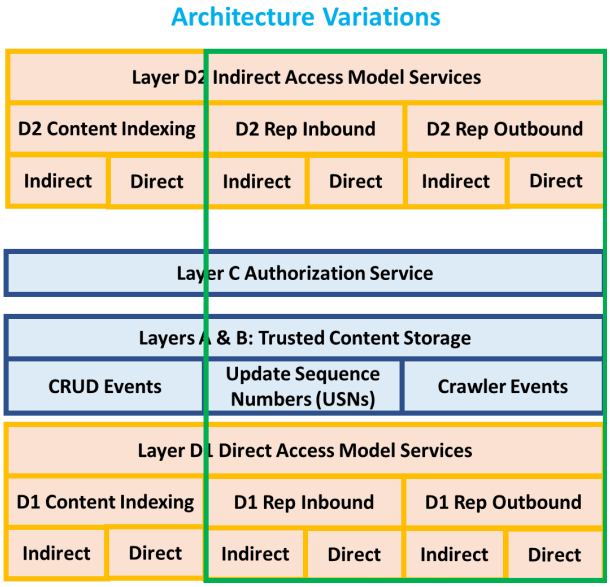


Figure 99. Layer D Replication Package Transfer Service: Architecture Variations

TODO

Table 8. Replication Package Transfer Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct	---		
D2 Indirect	●[20,25]		

TODO

Considerations, Principles, and Requirements

TODO

Prior Art

TODO

Assessment

TODO

Pros	Cons

TODO

In summary, the final (interim) assessment is TODO

Layer D Replication Inbound Processing Service: Assessment Variations

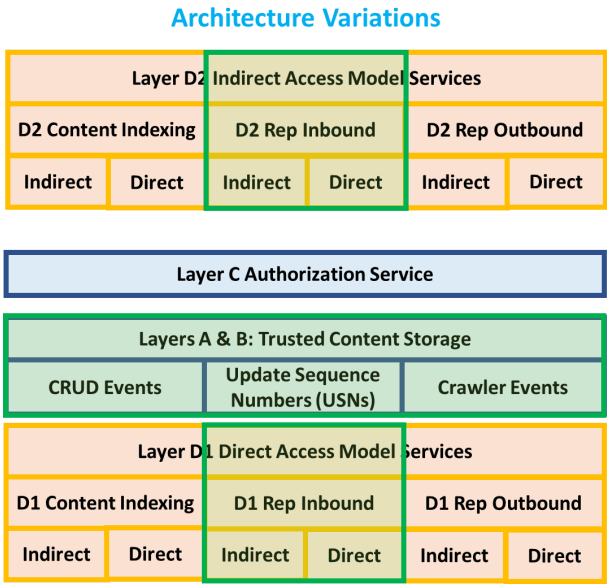


Figure 100. Layer D Replication Inbound Processing Service: Architecture Variations

TODO

Considerations, Principles, and Requirements

TODO

Prior Art

TODO

Assessment

TODO

Pros	Cons

TODO

In summary, the final (interim) assessment is TODO

Layer D Content Indexing and Search Query Processing Services: Assessments Variations

Layer D Indexing and Search Services includes:

i. Layer D Content Indexing (Index Manager) Service
ii. Layer D Search Query Processing Service

The ARMs for each of the Layer D Indexing and Search Services architecture variations (depicted in the following figure) are presented in the following sections.

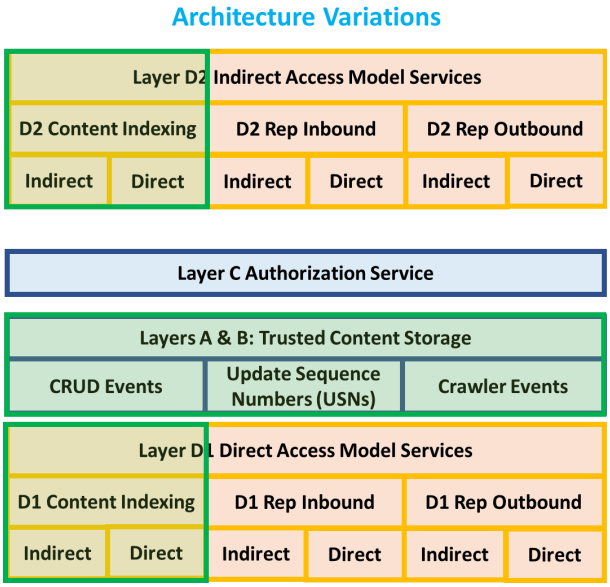


Figure 101. Layer D Indexing and Search Services: Architecture Variations

The Search Query Processing Service doesn't appear in the above diagram because it has a single architecture variation that is independent of the chosen Layer A and Layer B Content Change Detection/Tracking/Notification Model and Layer D Access Service Model.

TODO

Considerations, Principles, Requirements, and Assumptions

TODO

- Security crawling

Prior Art

Microsoft Search TODO

Layer D Content Indexing (Index Manager) Service: Assessment Variations

The ARMs for the architecture variations for the Content Indexing Service (Index Manager) Service are described below.

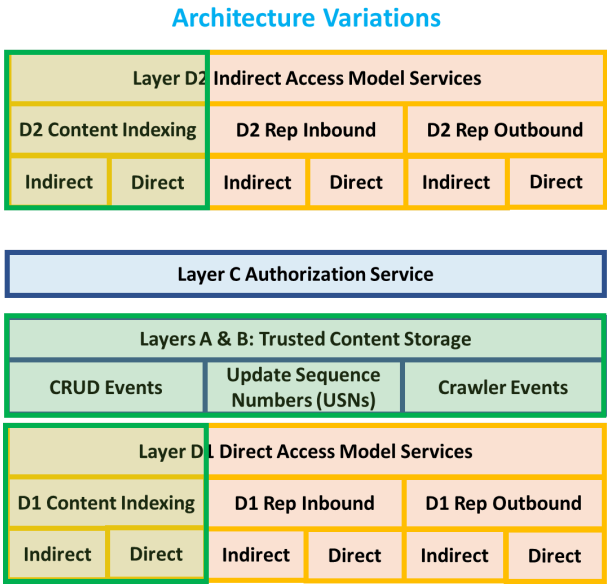


Figure 102. Layer D Content Indexing Service: Architecture Variations

TODO

Table 9. Content Indexing Service (Index Manager) Service: Architecture Variations

Content Access Model	CRUD Event Model Detection/Tracking	USN Model Detection/Tracking	Crawler Model Detection/Tracking
D1 Direct	●[15,16]	●[17,18]	●[14]
D2 Indirect	●[21]		

TODO

Considerations, Principles, Requirements, and Assumptions

TODO

- Security crawling

Prior Art

Microsoft Search TODO

Assessment

TODO

Pros	Cons

TODO

In summary, the final (interim) assessment is TODO

Layer D Search Querying Service: Assessment

Variations

TODO

Considerations, Principles, Requirements, and Assumptions

TODO

- Security trimming

Prior Art

Microsoft Search TODO

TODO

Assessment

TODO

Pros	Cons

TODO

In summary, the final (interim) assessment is TODO

TCS-Stack System-Level Architecture Variations: Overall Assessment

TODO

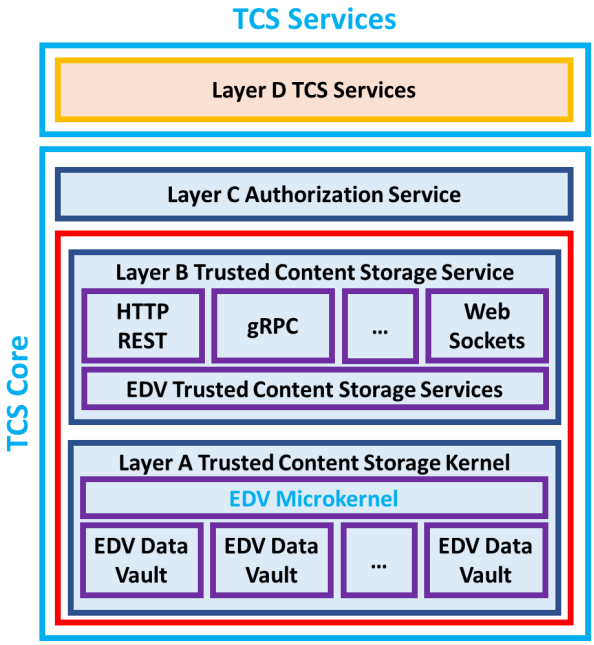


Figure 103. Detailed TCS Stack

TODO

System-Level Architecture Variations

TODO

CRUD Events Model

TODO

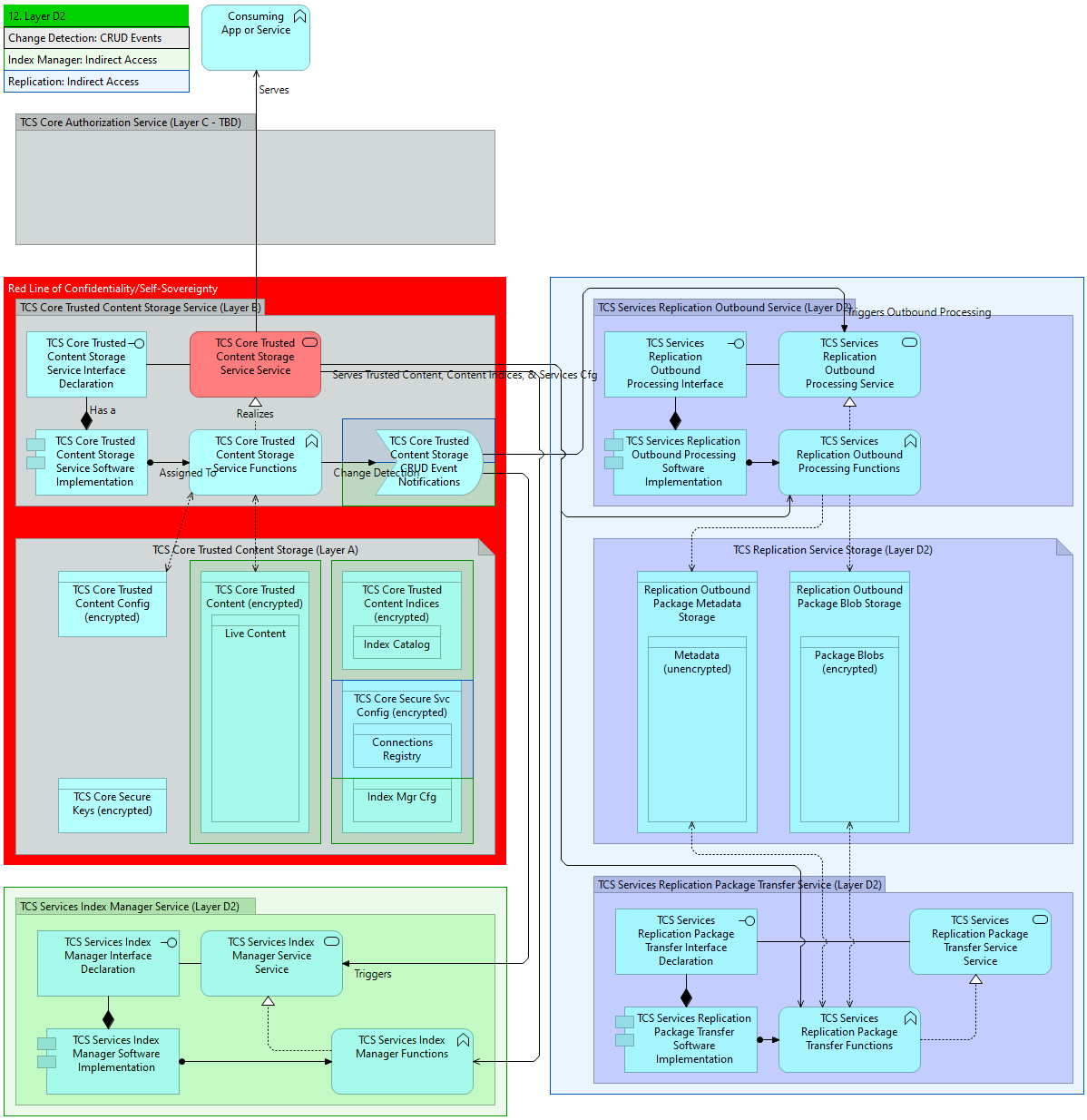


Figure 104. CRUD Events Content Change Detection/Tracking/Notification: Layer D2 Index Manager & D2 Replication Outbound Processing Service [12]

TODO

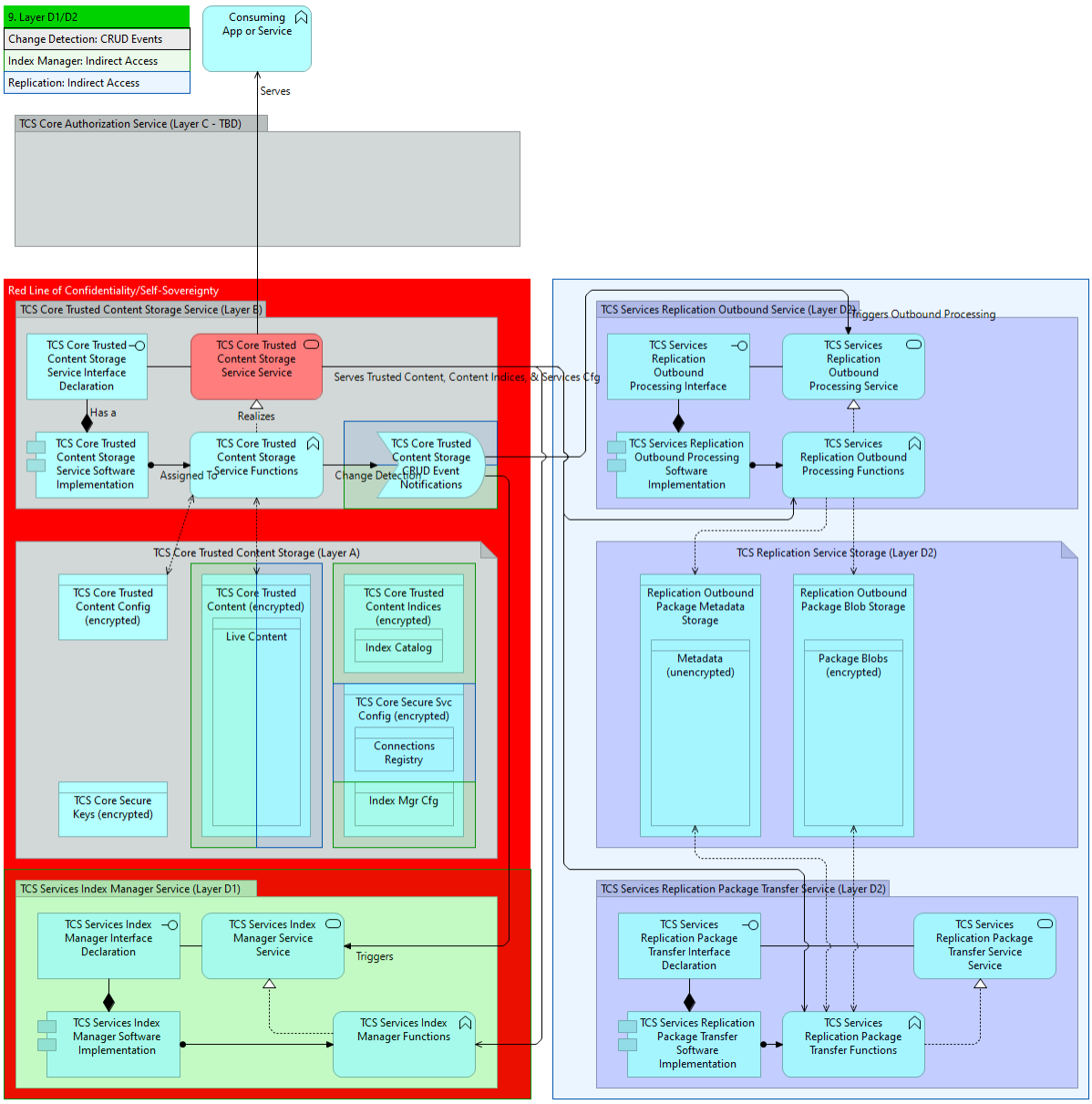


Figure 105. CRUD Events Content Change Detection/Tracking/Notification: Layer D1 Index Manager & D2 Replication Outbound Processing Service [9]

TODO

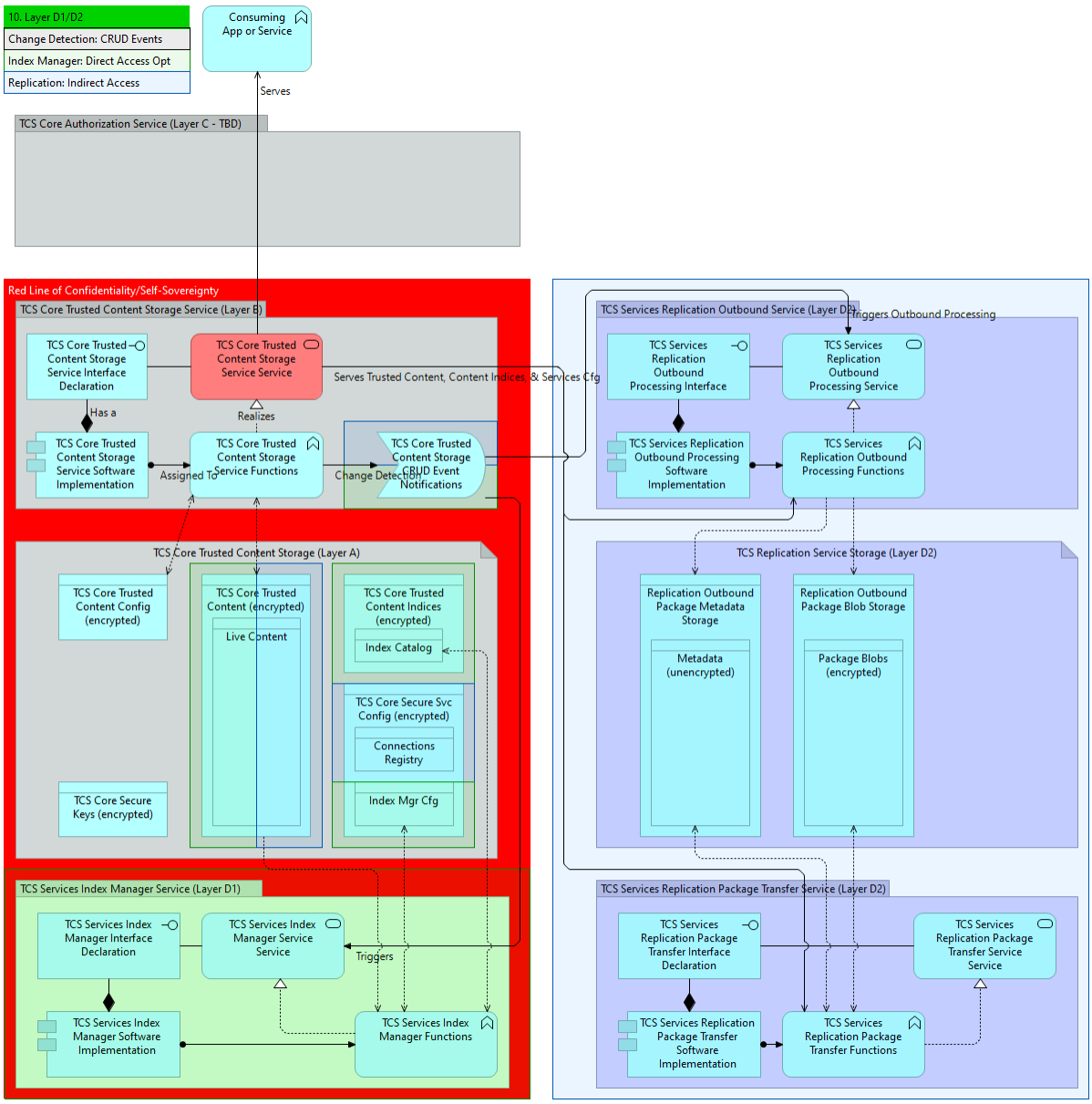


Figure 106. CRUD Events Content Change Detection/Tracking/Notification: Layer D1 Index Manager (optimized) & D2 Replication Outbound Processing Service [10]

TODO

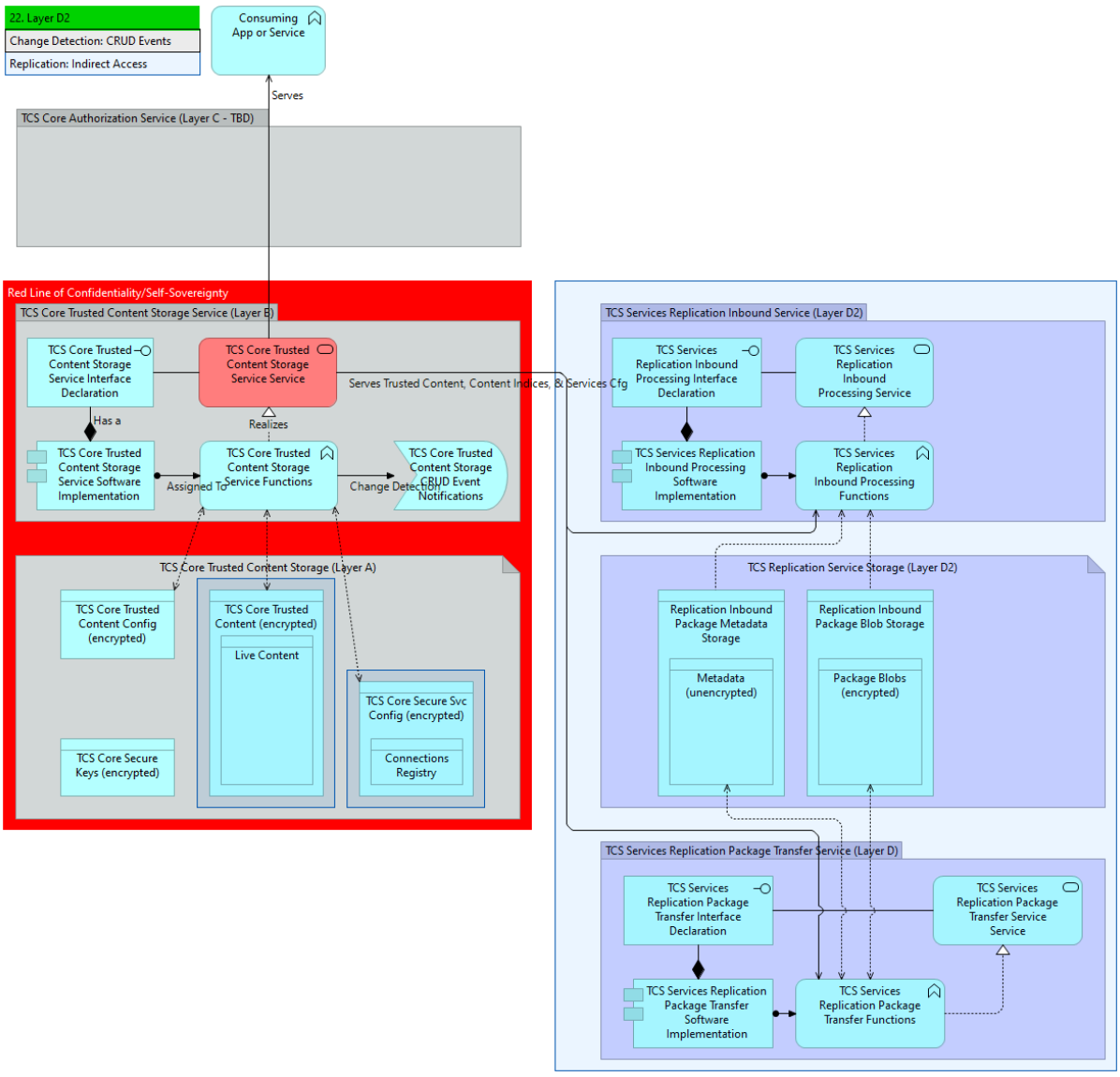


Figure 107. D2 Replication Inbound Processing Service [22]

TODO

Update Sequence Numbers (USNs) Model

TODO

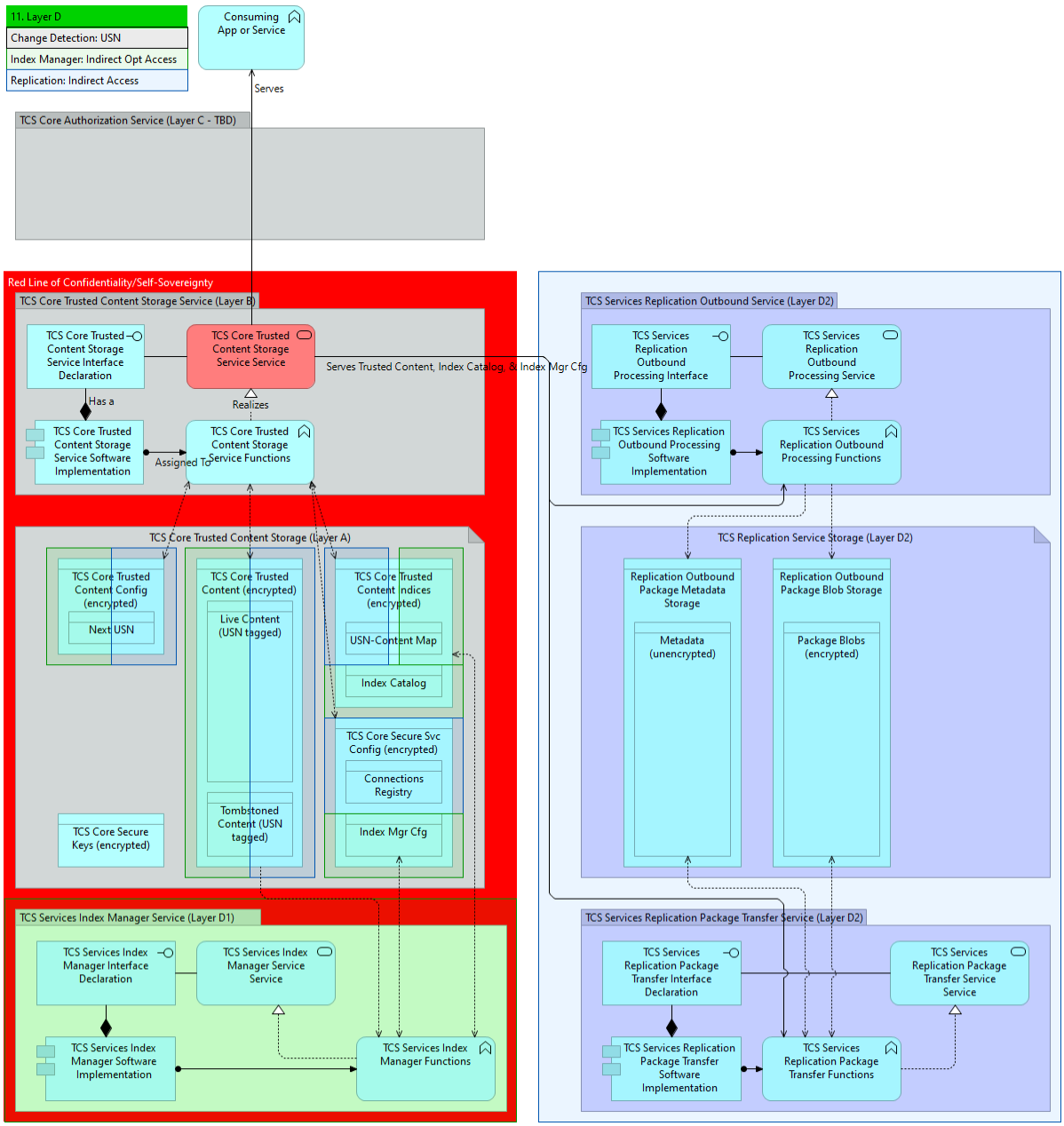


Figure 108. USN Content Change Detection/Tracking/Notification: Layer D1 Index Manager (optimized) & D2 Replication Outbound Processing Service [11]

TODO

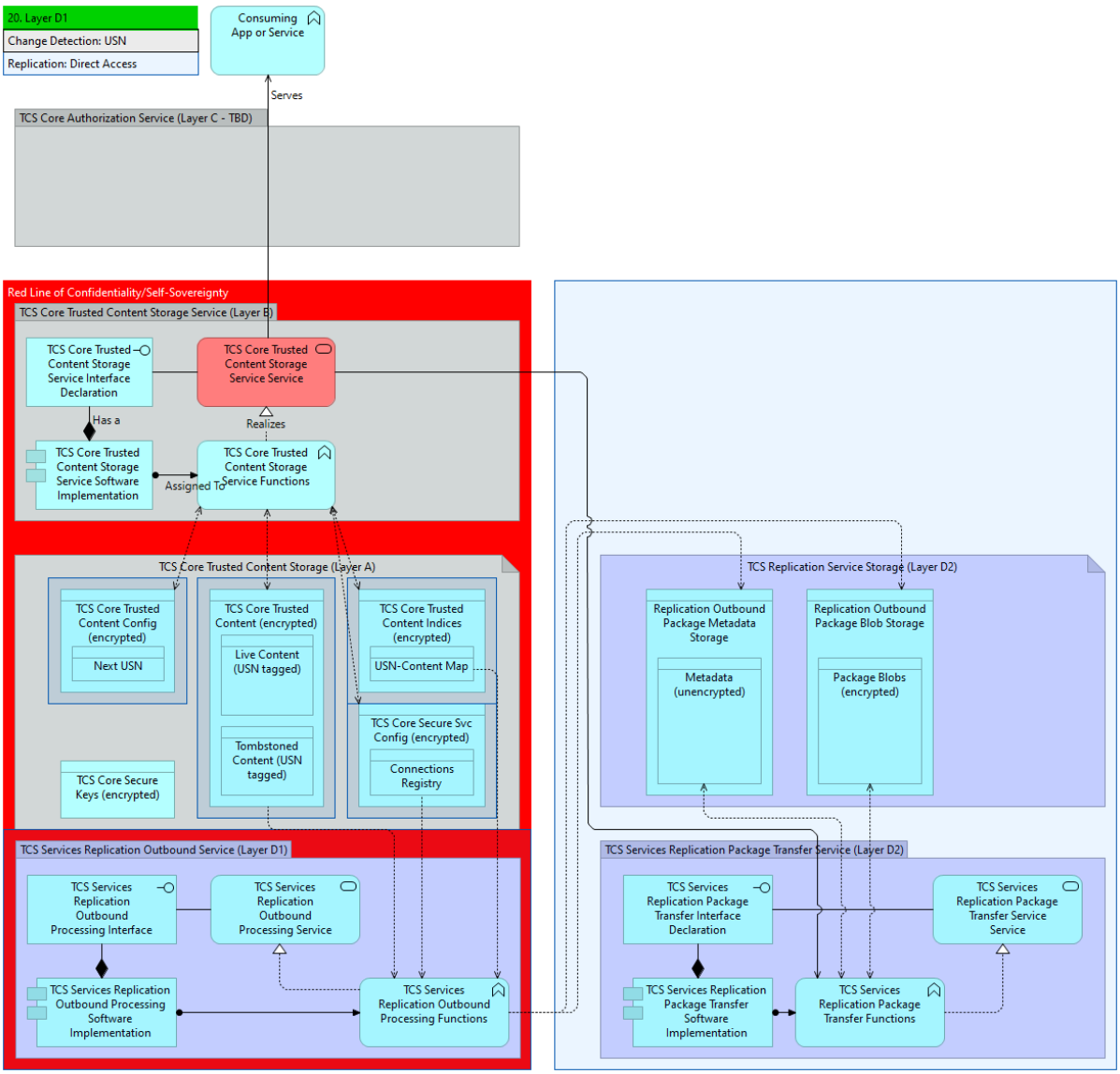


Figure 109. USN Content Change Detection/Tracking/Notification: Layer D1 Replication Outbound Processing Service & D2 Package Transfer Service [20]

TODO

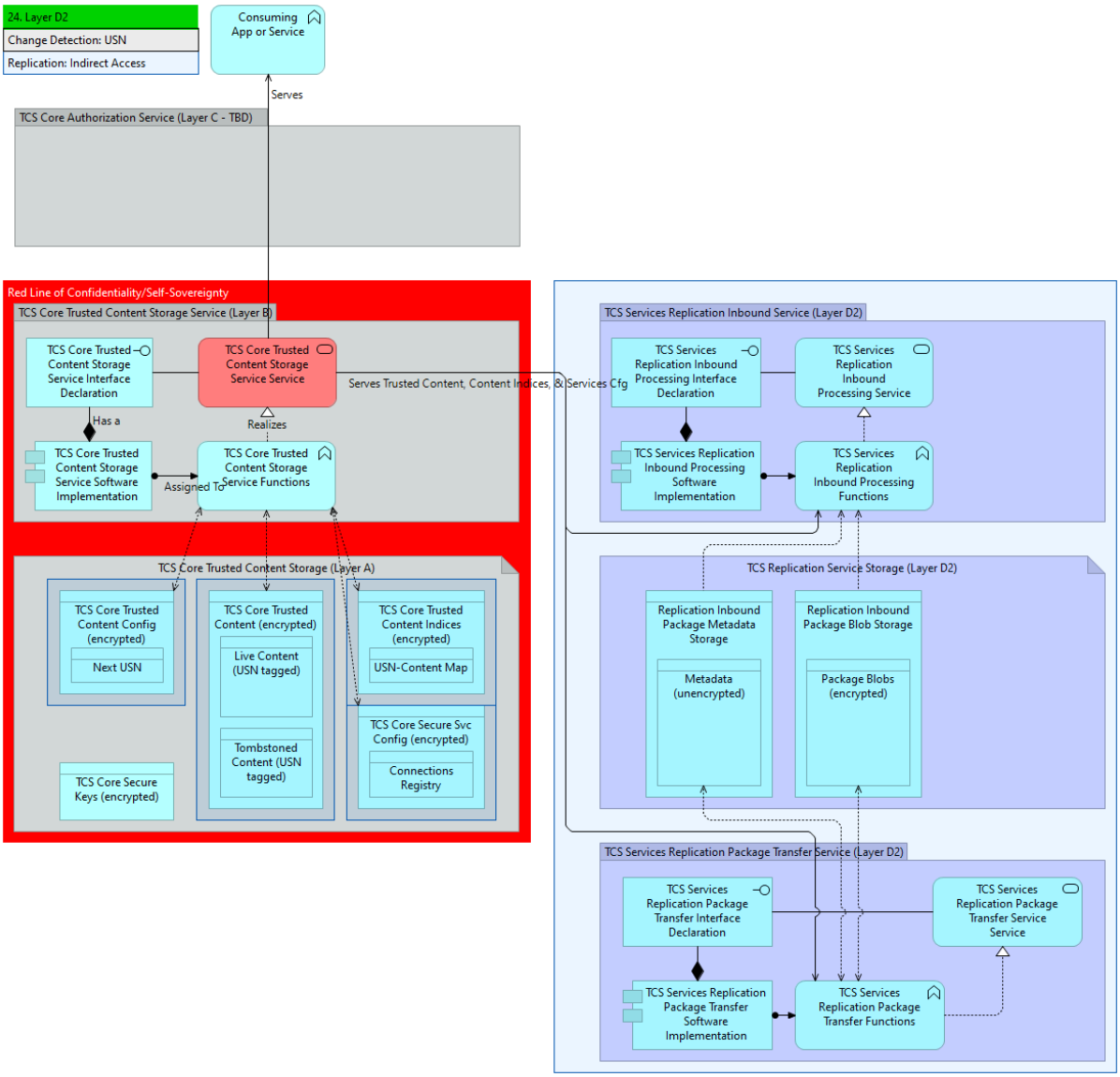


Figure 110. USN Content Change Detection/Tracking/Notification: Layer D2 Replication Inbound Processing Service & D2 Package Transfer Service [24]

TODO

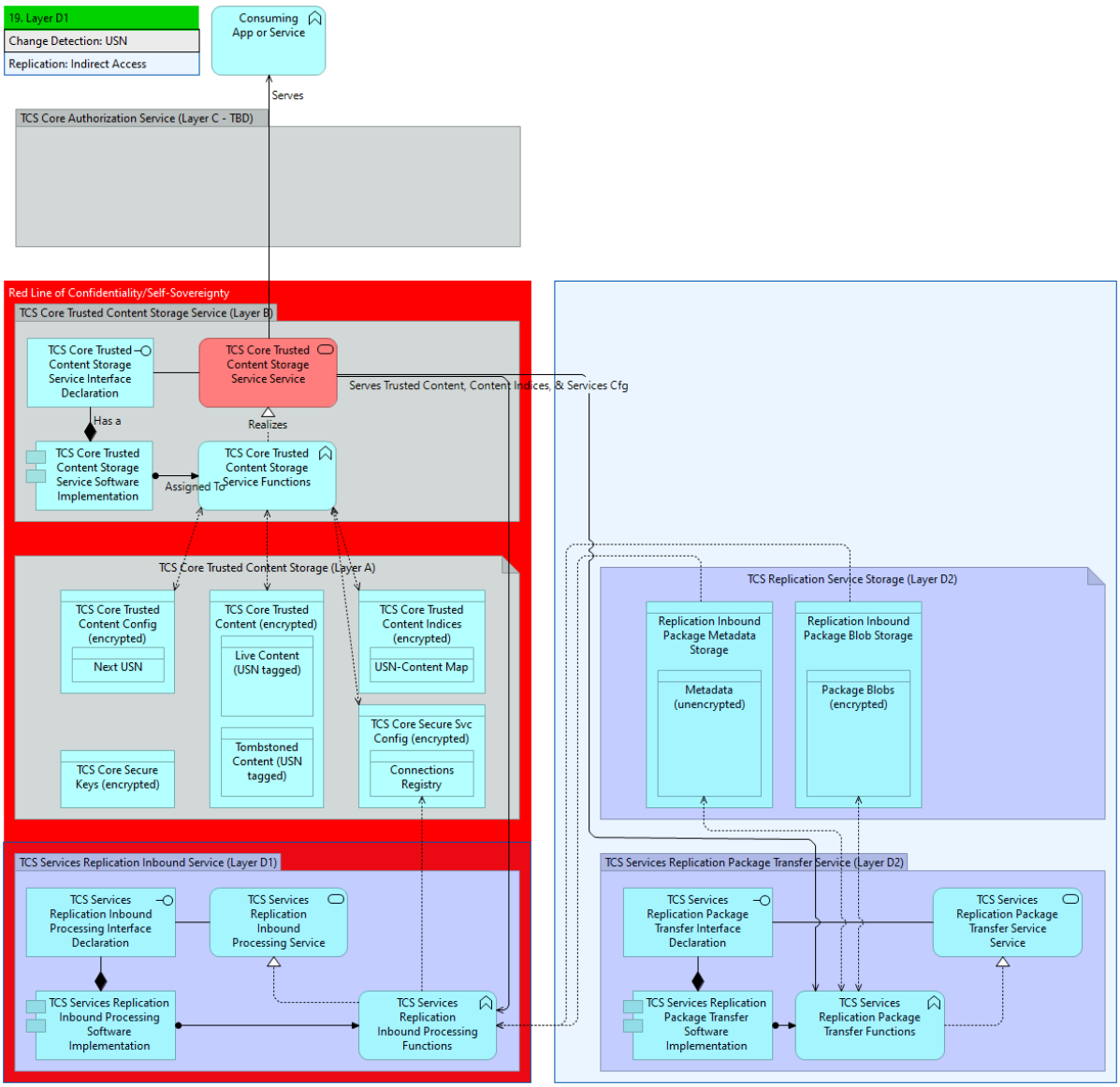


Figure 111. USN Content Change Detection/Tracking/Notification: Layer D1 Replication Inbound Processing Service and D2 Package Transfer Service [19]

TODO

Crawler Model

TODO

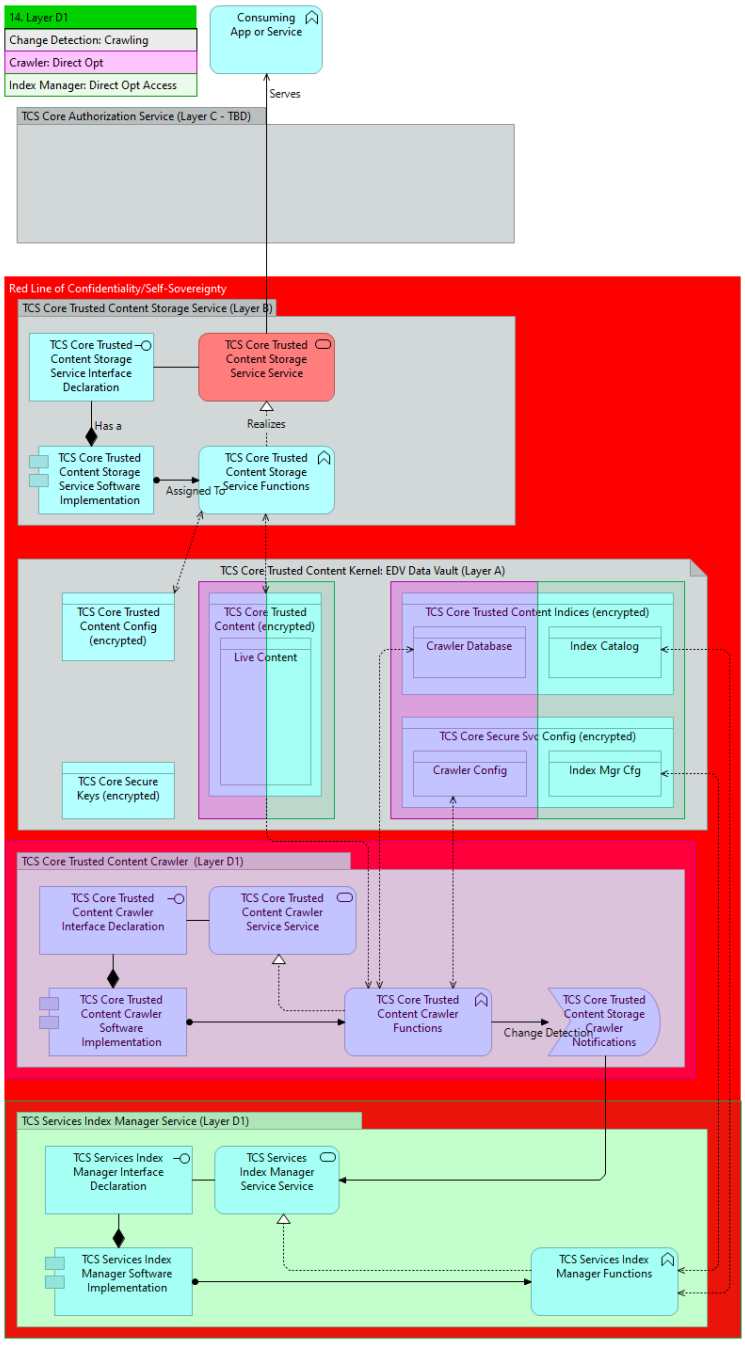


Figure 112. Crawler Content Change Detection/Tracking/Notification: Layer D1 Index Manager (optimized) [14]

TODO

TODO

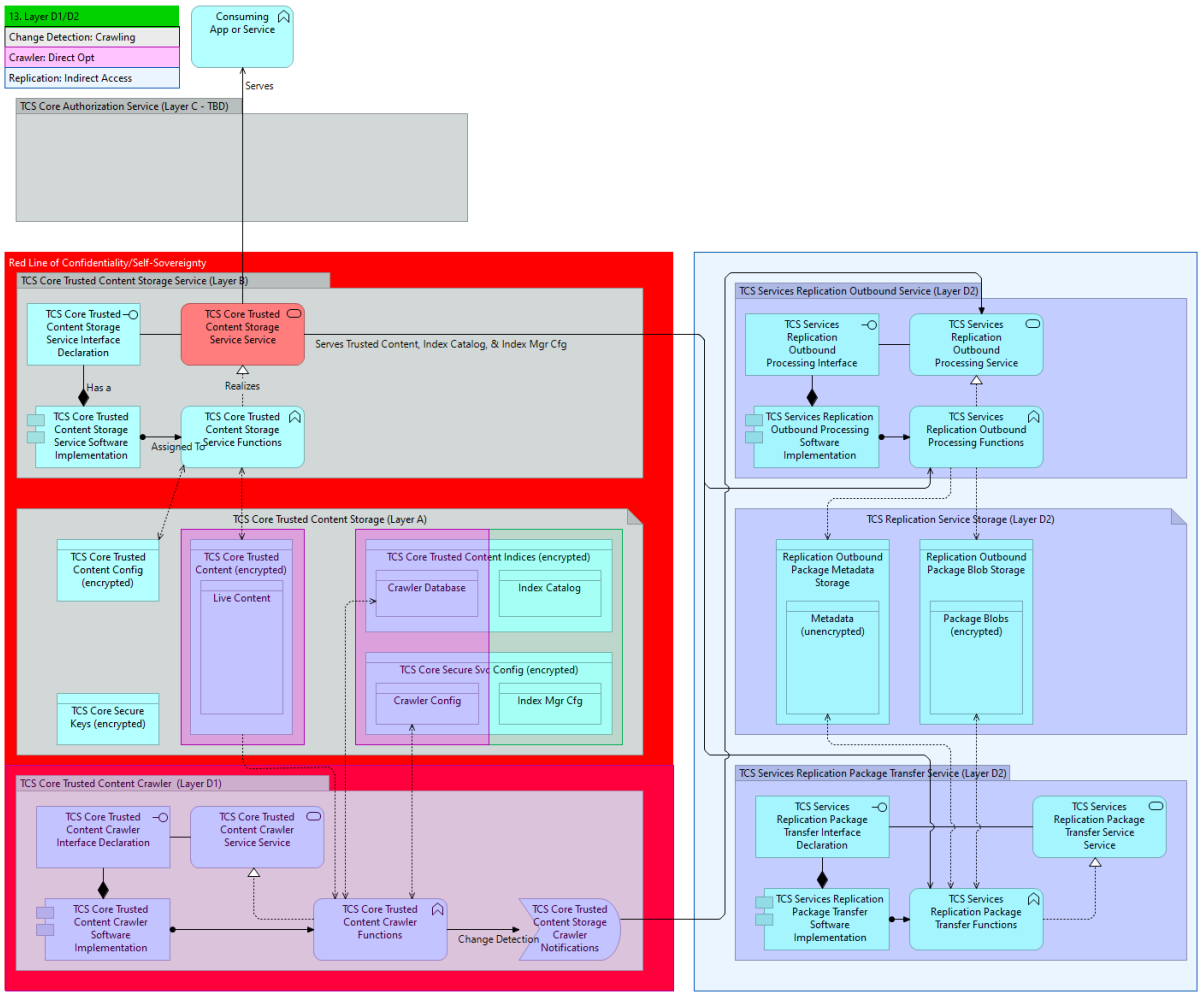


Figure 113. Crawler Content Change Detection/Tracking/Notification: Crawled Events Generator (D1 optimized) [11]

TODO

Assessment

TODO

Pros	Cons

In summary, the final (interim) assessment is TODO

Architecture Recommendations

TODO

TCS Core Layer A Recommendations

Layer A Trusted Content Storage Kernel

TODO

Content Change Detection/Tracking/Notification

Update Sequence Numbers (USNs) TODO

TCS Core Layer B Recommendations

Layer B Trusted Content Storage Service

TODO

Content Change Detection/Tracking/Notification

Update Sequence Numbers (USNs) TODO

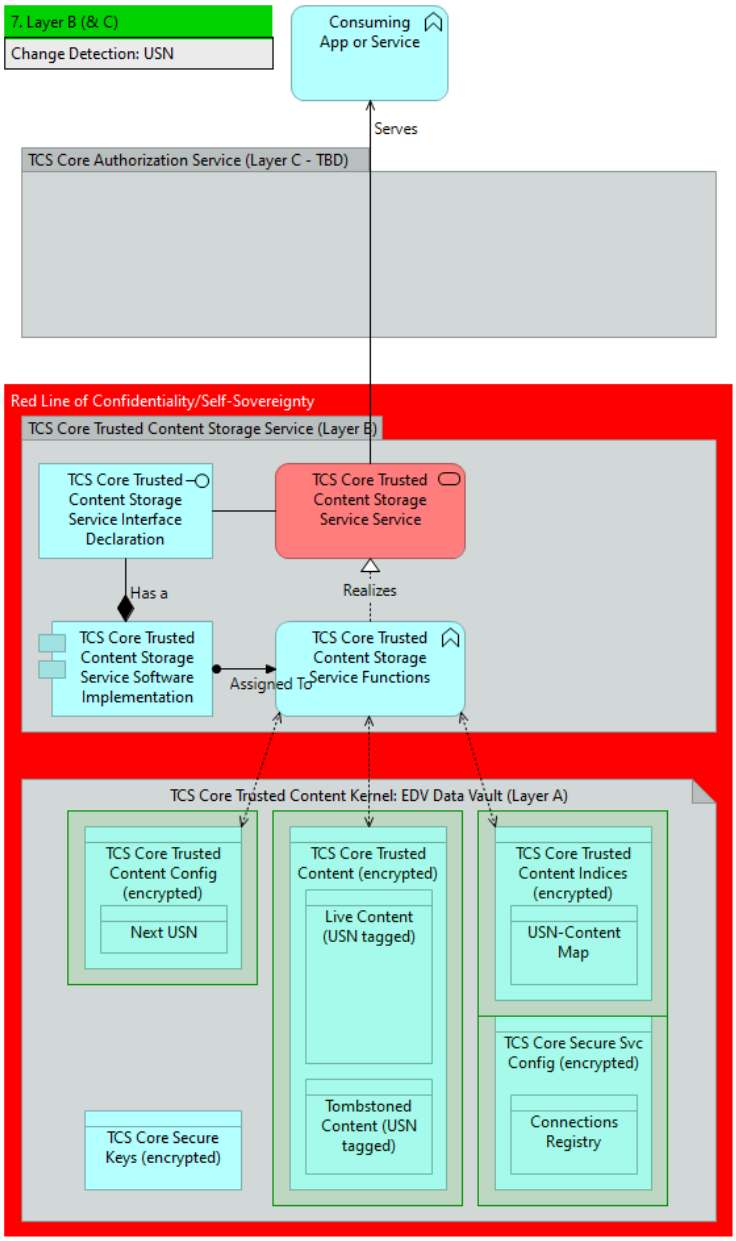


Figure 114. Layer B Content Change Detection/Tracking/Notification: USN Model [7]

TODO

TCS Core Layer C Recommendations

Layer C Authorization Service

NOTE: This section is a placeholder for future discussion and elicitation of the architecture variations for Layer C Authorization Service.

TCS Services Layer D Recommendations

TODO

Layer D Replication Services

TODO

Layer D Replication Outbound Processing Service

USN Model TCS Services D1 Replication Outbound Processing Service (Direct Access Service Model)

TODO

Layer D Replication Package Transfer Service

TCS Services D2 Replication Package Transfer Service (Indirect Access Service Model)

TODO

Layer D Replication Inbound Processing Service

TCS Services D1 Replication Inbound Processing Service (Direct Access Service Model)

TODO

Layer D Indexing & Search Services

TODO

Layer D Content Indexing Service

USN Model TCS Services D1 Index Manager Service (Direct Access Service Model)

TODO

Layer D Search Query Processing Service

TCS Services D2 Search Service (Indirect Access Service Model)

TODO

Bringing It All Together

TODO

Update Sequence Numbers (USNs) based ARM

TODO

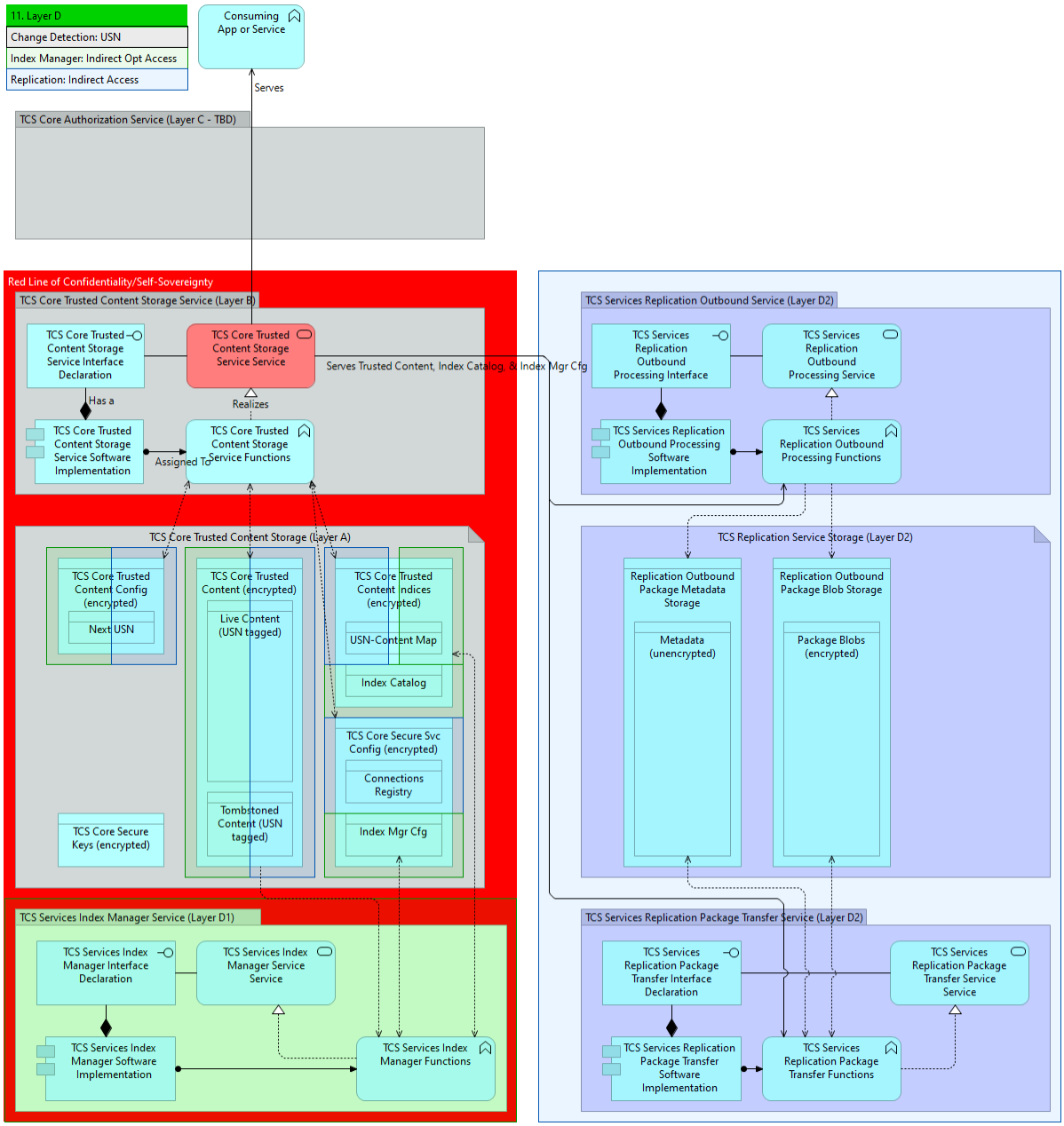


Figure 115. USN Content Change Detection/Tracking/Notification: Layer D1 Index Manager (optimized) & D2 Replication Outbound Processing Service [11]

TODO

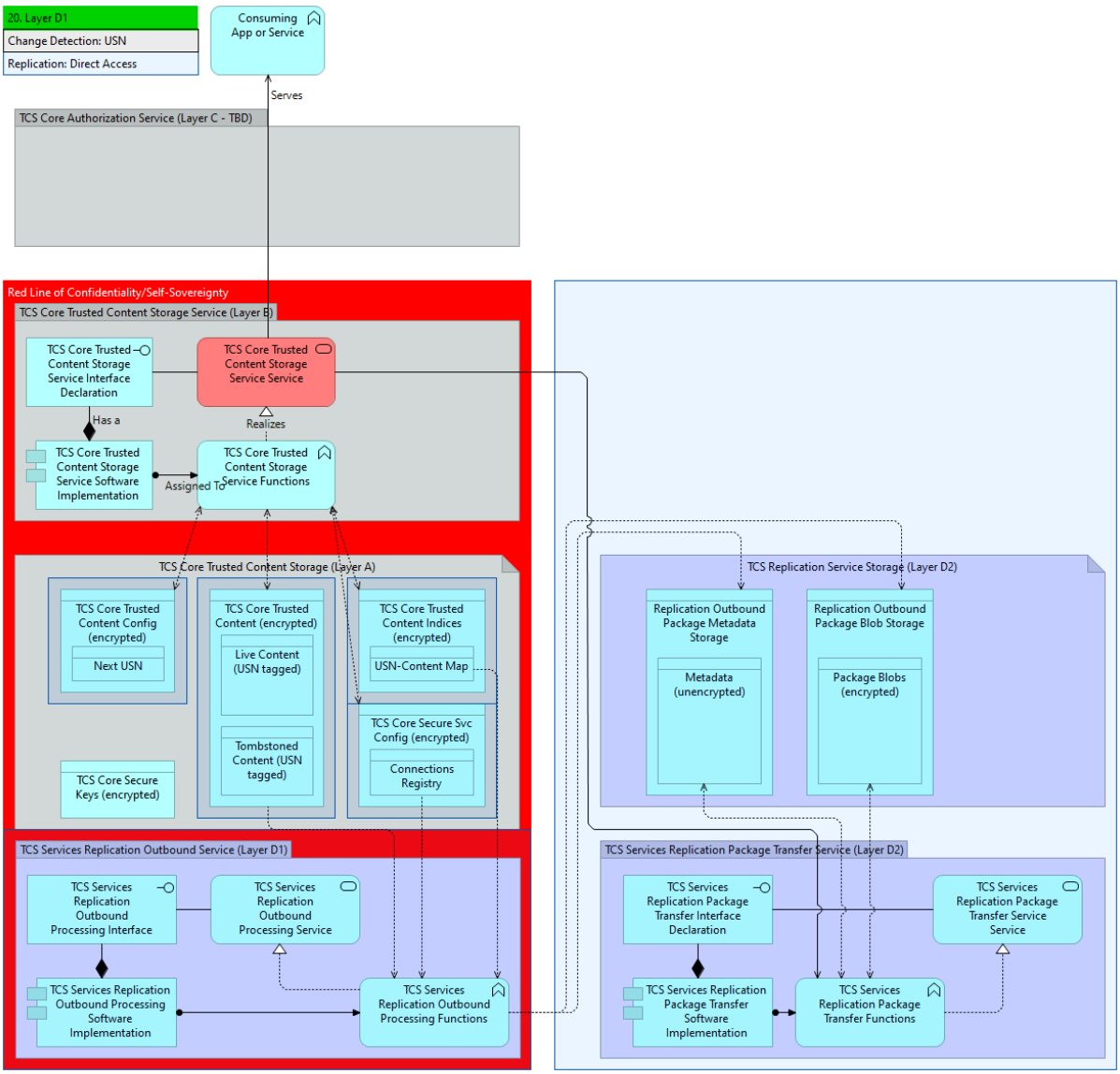


Figure 116. USN Content Change Detection/Tracking/Notification: Layer D1 Replication Outbound Processing Service & D2 Package Transfer Service [20]

TODO

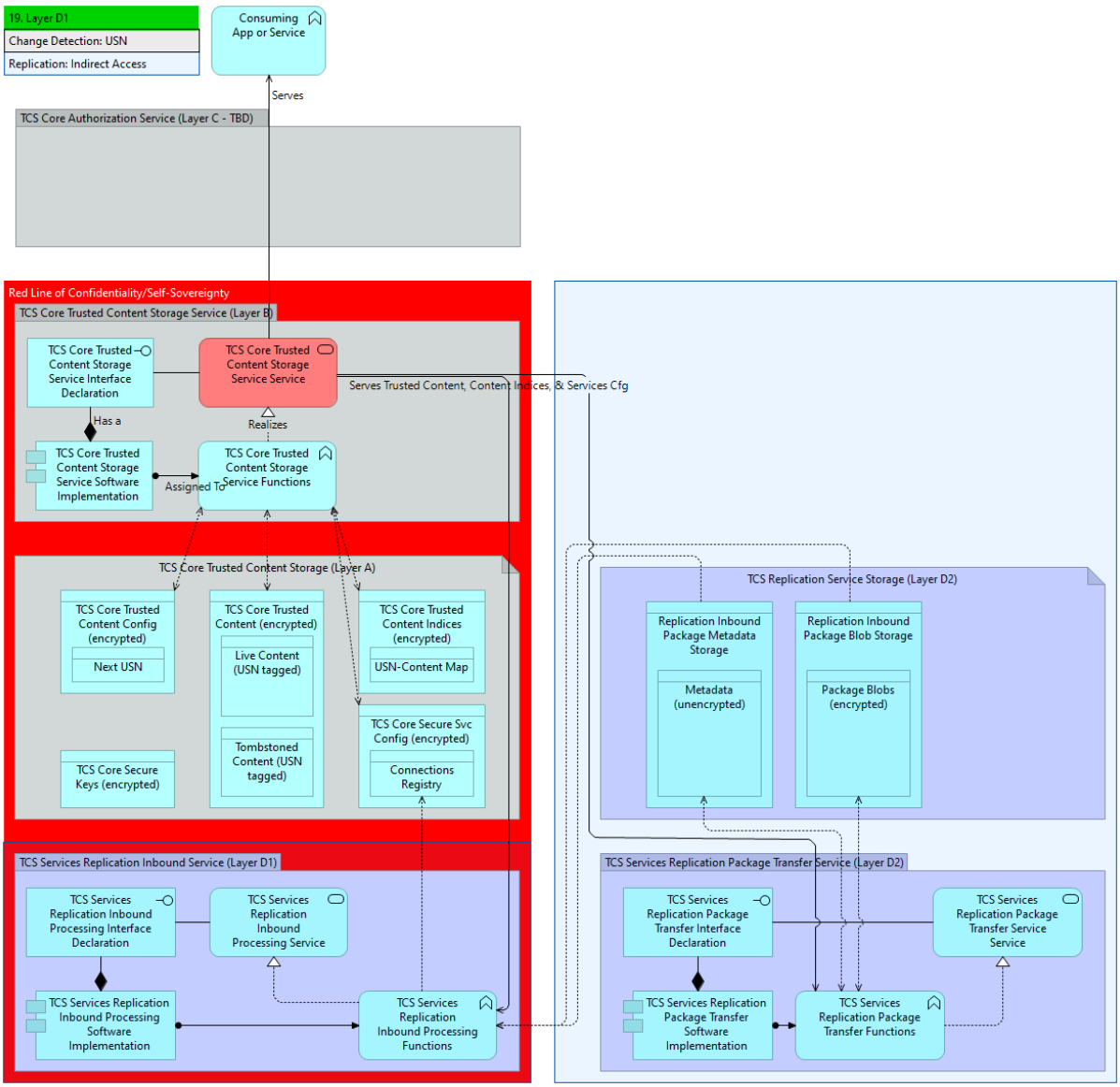


Figure 117. USN Content Change Detection/Tracking/Notification: Layer D1 Replication Inbound Processing Service and D2 Package Transfer Service [19]

TODO

Trusted Content Storage (TCS) Innovations

TODO

EDV Microkernel Architecture

TODO

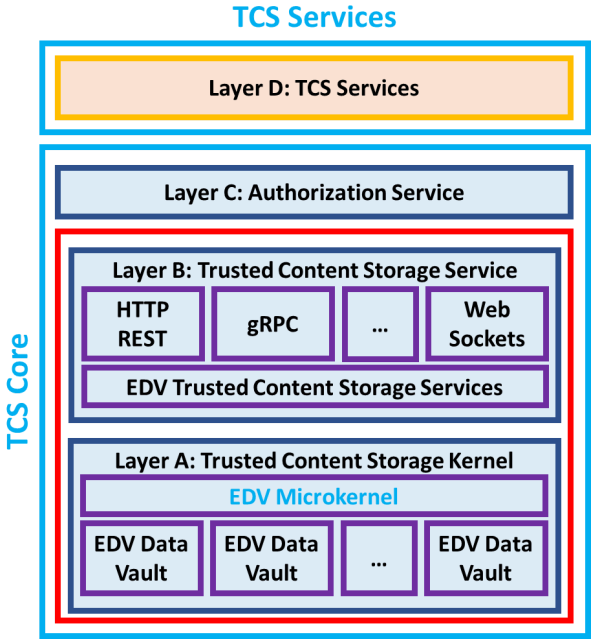


Figure 118. EDV Microkernel Architecture

Reference: https://github.com/decentralized-identity/confidential-storage/issues/181

Dependent on: TODO

Heterogeneous Multi-vaults/Server Instance Support

TODO

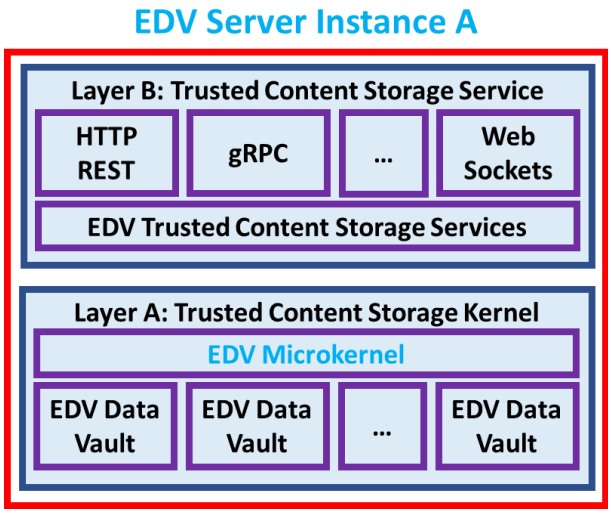


Figure 119. Homogeneous Multi-vaults/Server Instance Support

TODO

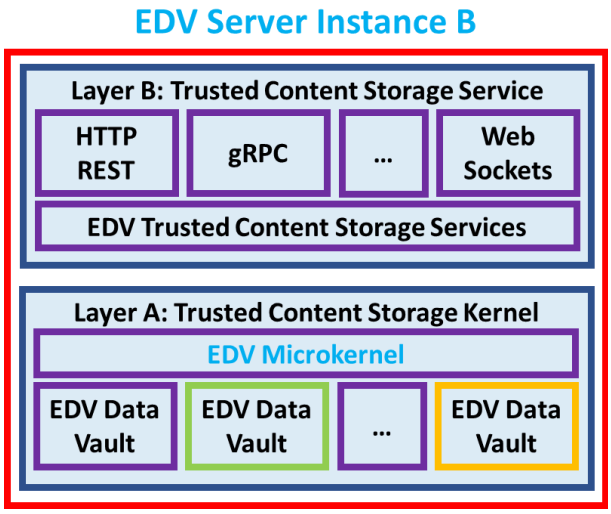


Figure 120. Heterogeneous Multi-vaults/Server Instance Support

Reference: https://github.com/decentralized-identity/confidential-storage/issues/189

Dependent on: TODO

Multi-vaults/Multi-Server Instance Deployment Model

TODO

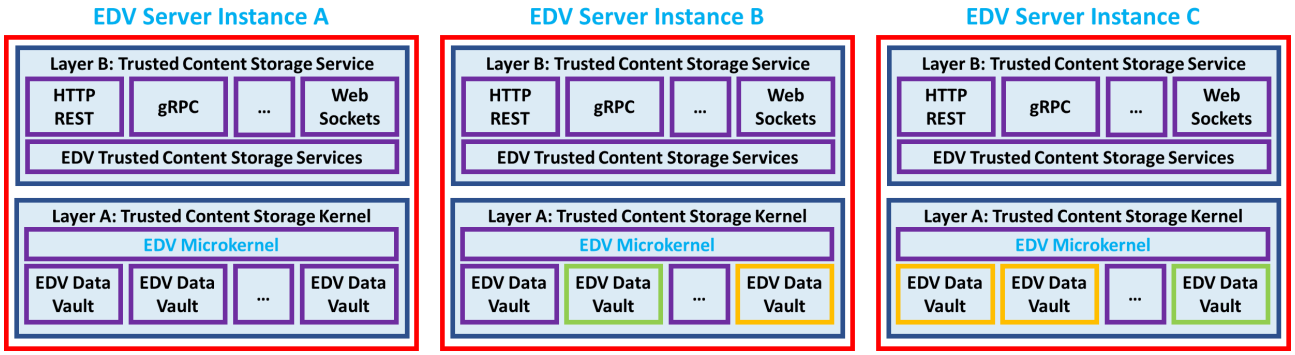


Figure 121. Multi-vaults/Multi-Server Instance Deployment Model

Reference: https://github.com/decentralized-identity/confidential-storage/issues/173

Dependent on: TODO

Multi-Resource/Multi-vaults/Multi-Server Instance Transaction Scopes

TODO

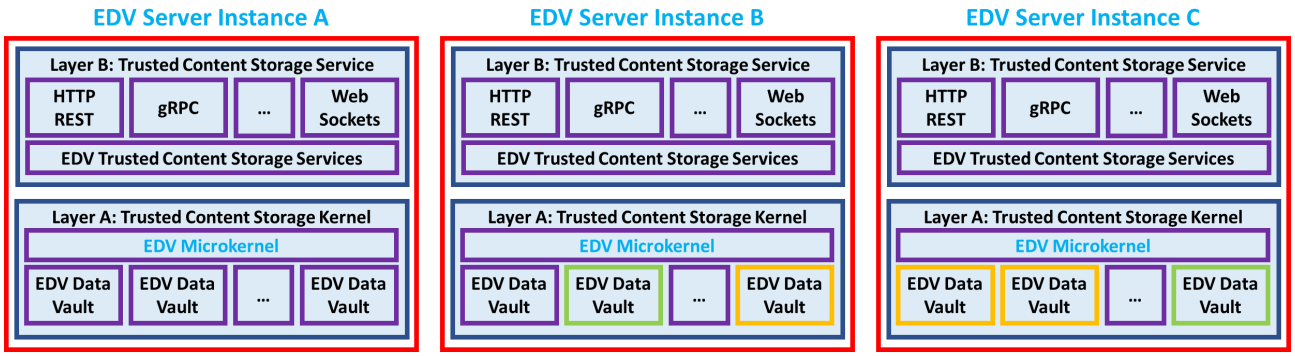


Figure 122. Multi-Resource/Multi-vaults/Multi-Server Instance Transaction Scopes

Reference: https://github.com/decentralized-identity/confidential-storage/issues/176

Dependent on: TODO

Logging of all operations/transactions

TODO

Reference: https://github.com/decentralized-identity/confidential-storage/issues/186

Dependent on: TODO

Directory Resource Schema

TODO

Reference: https://github.com/decentralized-identity/confidential-storage/issues/187

Dependent on: TODO

Trust Levels for Content Resources

TODO

Different categories of content resources have different trust levels depending on where and how they are stored.

Trust Level	Description	JWT Equivalent
Trust Level 0. Not Resolvable	The identified content resource is not resolvable as a content resource in an EDV Data Vault. All actions, whether successful or unsuccessful, are logged securely.	N/A
Trust Level 1. Unsigned Plain Text	The identified content resource is resolvable in an EDV Data Vault as an unsigned plain text resource. All actions, whether successful or unsuccessful, are logged securely.	Unsigned JWT
Trust Level 2. Signed Plain Text	The identified content resource is resolvable in an EDV Data Vault as a signed plain text resource. All actions, whether successful or unsuccessful, are logged securely.	JWS (Signed JWT)
Trust Level 3. Signed Encrypted	The identified content resource is resolvable in an EDV Data Vault as a signed encrypted resource. All actions, whether successful or unsuccessful, are logged securely.	JWE

Reference: https://github.com/decentralized-identity/confidential-storage/issues/188

Where Do We Go From Here?

[Strategists] need to 'think in time' linking an organization's past, present, and future in their thought processes. There are three components:

- *the predictive value of the past for the future;*
- *departures from the past which divert the organization from familiar patterns;*
- *the need for continuous comparison*

Jeanne Liedtaka in *The Business of Giving: The Theory and Practice of Philanthropy, Grantmaking and Social Investment* (https://www.amazon.ca/Business-Giving-Philanthropy-Grantmaking-Investment/dp/0230336795/ref=sr_1_1)

Current Status

TODO The current status of the SSI Personal Data Usage Licensing (SSI-PDUL) solution concept is as a process reference model as defined in this document. The next step towards adoption is to design and build a prototype app to actively demonstrate the concepts discussed in this whitepaper.

Technology Adoption Models

Careful consideration must be given to how a new solution as different and as important as the SSI Personal Data Usage Licensing (SSI-PDUL) Model is to the future of the Self-Sovereign Identity Ecosystem. Deployment and adoption are expected to be gradual and slow – full adoption taking place over several iterations and a significant amount of time.

A brief survey and discussion of a small number of technology adoption models taken from the article Technology Adoption Models: A Comprehensive Guide (https://hyperonomy.com/2019/10/16/technology-adoption-models/) is applicable. These include:

- 1. Crossing the Chasm: Technology Adoption Model
- 10. Technology Adoption Model illuminated by the Gartner Hype Cycle
- 19. Exponential Growth Model
- 20. Exponential Growth Model coupled with the Gartner Hype Cycle
- 2a. Social Evolution: Creation of a Nation State
- 2b. Social Evolution: Defining Principles

NOTE: To survey a comprehensive list of technology adoption models, check out the article Technology Adoption Models: A Comprehensive Guide (https://hyperonomy.com/2019/10/16/technology-adoption-models/).

Crossing the Chasm: Technology Adoption Model

Many people will be familiar with one of the original technology adoption models: The Technology Adoption Model. The Technology Adoption Model was originally described in the book *Crossing the Chasm, 3rd Edition: Marketing and Selling Disruptive Products to Mainstream Customers* (https://www.amazon.ca/Crossing-Chasm-3rd-Disruptive-Mainstream/dp/0062292986/ref=sr_1_1) and is depicted in the diagram below.

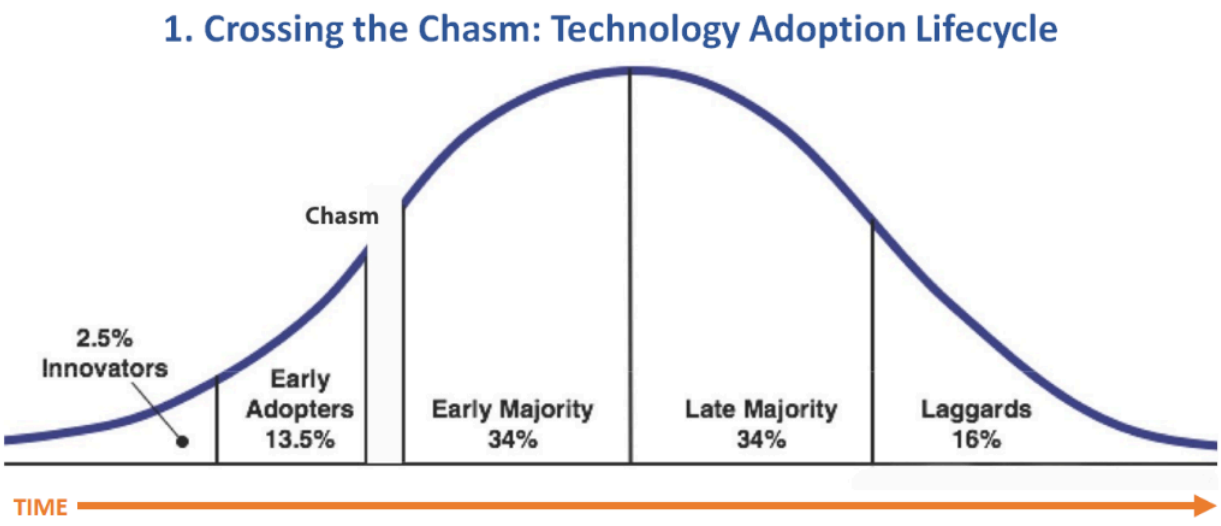


Figure 123. Model 1. Crossing the Chasm: Technology Adoption Lifecycle

While the Technology Adoption Lifecycle model depicted above is useful when explaining the need for a conservative, phased approach when introducing a new product or technology platform (in particular, a novel one), the model, while simplistic, does highlight where a project needs to start (at the left) and where most projects fail when they fail to only excite the Innovators and Early Adopters (at the Chasm).

A more interesting model results when the Technology Adoption Model is overlaid with the Gartner Hype Cycle. The Hype Cycle serves as a first derivative acceleration/deceleration curve (from Calculus). It illustrates what's happening "behind the scenes" through the Peak of Inflated Expectations and Trough of Disillusionment phases of the Cycle.

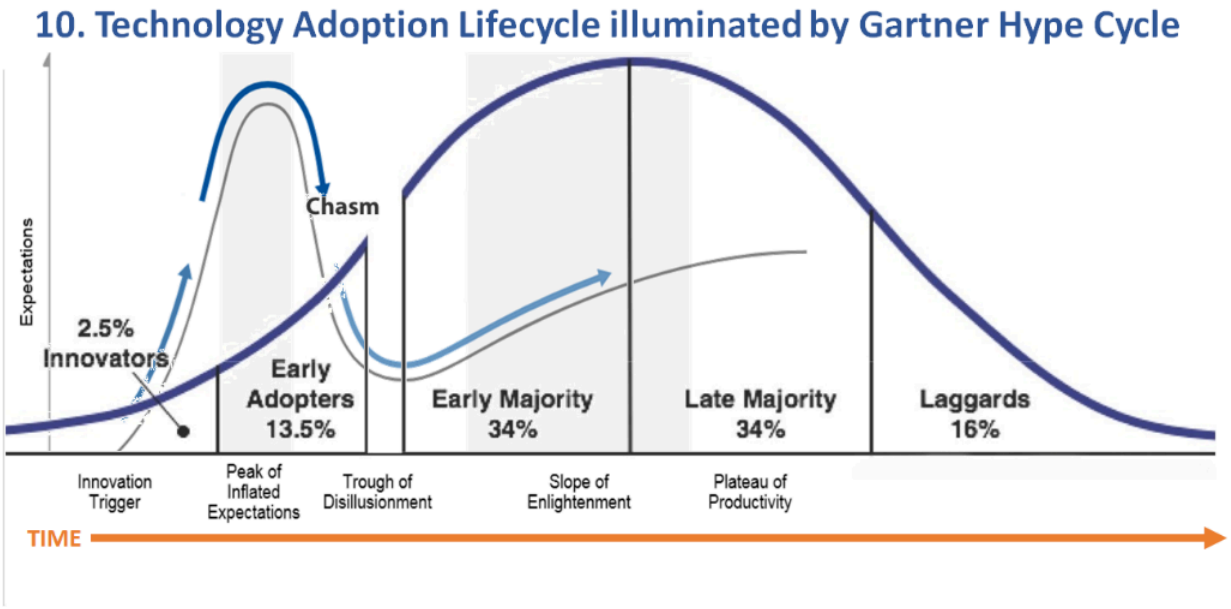


Figure 124. Model 10. Technology Adoption Lifecycle illuminated by the Gartner Hype Cycle

Another fallacy is the adoption of new products and services often takes place at an exponential pace (exponential growth) as depicted in the diagram below taken from a recent industry report of digital wallets.

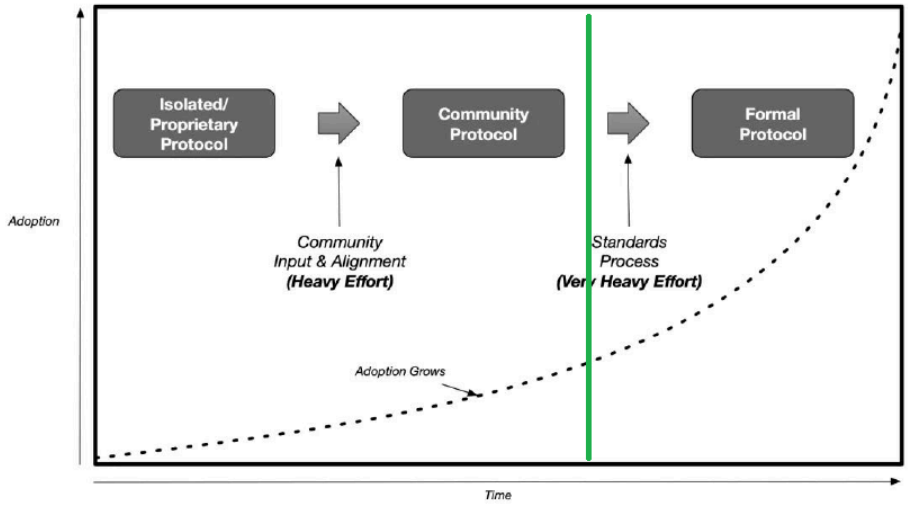


Figure 125. Model 19. Exponential Growth Model

Again, it's instructive to position an early-market exponential growth in the context of the Gartner Hype Cycle as shown below.

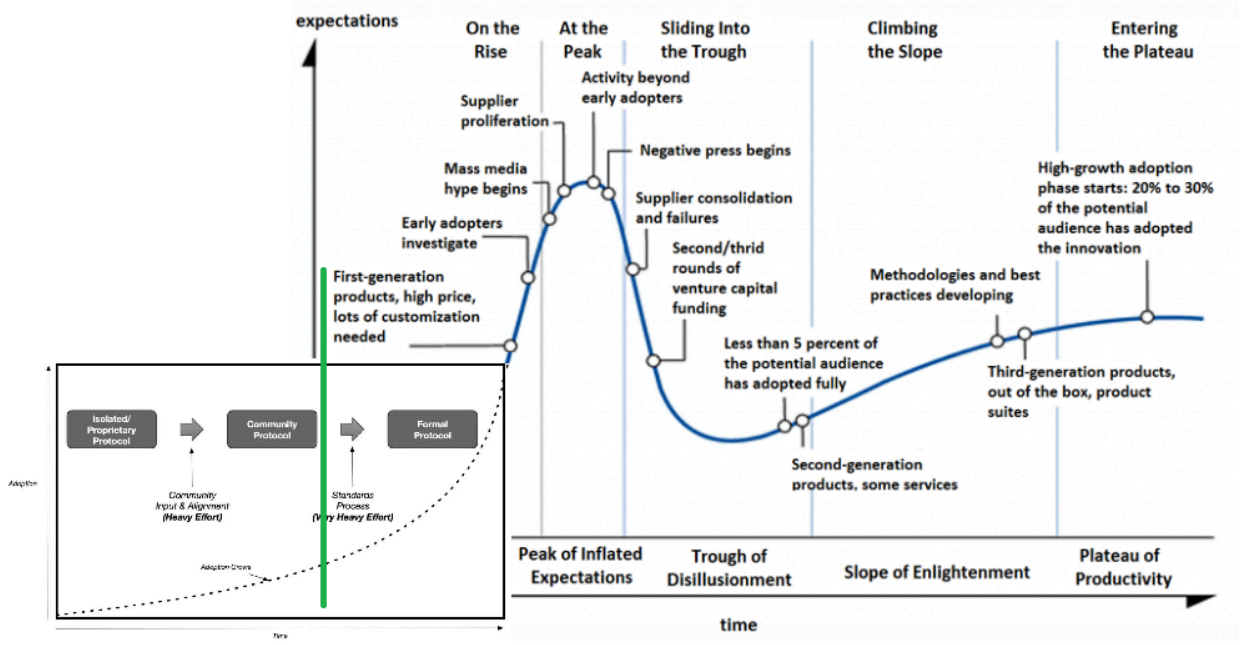


Figure 126. Model 20. Exponential Growth Model coupled with the Gartner Hype Cycle

Exponential growth by early market participants (i.e. unicorns) is often accompanied by a great deal of promotion (hype) until, again, the Peak of Inflated Expectations is reached, and the market caves in on itself. Eventually, if the technology is able to demonstrate ongoing promise and represents a true value differentiator relative to what is being used to solve a similar set of problems today, then the technology may be able to cross the Trough of Disillusionment and over to the lands of Enlightenment and Productivity.

Why does all this matter? More often than not, successful technology adoption more often reflects a social evolution as depicted in the following two models:

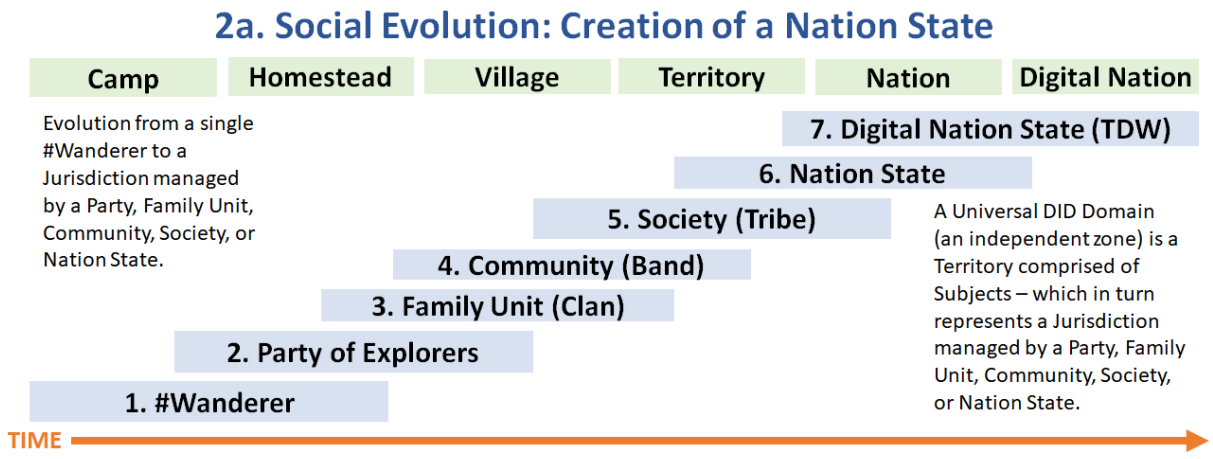


Figure 127. Model 2a. Social Evolution: Creation of a Nation State

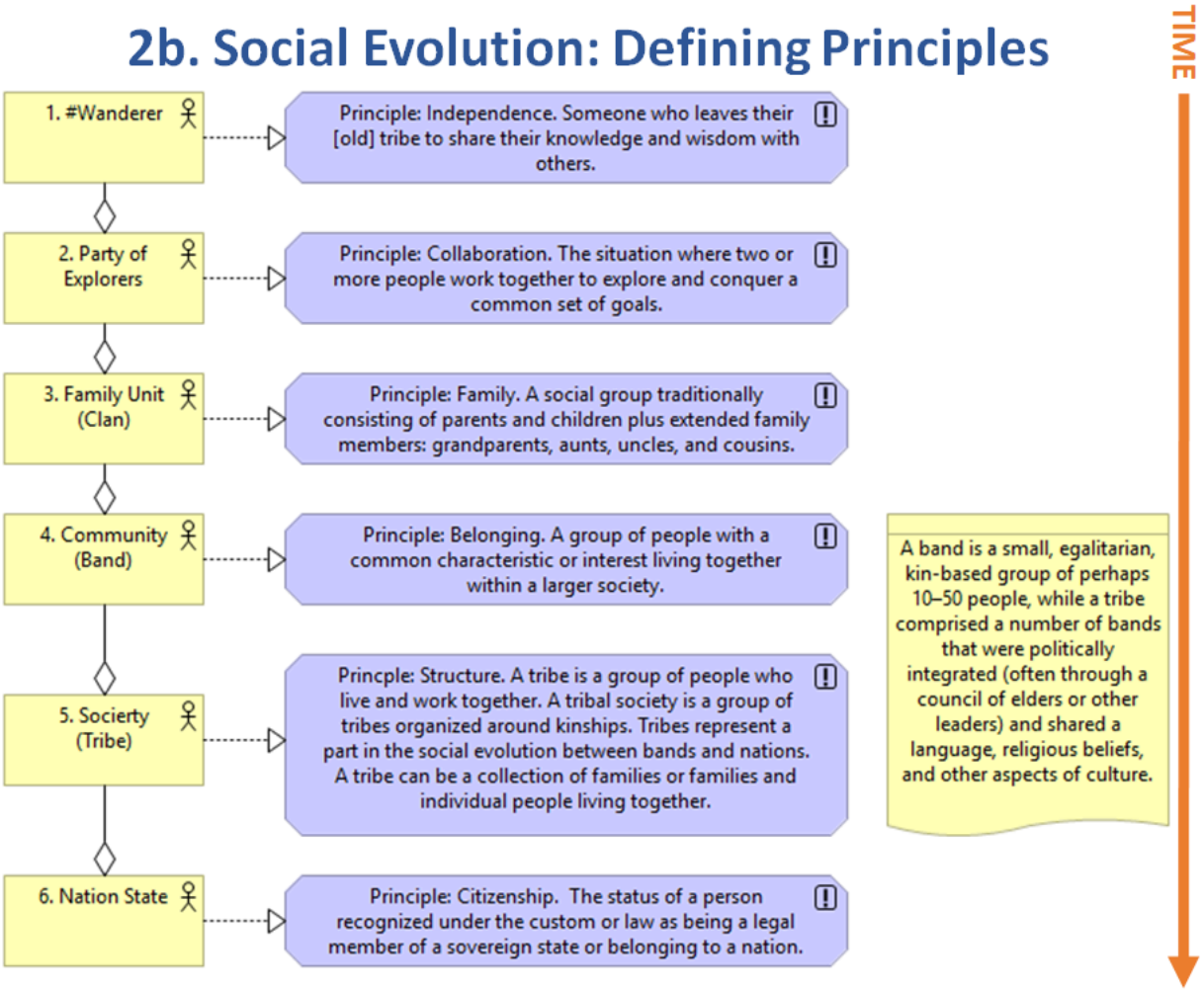


Figure 128. Model 2b. Social Evolution: Defining Principles

This is especially true for game-changing initiatives like the Self-Sovereign Identity Model and solutions like the SSI Personal Data Usage Licensing (SSI-PDUL) Model process reference model.

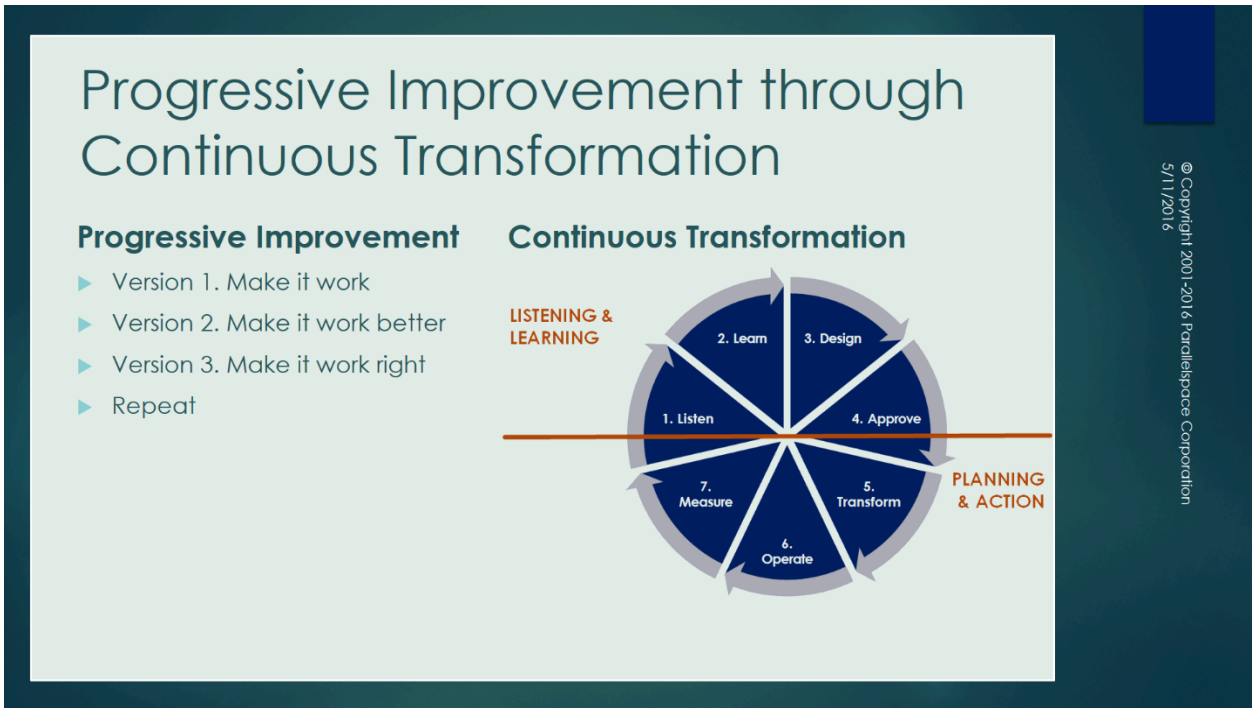
Next Steps

A #wanderer is someone who leaves their tribe to share their knowledge and wisdom with others; to later form a party of explorers to explore and conquer a common set of goals; and, even further on, create a clan, a band, a tribe, and a tribal society, a group of people who live and work together – a group of tribes organized around kinships.

Michael Herman, *Model 2b. Social Evolution: Defining Principles* in *Technology Adoption Models: A Comprehensive Guide* (*https://hyperonomy.com/2019/10/16/technology-adoption-models/*)

Progressive Improvement through Continuous Transformation

TODO How are we going to get there? The answer is progressive improvement through continuous transformation – as depicted below.



[How we think about how we work (https://hyperonomy.com/2016/05/09/how-do-we-think-about-how-we-work/)]

This whitepaper is the first attempt to describe the SSI Personal Data Usage Licensing (SSI-PDUL) Model in its entirety – both its motivations as well as a reasonable description of the complete, integrated solution. Full deployment will take several iterations and a significant amount of time – perhaps as long as the basic timeframe that it took the World Wide Web (running on top of the Internet) to develop into its current state.

A key advantage of the SSI Personal Data Usage Licensing (SSI-PDUL) Model is that it builds directly on top of existing Internet technologies, international standards (and specifications), and many open-source realizations of these technologies and standards.

Conclusion

If I have seen further, it is by standing on the shoulders of Giants.

[Issac Newton, 1675]

TODO The above quotation is absolutely true when describing the gestation of the SSI Personal Data Usage Licensing solution concept. In addition, to quote a colleague[11],

Otherwise, I think it's a pretty unstudied problem.

This whitepaper is a description of a conceptual solution to the SSI Personal Data Usage Licensing (SSI-PDUL) Model problem.

The next steps are to proceed with a prototype implementation of a proof-of-concept of all the features described in this document.

Acknowledgments

TODO

The responsibility for any errors or omissions lies with me alone.

[11] Daniel Hardman, Personal Communication, January 23, 2021.

APPENDIX A – REPLICATION SERVICES: PRIOR ART

TODO

Syntergy Replicator for SharePoint

TODO

Replication Topologies

TODO

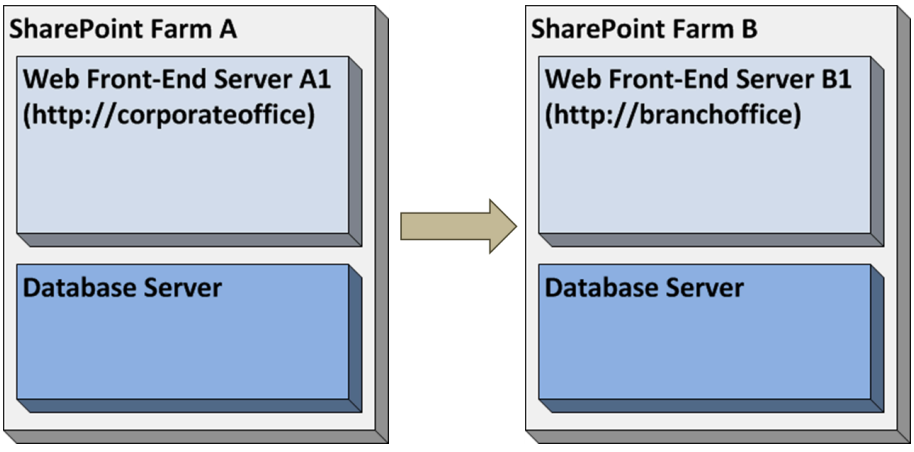


Figure 129. One-way Replication

TODO

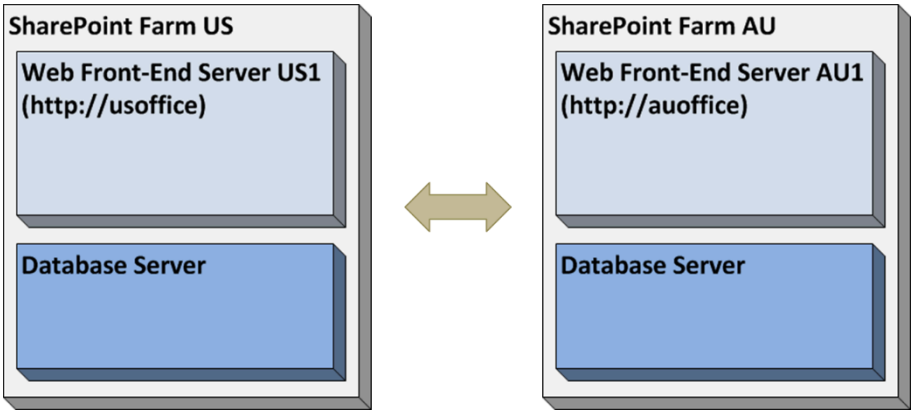


Figure 130. Two-way Synchronization

TODO

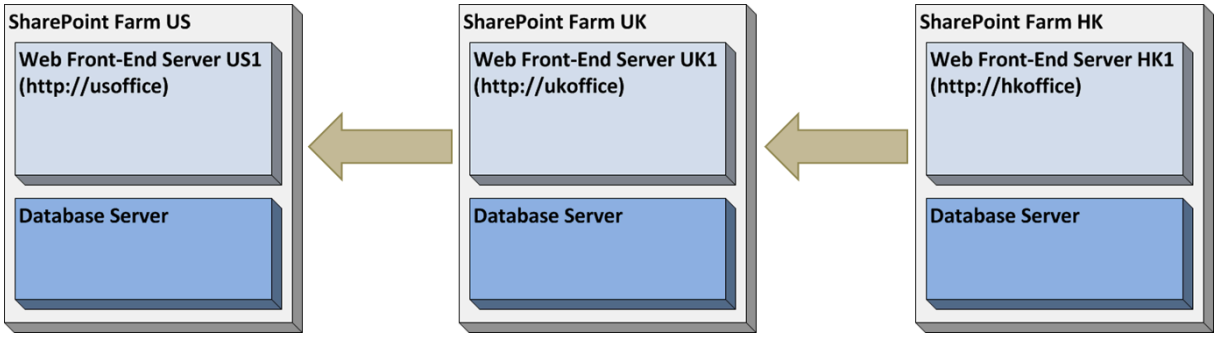


Figure 131. One-way Serial (and Cyclic) Replication

TODO

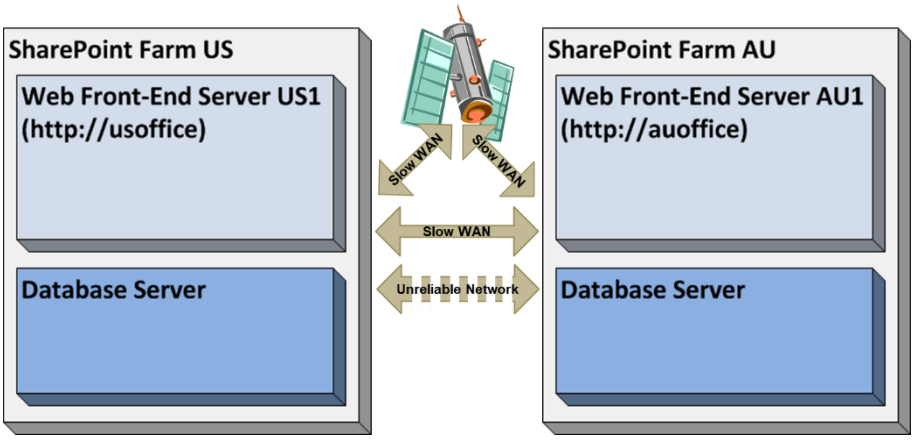


Figure 132. Slow, Unreliable, Intermittent, and High-latency Connections

TODO

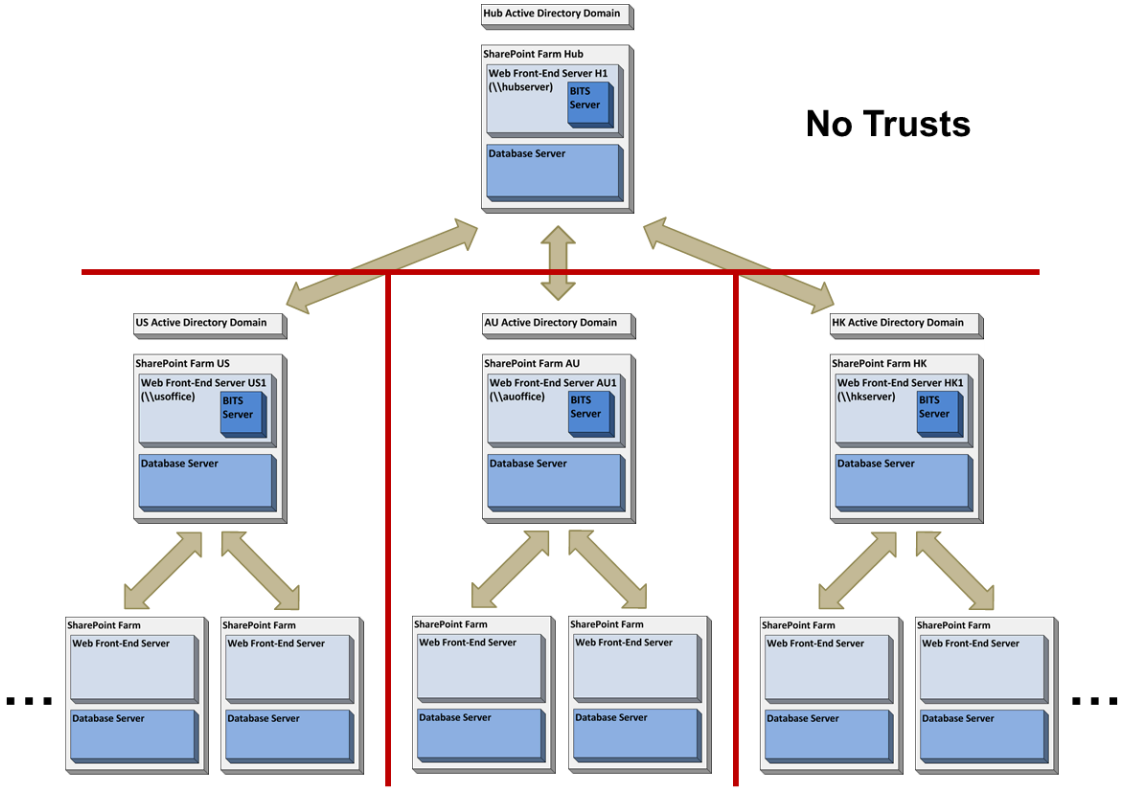


Figure 133. Federated Hub & Spoke Synchronization

TODO

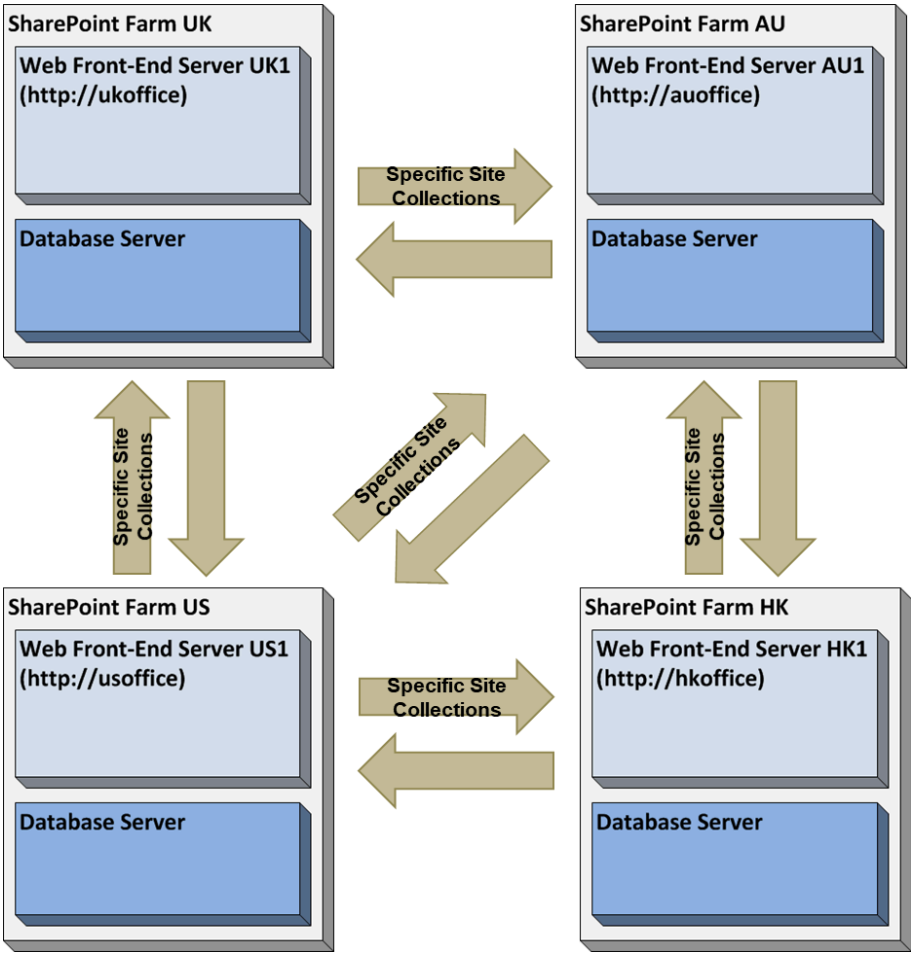


Figure 134. Sparse Mesh Synchronization

TODO

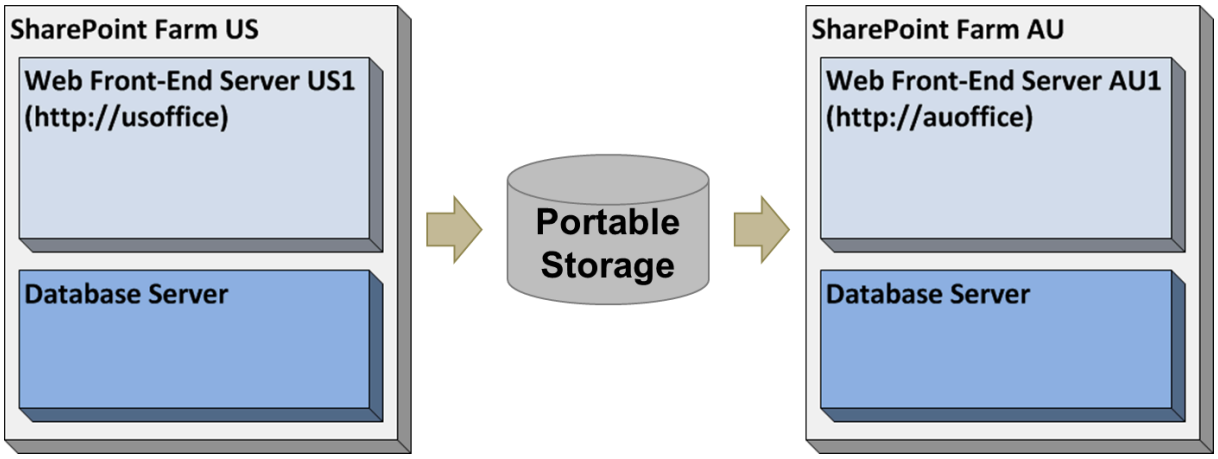


Figure 135. Offline or Air-gap Replication

TODO

Replication Pipeline

The Replication Pipeline is illustrated below.

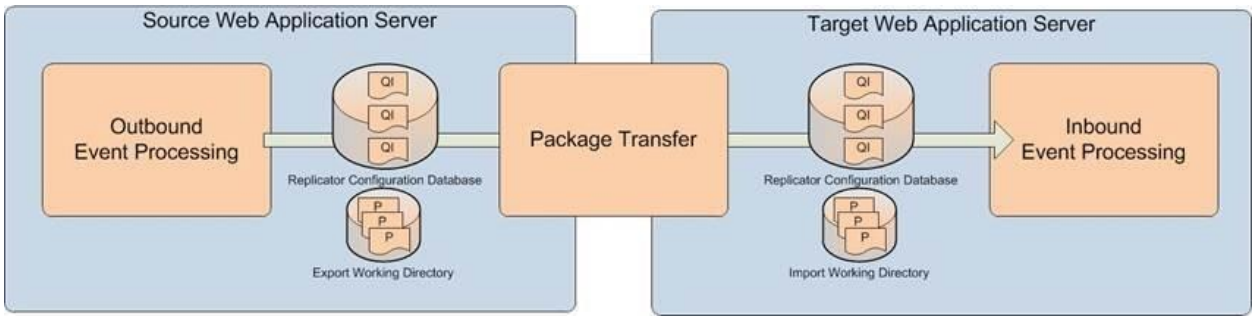


Figure 136. Syntergy Replicator for SharePoint: Replication Pipeline

Outbound Processing Service

Outbound Processing Service is responsible for capturing and recording Replication Events that occur in the Source Repository. Outbound Processing Service is controlled by Replication Maps which determine what Events need to be captured, packaged, and transferred to the Target Repository. Groups of Replication Events are packaged into two types of messages or objects: Queued Items and Replication Packages.

Queued Item

A Queued Item is a unit of work to be transferred to a Target Repository for remote execution. The Replicator Web Service on a Target Repository is called to push a Queued Item from the Source Repository to a Target Repository.

Replication Package

A Replication Package is a collection of one or more Replication Events plus data about the changed information that is packaged in a format specific to the Replication Transport being used. When an Event is being processed, Outbound Processing Service calls the Source repository object model to extract the changed information from the Source Repository.

Package Transfer Service

The Package Transfer Service activity is responsible for the transfer of Queued Items and Replication Packages (Packages) from the Source Repository to the Target Repository. Package Transfer is the process that sits between Outbound Processing Service (on the Source Repository) and Inbound Processing Service (on the Target Repositories).

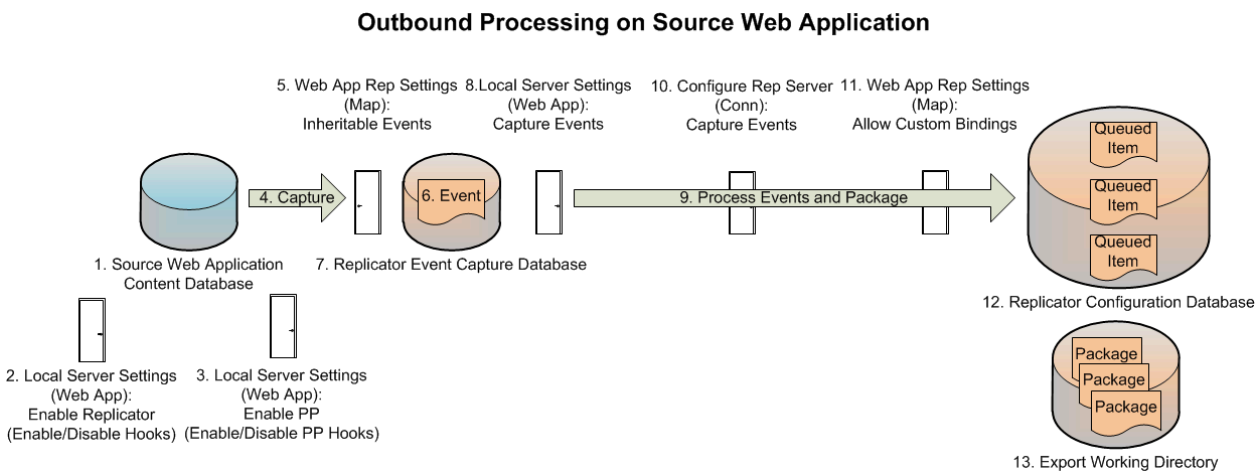
Inbound Processing Service

Inbound Processing Service is responsible for processing the Queued Items and Packages received and accepted by the Target Repository and applying them to the repository. The Queued Items and Packages are applied to the Target Repository content base by calling the Target Repository object model.

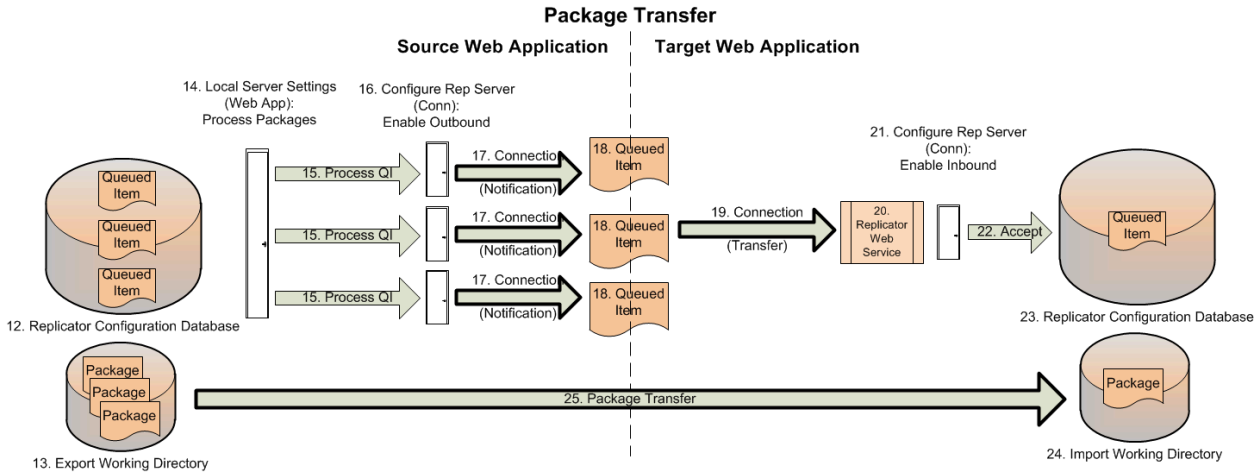
Replication Pipeline Terminology

Term	Definition
Queued Item	A Queued Item is a unit of work to be transferred to a Target Repository for remote execution. The Replicator Web Service on a Target Repository is called to push a Queued Item from the Source Repository to a Target Repository.
Replication Connection (Connection)	A network path over which Queued Items and Packages are transferred from one Source Repository to a Target Repository as part of Package Transfer Service.
Replication Event (Event)	A Replication Event is a change in a SharePoint Farm's content or its configuration that is captured and logged by the Replicator Engine. Each version of Replicator supports a specific list of Events. There are two methods by which Events are created: Captured and Queued.
Replication Package	A Replication Package is a collection of one or more Replication Events plus additional SharePoint state information that is packaged in a format specific to the Replication Transport being used.

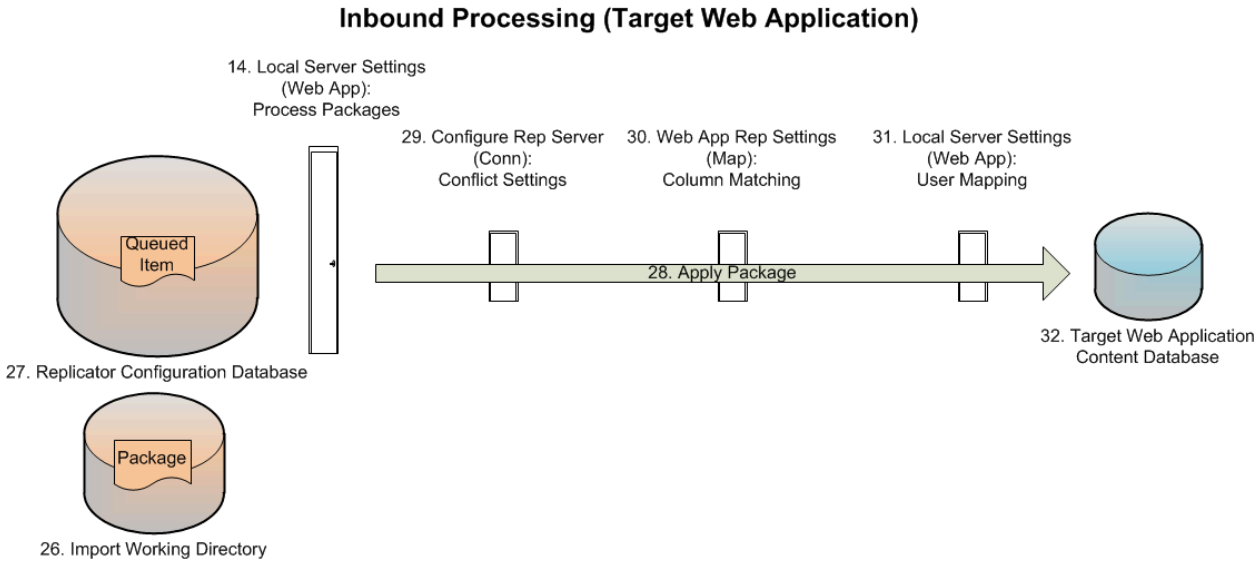
TODO



TODO


Package Transfer
Source Web Application
Target Web Application
14. Local Server Settings (Web App): Process Packages
16. Configure Rep Server (Conn): Enable Outbound
15. Process QI
17. Connectio (Notification)
18. Queued Item
19. Connection (Transfer)
20. Replicator Web Service
21. Configure Rep Server (Conn): Enable Inbound
22. Accept
Queued Item
12. Replicator Configuration Database
23. Replicator Configuration Database
Package
13. Export Working Directory
25. Package Transfer
24. Import Working Directory

TODO


Inbound Processing (Target Web Application)
14. Local Server Settings (Web App): Process Packages
29. Configure Rep Server (Conn): Conflict Settings
30. Web App Rep Settings (Map): Column Matching
31. Local Server Settings (Web App): User Mapping
Queued Item
28. Apply Package
32. Target Web Application Content Database
27. Replicator Configuration Database
Package
26. Import Working Directory

TODO

Groove Workspace

TODO

Groove Workspace supports real-time, secure, bi-directional, multi-master, peer-to-peer (direct and mediated), cross-firewall replication between desktop clients[12].

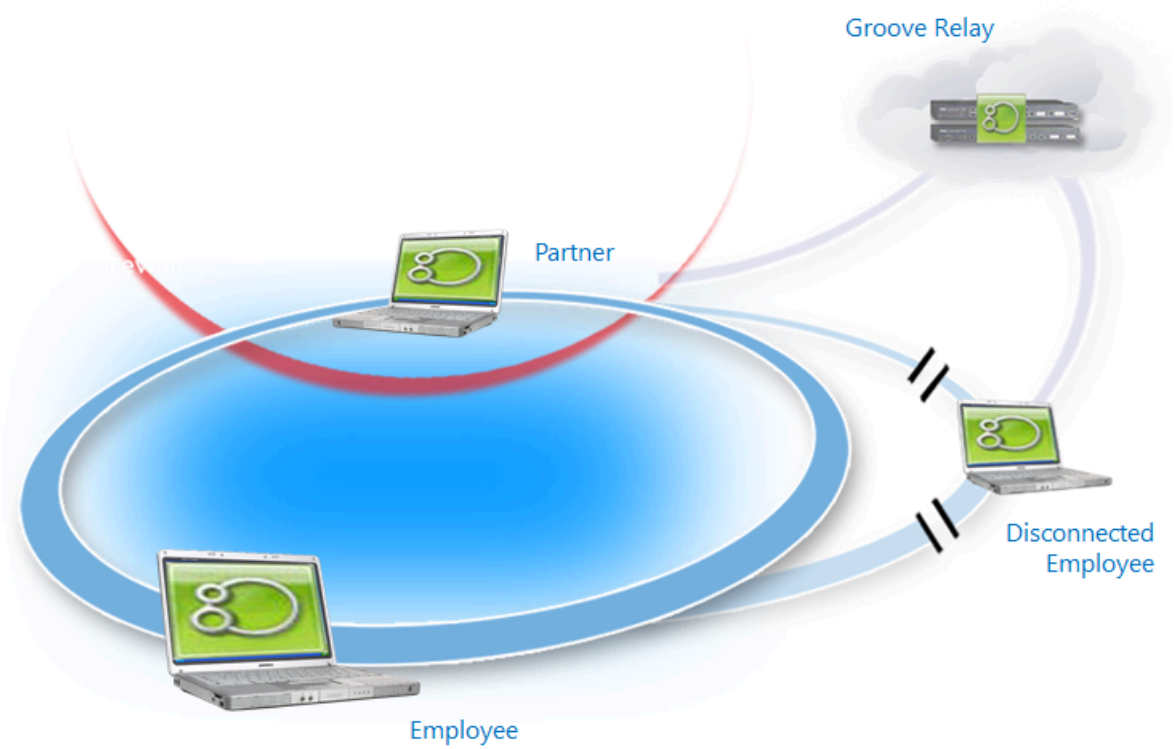


Figure 137. Groove Workspace: P2P Replication Model

TODO

[12] All the adjectives are necessary and correct and orthogonal to each other …contrary to some discussions.

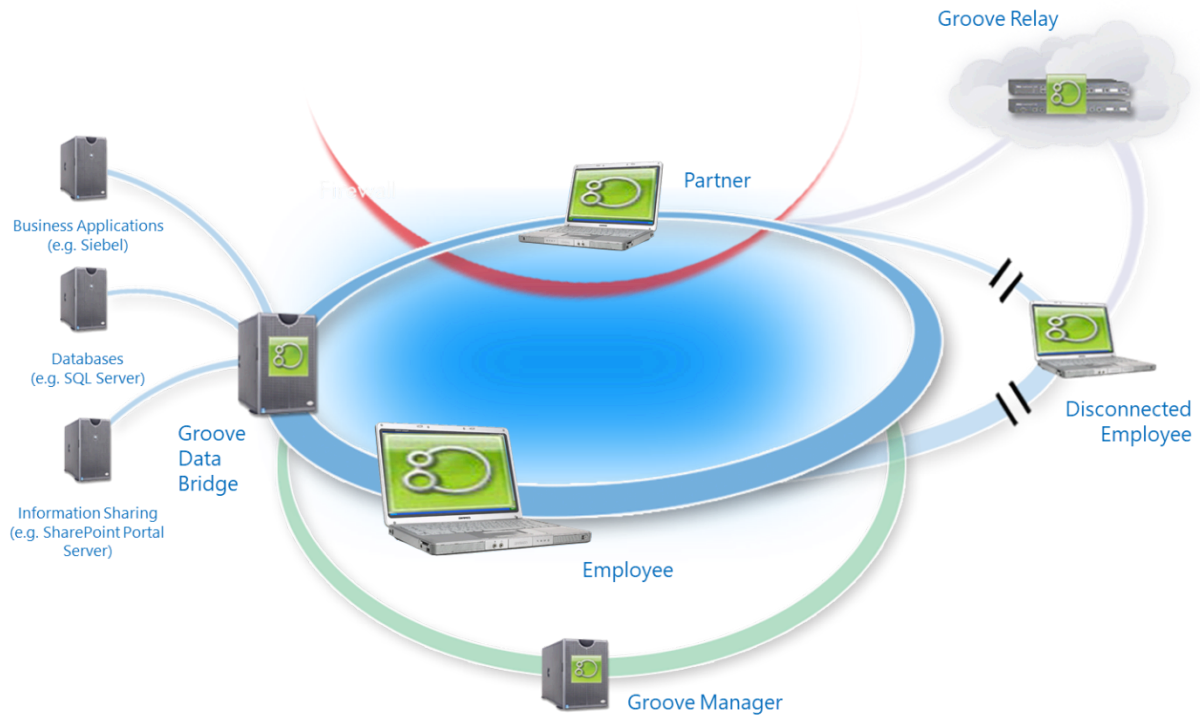


Figure 138. Groove Workspace: Enterprise Integration

TODO

Microsoft Active Directory

TODO

Reference:
https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc772726(v=ws.10)?redirectedfrom=MSDN#determining-changes-to-replicate-update-sequence-numbers

Enterprise Replication Scenarios

Reference:
https://social.technet.microsoft.com/wiki/contents/articles/16968.active-directory-concepts-part-1.aspx

TODO

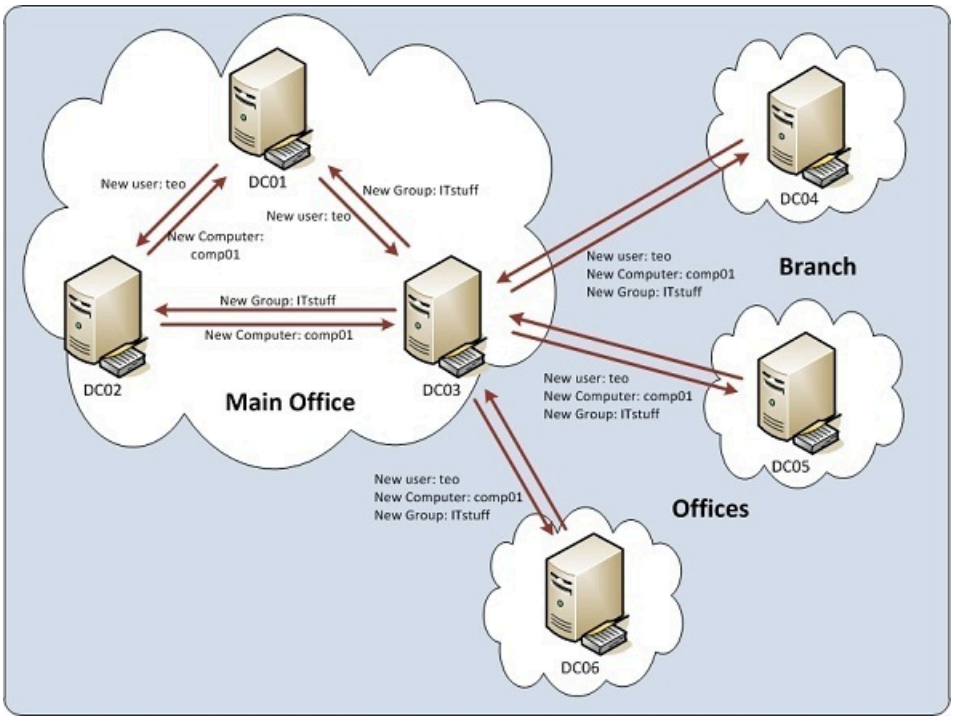


Figure 139. Active Directory: Sample Enterprise Replication Scenario

Replication Protocol

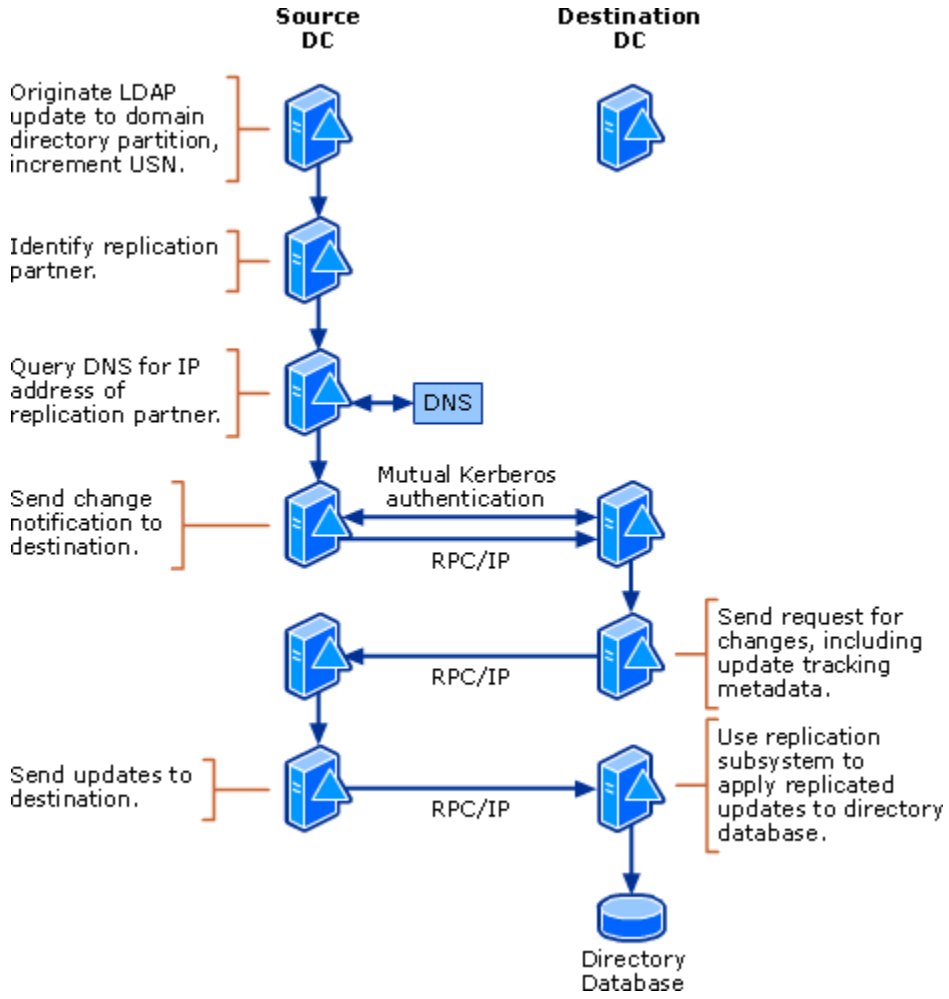


Figure 140. Active Directory: Replication Protocol

Update Sequence Numbers (USNs)

The following diagram shows the replication-related data for the user object when it is first created on domain controller DC1. Before the user object is created, the current USN for the domain controller is 4710. When the object is created, the local USN of 4711 is assigned to each attribute of the user object, and the current USN for the domain controller increments from 4710 to 4711.

Property	Value	Local USN	Version	Originating Time	Originating DC	Originating USN
cn	Jeff Smith	4711	1	2003-09-10 10:49.03	<DC1_GUID>	4711
userPassword	6Be8W5q-	4711	1	2003-09-10 10:49.03	<DC1_GUID>	4711
sAMAccountName	JSmith	4711	1	2003-09-10 10:49.03	<DC1_GUID>	4711
userPrincipalName	JSmith@contoso.com	4711	1	2003-09-10 10:49.03	<DC1_GUID>	4711

Figure 141. Active Directory: Update Sequence Numbers (USNs)

TODO

Active Directory replication does not primarily depend on time to determine what changes need to be propagated. Instead, it uses update sequence numbers (USNs) that are assigned by a counter that is local to each domain controller. Because these USN counters are local, it is easy to ensure that they are reliable and never run backward (that is, they cannot decrease in value).

Failures and Disaster Recovery

A source domain controller uses USNs to determine what changes have already been received by a destination domain controller that is requesting changes. The destination domain controller uses USNs to determine what changes it needs to request.

The current USN is a 64-bit counter that is maintained by each Active Directory domain controller as the highestCommittedUsnattribute on the rootDSE object. At the start of each update transaction (originating or replicated), the domain controller increments its current USN and associates this new value with the update request.

Local USN: The USN for the update is stored in the metadata of each attribute that is changed by the update as the local USN of that attribute (originating and replicated writes). As the name implies, this value is local to the domain controller where the change occurs.

Destination domain controllers use the originating USN to track changes they have received from other domain controllers with which they replicate. When requesting changes from a source domain controller, the destination informs the source of the updates it has already received so that the source never replicates changes that the destination does not need.

Multi-master Conflict Resolution Policy

Active Directory must ensure that all domain controllers agree on the value of the updated attribute after replication occurs. The general approach to resolving conflicts is to order all update operations (Add, Modify, Move, and Delete) by assigning a globally unique (per-object and per-attribute) stamp to the originating update. Thus each replicated attribute value (or multivalue) is stamped during the originating update and this stamp is replicated with the value.

Conflict Resolution Stamp

The stamp that is applied during an originating write has the following three components:

A version is a number that is incremented for each originating write. The version of the first originating write is 1. The version of each successive originating write is increased by 1.

The originating time is the time of the originating write, to a one-second resolution, according to the system clock of the domain controller that performed the write.

The originating DC is the invocationID of the domain controller that performed the originating write.

When stamps are compared, the version is the most significant, followed by the originating time and then the originating DC. If two stamps have the same version, the originating time almost always breaks the tie. In the extremely rare event that the same attribute is updated on two different domain controllers during the same second, the originating DC breaks the tie in an arbitrary fashion.

Two different originating writes of a specific attribute of a particular object cannot assign the same stamp because each originating write advances the version at a specified originating domain controller. The originating time does not contribute to uniqueness. Replicated writes cannot decrease the version because values with smaller versions lose during conflict resolution.

TODO

Microsoft Sync Foundation

Reference:
https://docs.microsoft.com/en-us/previous-versions/mt763482(v=msdn.10)?redirectedfrom=MSDN

Reference:
https://docs.microsoft.com/en-us/previous-versions/sql/synchronization/mt490616(v=msdn.10)?redirectedfrom=MSDN

Reference: https://en.wikipedia.org/wiki/Microsoft_Sync_Framework

Sample Sync Foundation Scenario

TODO

Figure 142. Microsoft Sync Framework: P2P Synchronization for any Pair of Devices

TODO

Sync Foundation Architecture

Reference:
https://www.slideshare.net/goodfriday/using-the-microsoft-sync-framework-and-feedsync?qid=a3ad4f7c-1620-435b-8ed5-44e489e3123c&v=&b=&from_search=3

TODO

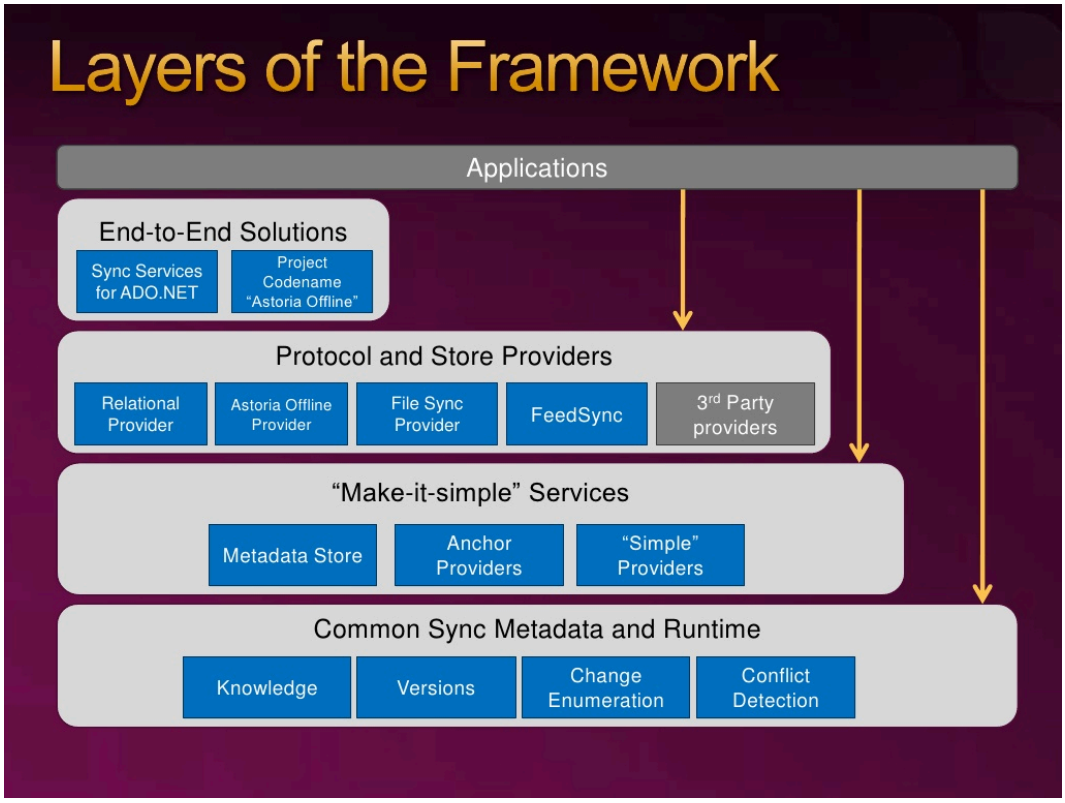


Figure 143. Sync Foundation: Layered Architecture

TODO

Sync Provider Model: Core Components

TODO

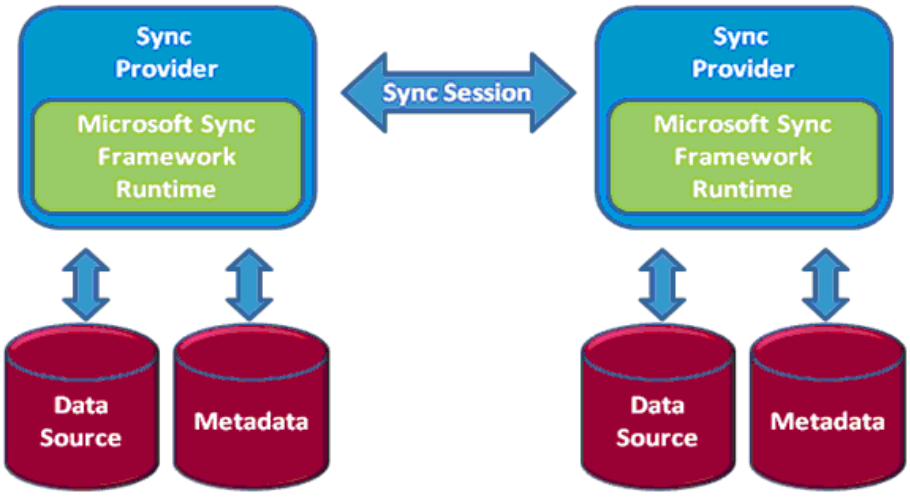


Figure 144. Microsoft Sync Framework: Sync Provider Model: Core Components

TODO

Sync Framework Protocol

TODO

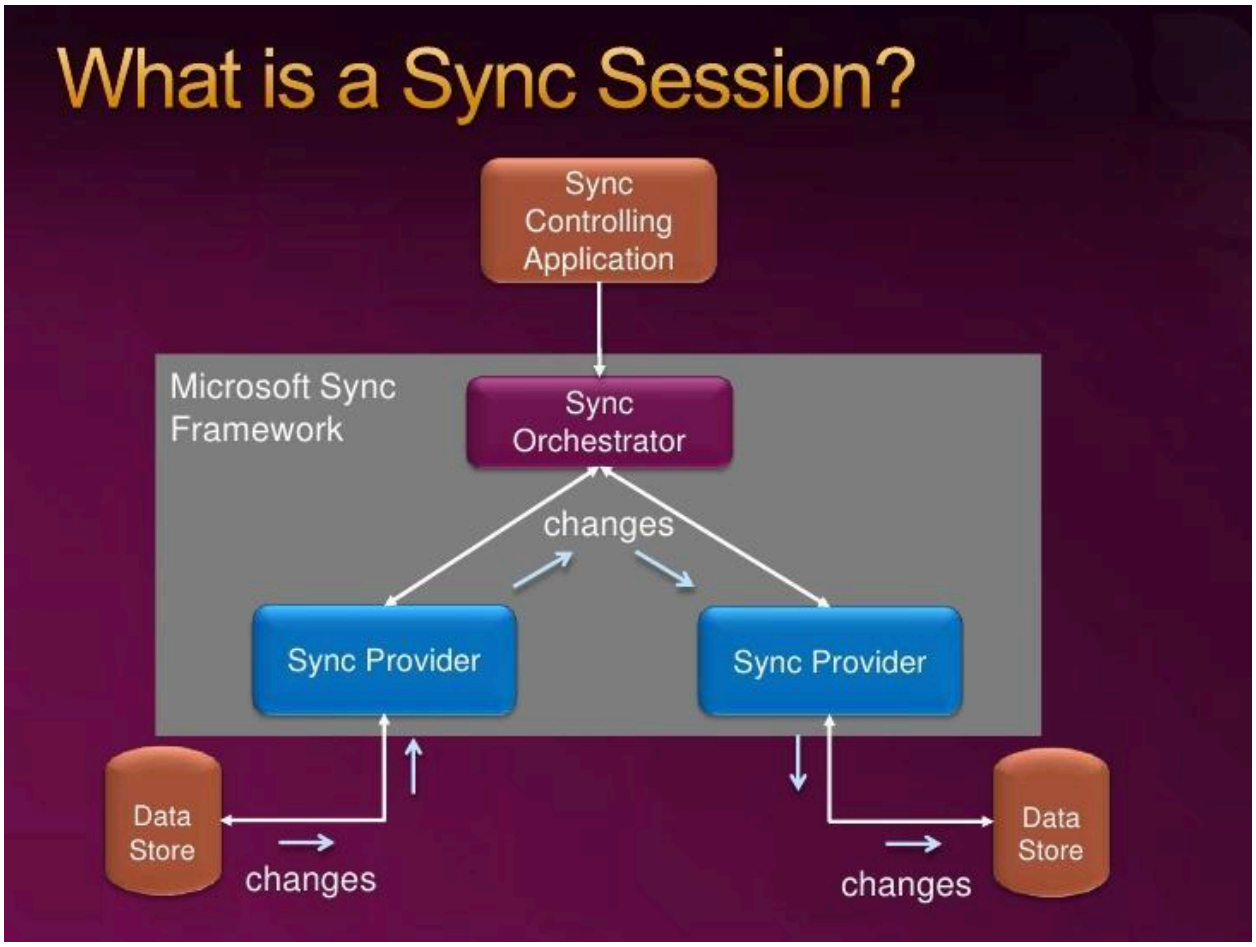


Figure 145. Sync Foundation: Sync Sessions

TODO

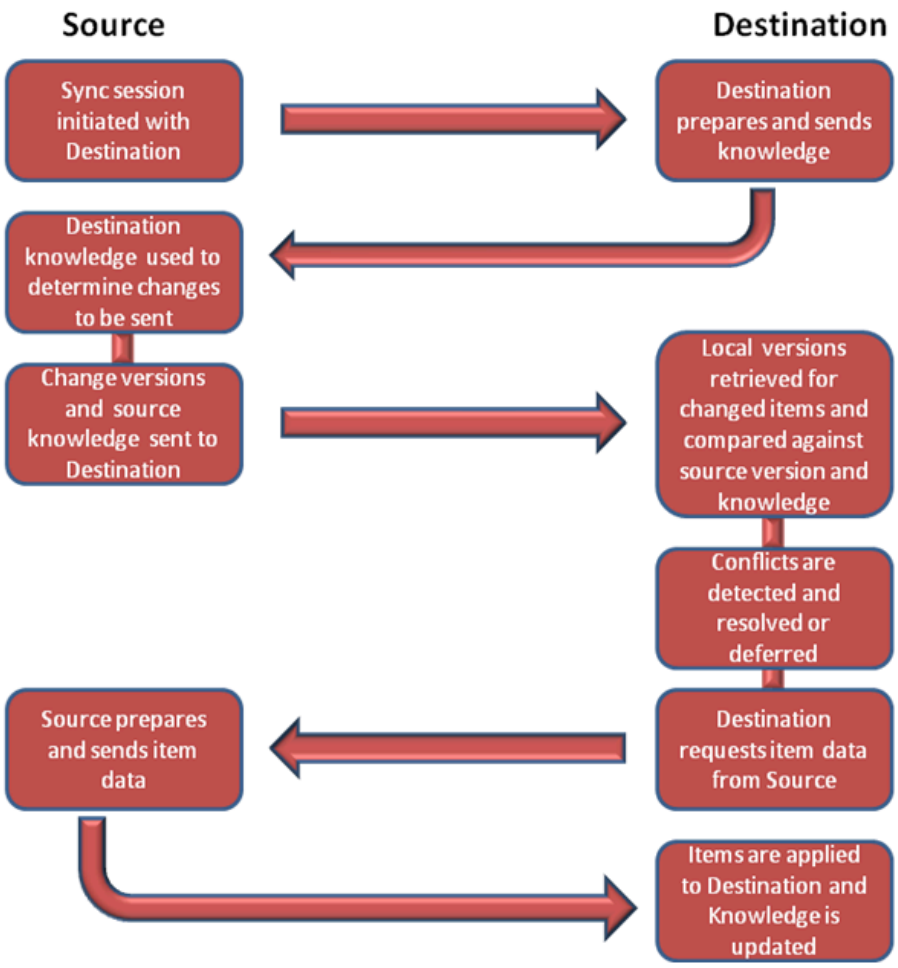


Figure 146. Microsoft Sync Framework: Sync Framework Protocol

TODO

FeedSync Provider

TODO

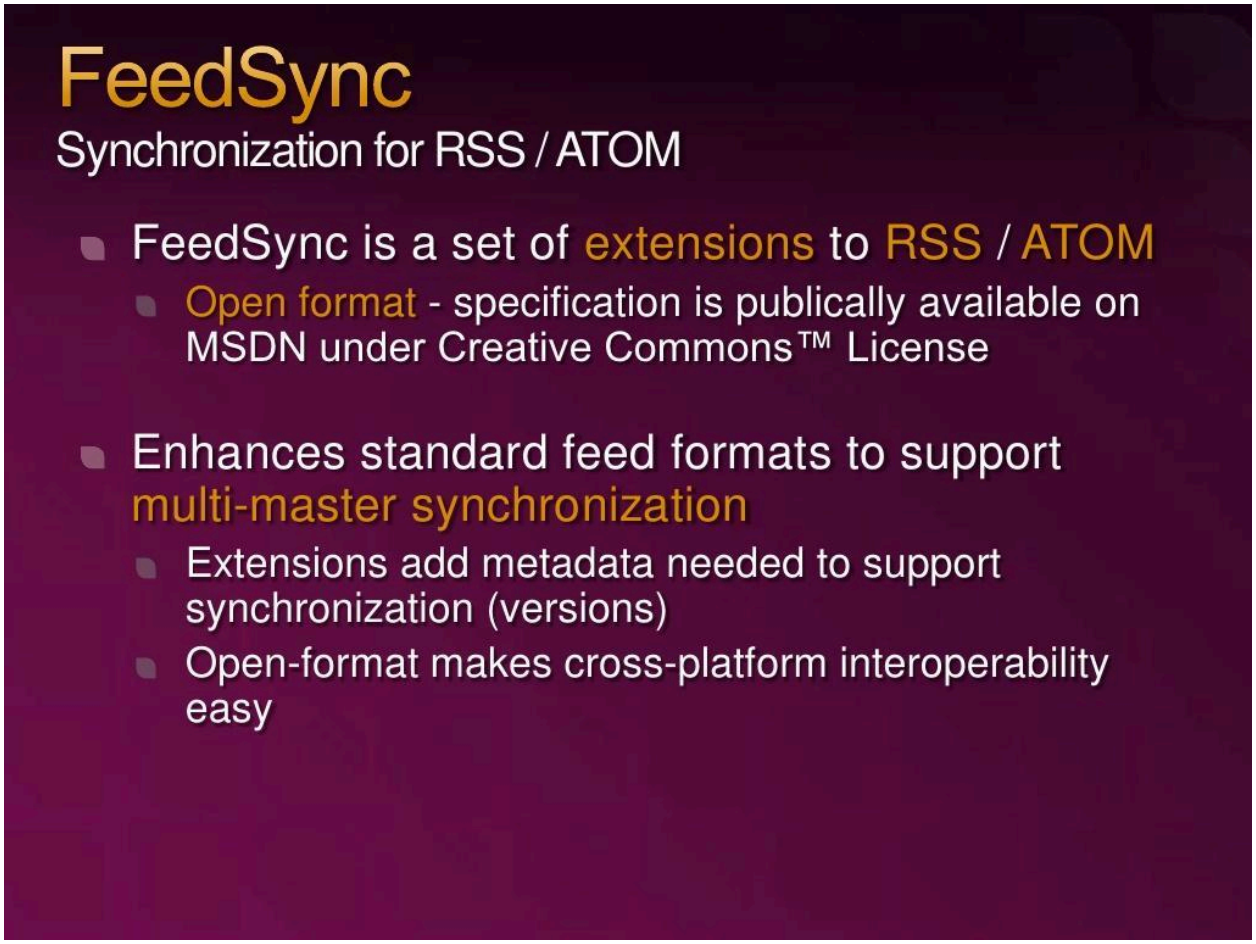


Figure 147. FeedSync Provider: Synchronization for RSS/ATOM

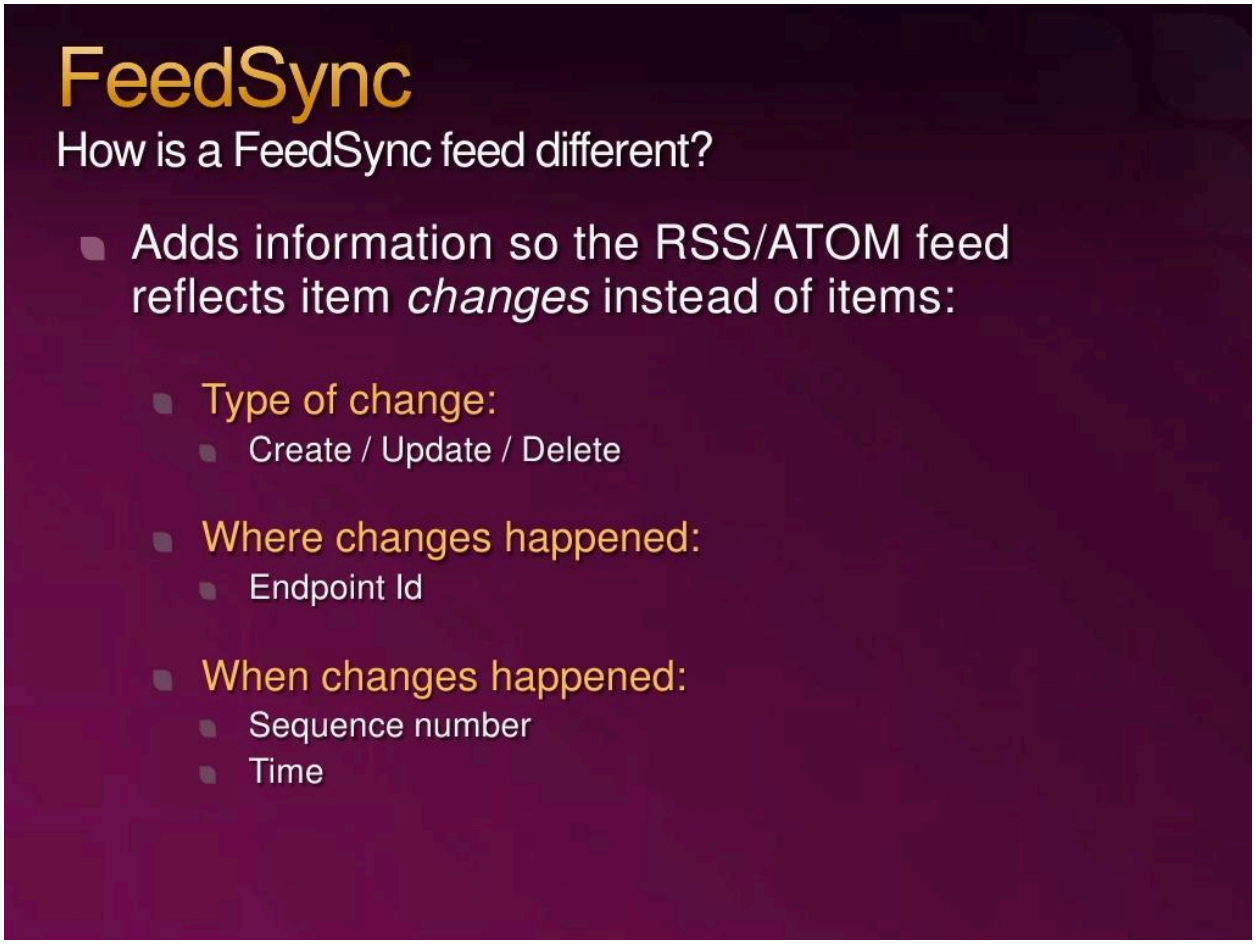


Figure 148. FeedSync Provider: Protocol

Microsoft OneDrive

TODO

Reference: https://docs.microsoft.com/en-us/onedrive/developer/sample-code?view=odsp-graph-online

TOSO

APPENDIX B – INDEX AND SEARCH TECHNOLOGIES: PRIOR ART

For the purposes of this whitepaper, Microsoft Search was chosen as the sole example of prior art because of:

- Its canonical indexing & search architecture
- Ease of availability of plentiful documentation

Microsoft Search

TODO

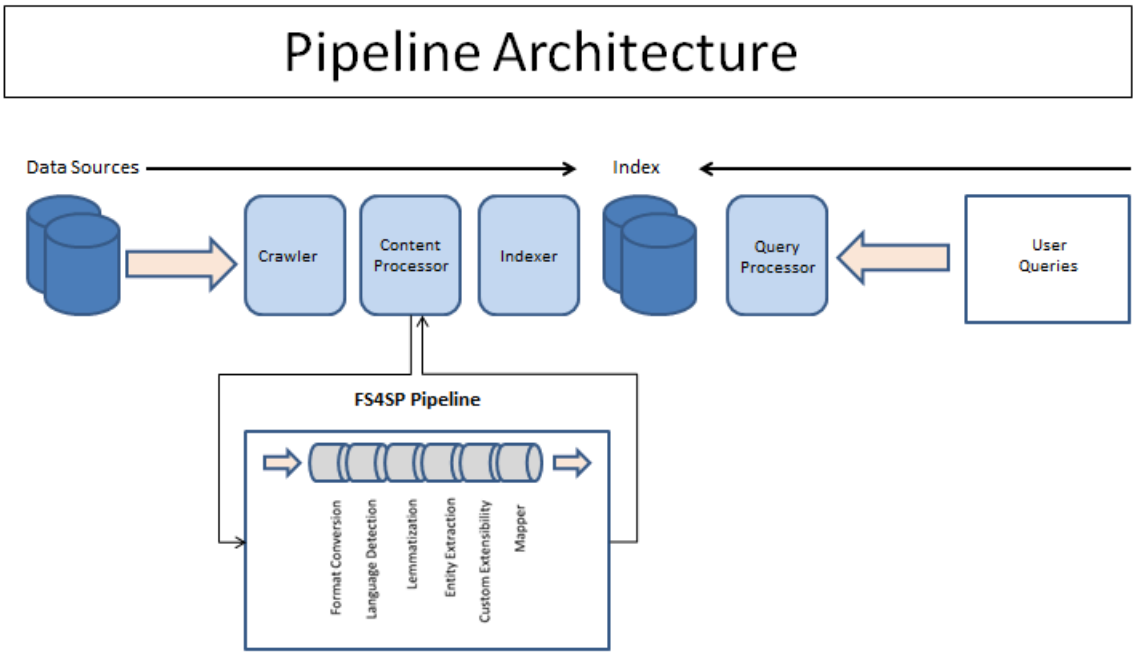


Figure 149. Microsoft FAST Search Pipeline Architecture

TODO

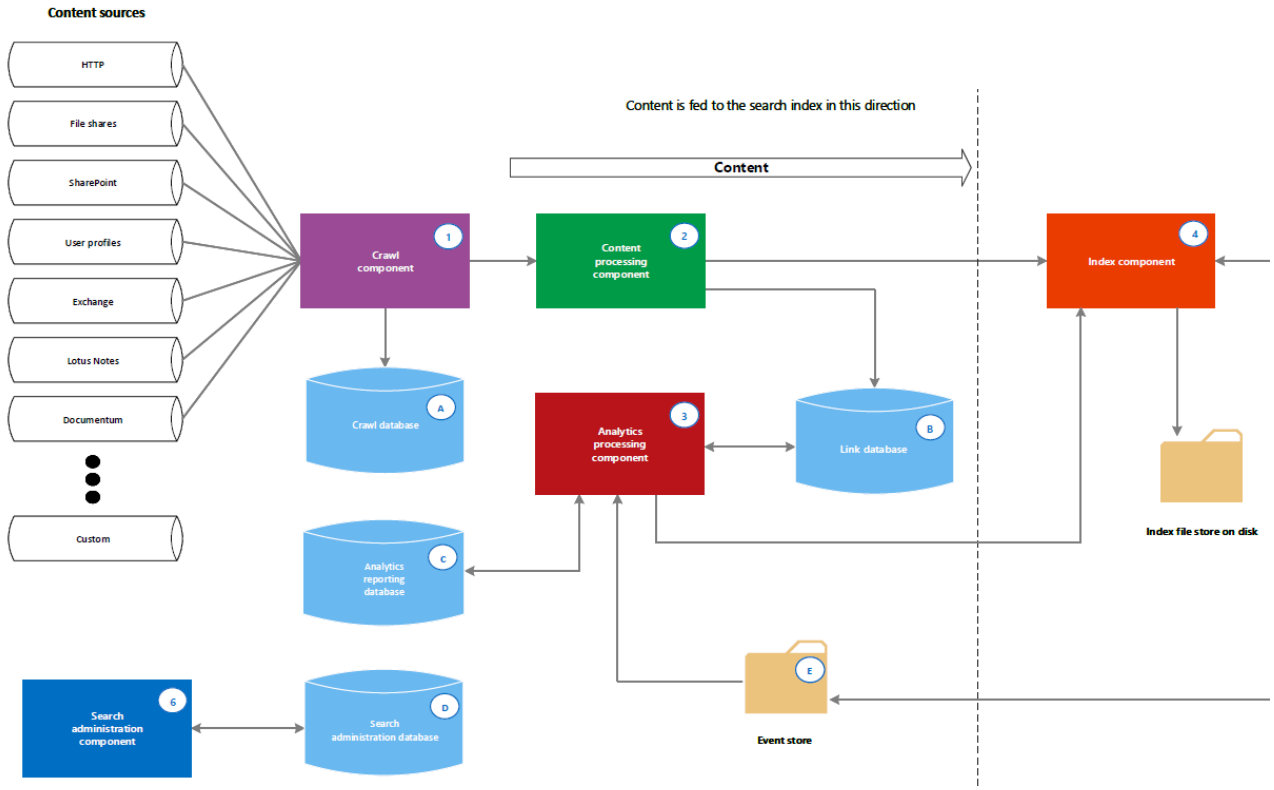


Figure 150. Microsoft Search: Indexing Pipeline

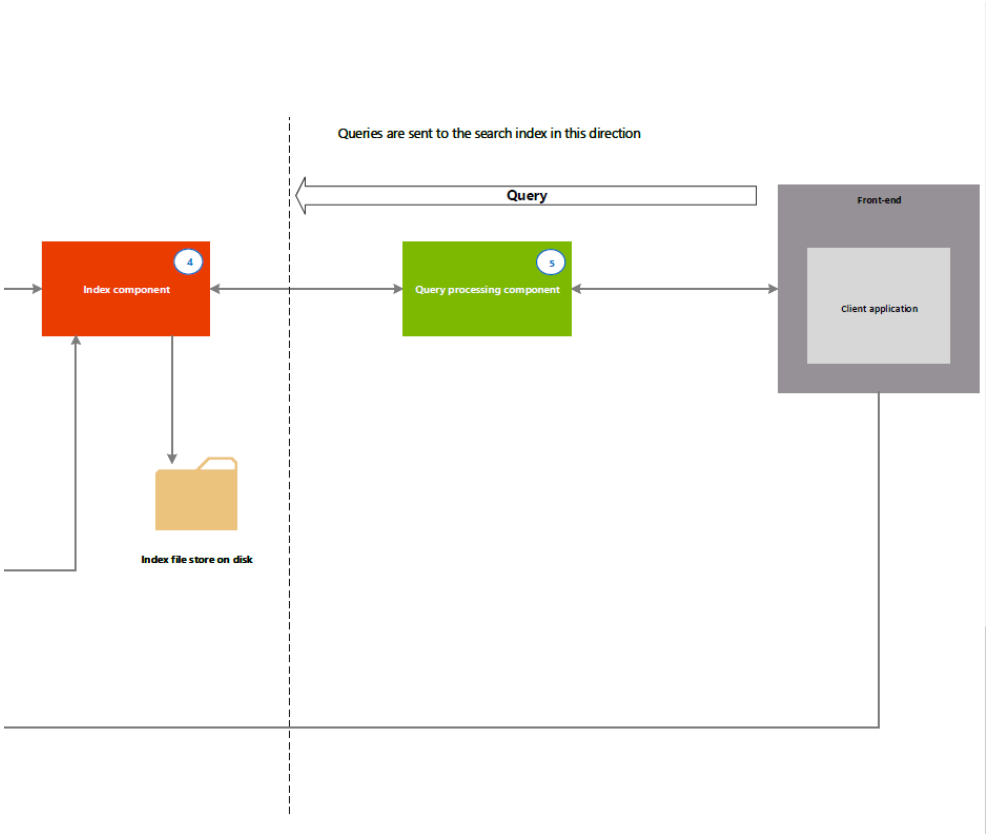


Figure 151. Microsoft Search: Query Processing

TODO

Other Examples

Lucene Search

Reference: http://placetoshareall.blogspot.com/2017/09/lucene-architecture-image.html

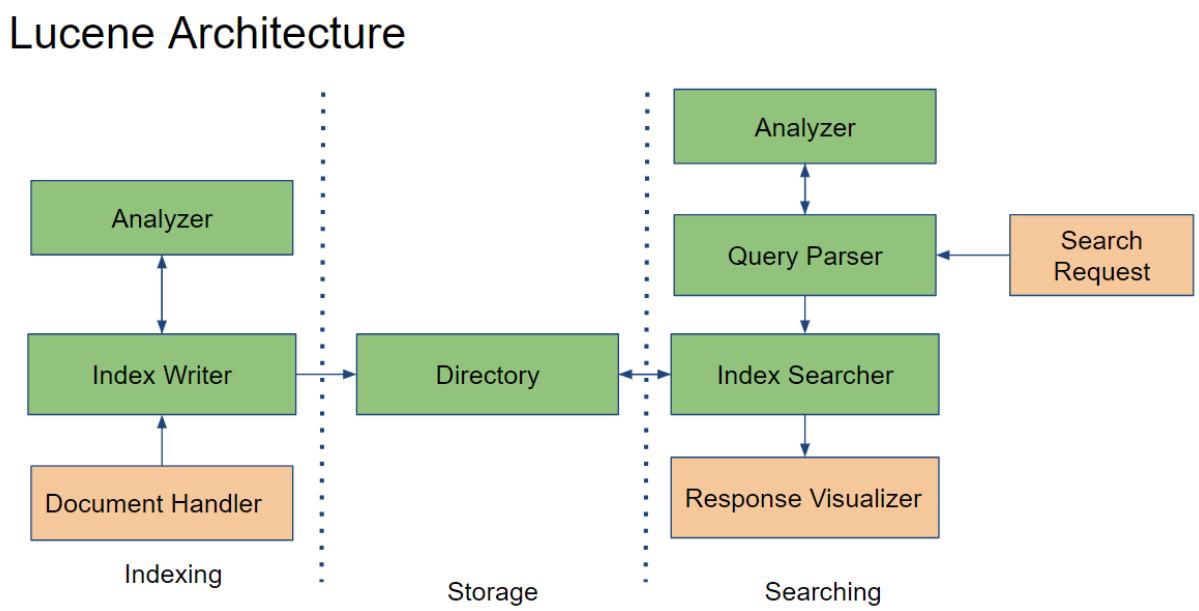


Figure 152. Lucene Index and Search Architecture

APPENDIX C – TRUSTED DIGITAL ASSISTANT: A CONCEPTUAL DESIGN

"HPEC" Trusted Digital Assistant

TODO

HPEC App Conceptual Architecture

TODO

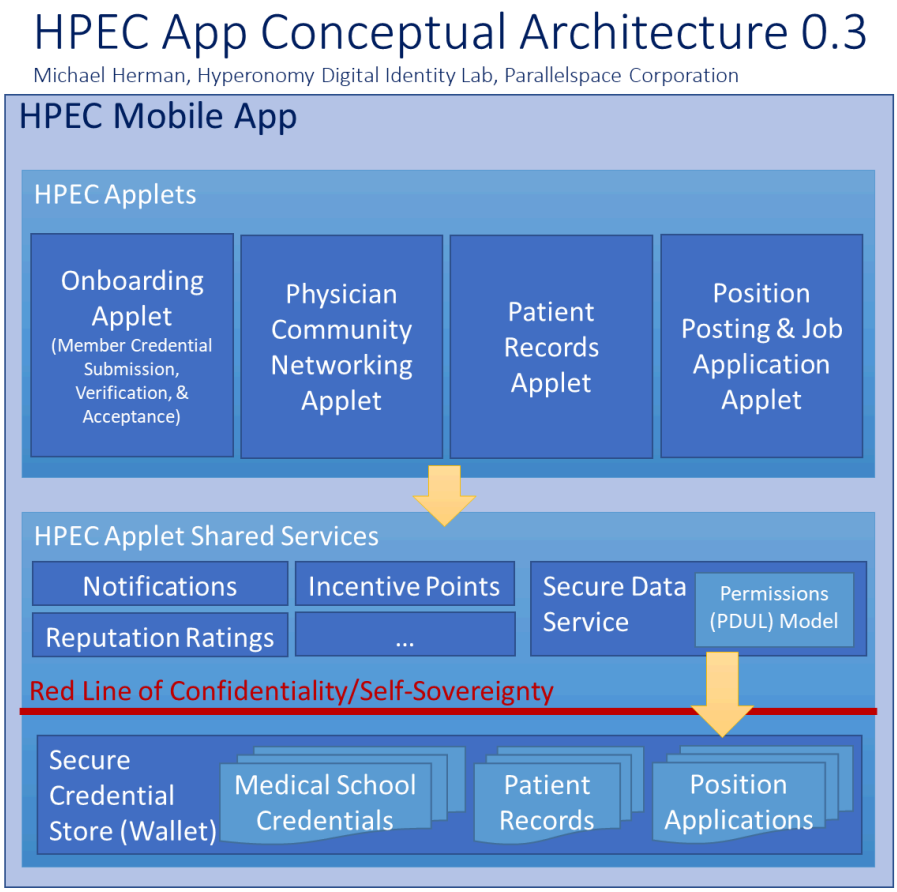


Figure 153. "HPEC" Trusted Digital Assistant: Conceptual Architecture

Mapping HPEC App Conceptual Architecture to the TCS Stack

TODO

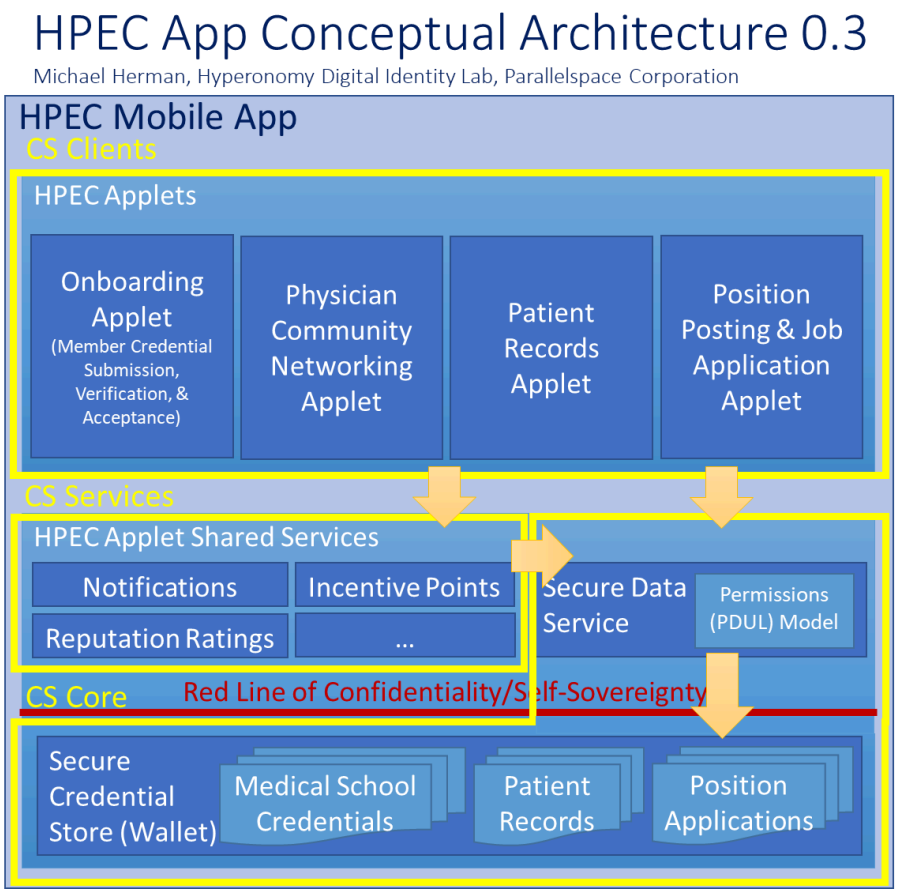


Figure 154. "HPEC" Trusted Digital Assistant mapped to the TCS Stack

APPENDIX D – TCS-ARMS ARCHIMATE SAMPLE

TODO

ArchiMate Modeling Notation

> *The ArchiMate® modeling language is an open and independent Enterprise Architecture standard that supports the description, analysis, and visualization of architecture within and across business domains. ArchiMate is one of the open standards hosted by The Open Group® and is fully aligned with TOGAF®. ArchiMate aids stakeholders in assessing the impact of design choices and changes.*
>
> [Archi open source modeler website (https://www.archimatetool.com/)]

The ArchiMate modeling notation is defined by the Open Group's ArchiMate Specification (current version 3.1). Quoting from https://www.opengroup.org/archimate-home,

> *The ArchiMate Specification, a standard of The Open Group, is an open and independent modeling language for Enterprise Architecture that is supported by different tool vendors and consulting firms. The ArchiMate Specification provides a common language and elements to enable enterprise, business, solution, and technology architects, business analysts and modelers, and software engineers to describe, analyze, and visualize the relationships among business domains in an unambiguous way.*

A partial set of ArchiMate symbols for both entities and relationships appear in the figure below. In this architecture variations presented in this whitepaper, only about 6 of these symbols (mostly from the Application Architecture domain) are used.

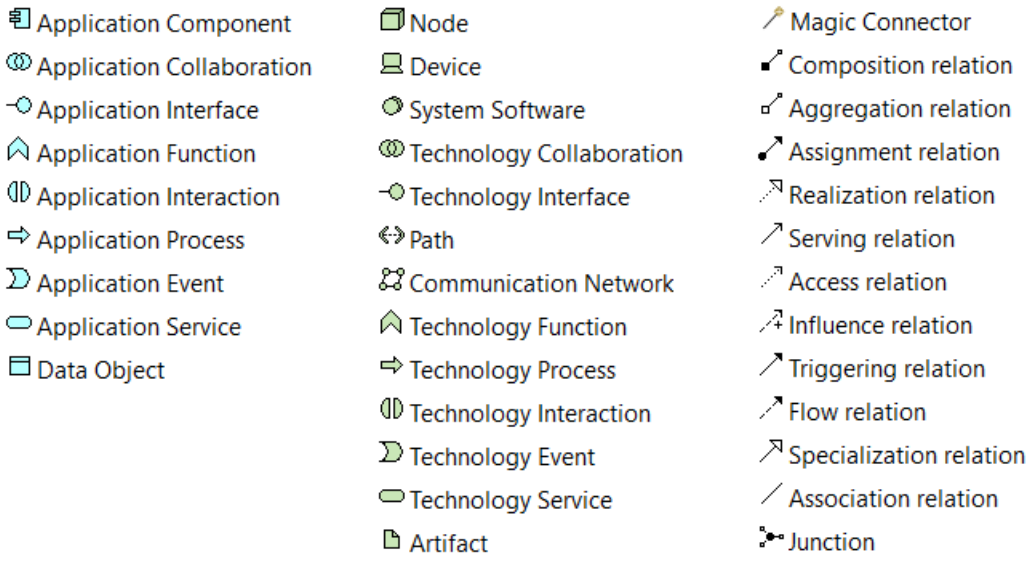


Figure 155. ArchiMate Symbols: (a) Application, (b) Technology, and (c) Relationships

The list of symbols used in this whitepaper includes:

- Technology domain

    - Artifact (used in Layer A Trusted Content Storage Kernel)
- Application domain
    - Application Component (software implementation)
    - Application Service (public capabilities defined by an API definition)
    - Application Interface (public API definition)
    - Application Function (component capabilities)
    - Application Event (content change detection trigger)

TCS-ARMs ArchiMate Sample

The following figure is an illustration of a repeating pattern of 5 of these ArchiMate symbols being used to define, implement, and expose a software service.

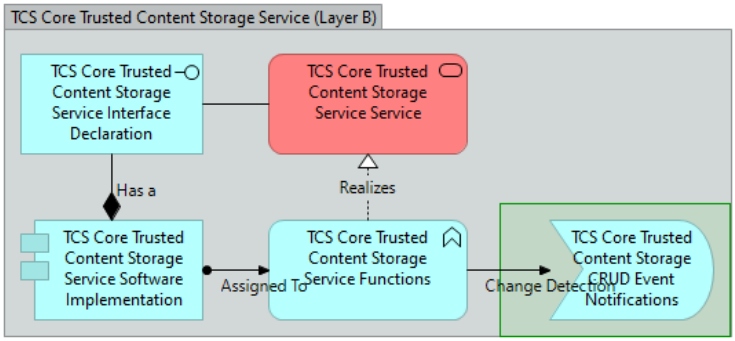


Figure 156. TCS-ARMs ArchiMate Sample [26]

Looking specifically at Layer B Trusted Content Storage Service, the “service” is comprised of 4 elements:

- Application Component (software implementation)
- Application Service (public capabilities defined by an API definition)
- Application Interface (public API definition)
- Application Function (component capabilities)
- Application Event (content change detection trigger) – optionally

The Application Component represents the software implementation of the Layer B Trusted Content Storage Service (the lower-left component in the diagram below). The Application Component has or composes with an Application Interface: the Layer B Trusted Content Storage Service Interface Declaration (the top-left component in the diagram below). In addition, the Application Component is assigned to an Application Function element (bottom-center), Layer B Trusted Content Storage Service Functions, representing the functionality exposed by the Application Service (top-center), the Layer B Trusted Content Storage Service.

The functionality exposed by the Layer B Trusted Content Storage Service is declared by the Application Interface (top-left), Layer B Trusted Content Storage Service Interface Declaration, described earlier.

NOTE: These 4 components represent a repeating pattern that is re-used throughout the ARMs presented in this document.

Lastly, an Application Event (lower-right corner) is used to represent the Layer B Trusted Content Storage CRUD Event Notifications.

Archi Open Source Modeler for ArchiMate

The Archi® modeling toolkit is targeted toward all levels of Enterprise Architects and Modelers. It provides a low cost to entry solution to users who may be making their first steps in the ArchiMate modeling language, or who are looking for an open-source, cross-platform ArchiMate modeling tool for their company or institution and wish to engage with the language within a TOGAF® or other Enterprise Architecture framework.

[Archi open source modeler website (https://www.archimatetool.com/)]

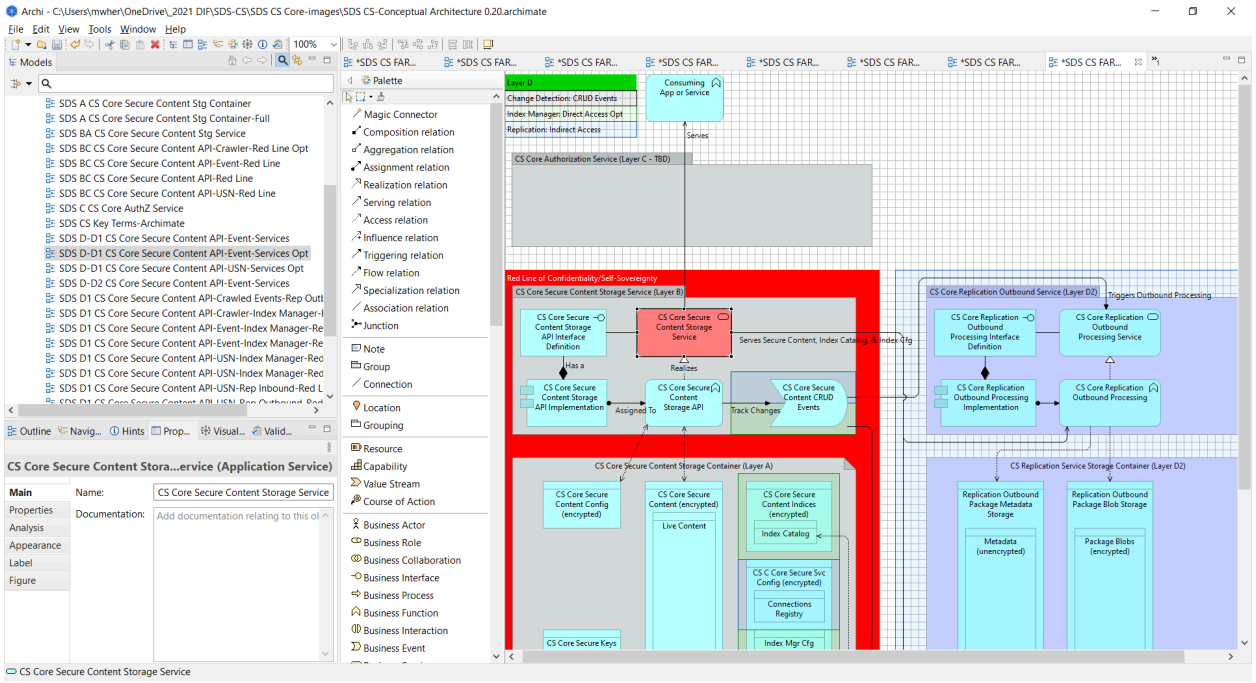


Figure 157. Archi Open Source Modeler for ArchiMate

APPENDIX E – MIT LICENSE